

HTML

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</
body>
</html>
```

- The `<!DOCTYPE html>` declaration defines this document to be HTML5
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the document
- The `<title>` element specifies a title for the document
- The `<body>` element contains the visible page content
- The `<h1>` element defines a large heading
- The `<p>` element defines a paragraph

-
- HTML tags normally come **in pairs** like `<p>` and `</p>`
 - The first tag in a pair is the **start tag**, the second tag is the **end tag**
 - The end tag is written like the start tag, but with a **forward slash** inserted before the tag name
 - **HTML Headings**
 - HTML headings are defined with the `<h1>` to `<h6>` tags.
 - `<h1>` defines the most important heading. `<h6>` defines the least important heading:

- **Example**
 - `<h1>`This is heading 1`</h1>`
 - `<h2>`This is heading 2`</h2>`
 - `<h3>`This is heading 3`</h3>`
-

HTML Lists

HTML List Example

An Unordered List:

- Item
- Item
- Item
- Item

An Ordered List:

1. First item
 2. Second item
 3. Third item
 4. Fourth item
-

Unordered HTML List

An unordered list starts with the `` tag. Each list item starts with the `` tag.

The list items will be marked with bullets (small black circles) by default:

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Unordered HTML List - Choose List Item Marker

The CSS **list-style-type** property is used to define the style of the list item marker:

Value	Description
disc	Sets the list item marker to a bullet (default)
circle	Sets the list item marker to a circle
square	Sets the list item marker to a square
none	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc">  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle">  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>
```

Example - Square

```
<ul style="list-style-type:square">  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>
```

Example - None

```
<ul style="list-style-type:none">  
  <li>Coffee</li>  
  <li>Tea</li>  
  <li>Milk</li>  
</ul>
```

Ordered HTML List

An ordered list starts with the `` tag. Each list item starts with the `` tag.

The list items will be marked with numbers by default:

Example

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Ordered HTML List - The Type Attribute

The **type** attribute of the `` tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
```

```
<li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

HTML also supports description lists.

A description list is a list of terms, with a description of each term.

The **<dl>** tag defines the description list, the **<dt>** tag defines the term (name), and the **<dd>** tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Nested HTML Lists

List can be nested (lists inside lists):

Example

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Note: List items can contain new list, and other HTML elements, like images and links, etc.

Horizontal Lists

HTML lists can be styled in many different ways with CSS.

One popular way is to style a list horizontally, to create a menu:

Example

```
<!DOCTYPE html>
<html>
<head>
<style>
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333333;
}

li {
  float: left;
}
```

```
li a {
  display: block;
  color: white;
  text-align: center;
  padding: 16px;
  text-decoration: none;
}

li a:hover {
  background-color: #111111;
}
</style>
</head>
<body>

<ul>
  <li><a href="#home">Home</a></li>
  <li><a href="#news">News</a></li>
  <li><a href="#contact">Contact</a></li>
  <li><a href="#about">About</a></li>
</ul>

</body>
</html>
```

HTML Links - Hyperlinks

HTML links are hyperlinks.

You can click on a link and jump to another document.

When you move the mouse over a link, the mouse arrow will turn into a little hand.

Note: A link does not have to be text. It can be an image or any other HTML element.

HTML Links - Syntax

In HTML, links are defined with the `<a>` tag:

```
<a href="url">link text</a>
```

Example

`Visit our HTML tutorial`

The **href** attribute specifies the destination address (https://www.w3schools.com/html/) of the link.

The **link text** is the visible part (Visit our HTML tutorial).

Clicking on the link text will send you to the specified address.

Note: Without a forward slash on subfolder addresses, you might generate two requests to the server. Many servers will automatically add a forward slash to the address, and then create a new request.

Local Links

The example above used an absolute URL (A full web address).

A local link (link to the same web site) is specified with a relative URL (without http://www....).

Example

`HTML Images`

HTML Table Example

Company	Contact	Country
Alfreds Futterkiste	Maria Anders	Germany
Centro comercial Moctezuma	Francisco Chang	Mexico
Ernst Handel	Roland Mendel	Austria
Island Trading	Helen Bennett	UK
Laughing Bacchus Winecellars	Yoshi Tannamuri	Canada
Magazzini Alimentari Riuniti	Giovanni Rovelli	Italy

Defining an HTML Table

An HTML table is defined with the **<table>** tag.

Each table row is defined with the **<tr>** tag. A table header is defined with the **<th>** tag. By default, table headings are bold and centered. A table data/cell is defined with the **<td>** tag.

Example


```
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Note: The <td> elements are the data containers of the table.
They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

HTML Table - Adding a Border

If you do not specify a border for the table, it will be displayed without borders.

A border is set using the CSS **border** property:

Example

```
table, th, td {
  border: 1px solid black;
}
```

Remember to define borders for both the table and the table cells.

HTML Table - Collapsed Borders

If you want the borders to collapse into one border, add the CSS **border-collapse** property:

Example

```
table, th, td {
  border: 1px solid black;
```

```
border-collapse: collapse;
}
```

HTML Table - Adding Cell Padding

Cell padding specifies the space between the cell content and its borders.

If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS **padding** property:

Example

```
th, td {
padding: 15px;
}
```

HTML Table - Left-align Headings

By default, table headings are bold and centered.

To left-align the table headings, use the CSS **text-align** property:

Example

```
th {
text-align: left;
}
```

HTML Table - Adding Border Spacing

Border spacing specifies the space between the cells.

To set the border spacing for a table, use the CSS **border-spacing** property:

Example

```
table {
border-spacing: 5px;
}
```

Note: If the table has collapsed borders, border-spacing has no effect.

HTML Table - Cells that Span Many Columns

To make a cell span more than one column, use the **colspan** attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Cells that Span Many Rows

To make a cell span more than one row, use the **rowspan** attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Adding a Caption

To add a caption to a table, use the **<caption>** tag:

Example

```
<table style="width:100%">
  <caption>Monthly savings</caption>
```

```

<tr>
  <th>Month</th>
  <th>Savings</th>
</tr>
<tr>
  <td>January</td>
  <td>$100</td>
</tr>
<tr>
  <td>February</td>
  <td>$50</td>
</tr>
</table>

```

Note: The <caption> tag must be inserted immediately after the <table> tag.

A Special Style for One Table

To define a special style for a special table, add an **id** attribute to the table:

Example

```

<table id="t01">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>

```

Now you can define a special style for this table:

```

table#t01 {
  width: 100%;
  background-color: #f1f1c1;
}

```

And add more styles:

```

table#t01 tr:nth-child(even) {
  background-color: #eee;
}

```

```
table#t01 tr:nth-child(odd) {  
    background-color: #fff;  
}  
table#t01 th {  
    color: white;  
    background-color: black;  
}
```

Summary

- Use the HTML **<table>** element to define a table
 - Use the HTML **<tr>** element to define a table row
 - Use the HTML **<td>** element to define a table data
 - Use the HTML **<th>** element to define a table heading
 - Use the HTML **<caption>** element to define a table caption
 - Use the CSS **border** property to define a border
 - Use the CSS **border-collapse** property to collapse cell borders
 - Use the CSS **padding** property to add padding to cells
 - Use the CSS **text-align** property to align cell text
 - Use the CSS **border-spacing** property to set the spacing between cells
 - Use the **colspan** attribute to make a cell span many columns
 - Use the **rowspan** attribute to make a cell span many rows
 - Use the **id** attribute to uniquely define one table
-

HTML Table Tags

Tag	Description
<u><table></u>	Defines a table
<u><th></u>	Defines a header cell in a table
<u><tr></u>	Defines a row in a table
<u><td></u>	Defines a cell in a table
<u><caption></u>	Defines a table caption
<u><colgroup></u>	Specifies a group of one or more columns in a table for formatting
<u><col></u>	Specifies column properties for each column within a <colgroup> element
<u><thead></u>	Groups the header content in a table
<u><tbody></u>	Groups the body content in a table
<u><tfoot></u>	Groups the footer content in a table

The <form> Element

The HTML **<form>** element defines a form that is used to collect user input:

`<form>`

.

form elements

.

`</form>`

An HTML form contains **form elements**.

Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The `<input>` Element

The `<input>` element is the most important form element.

The `<input>` element can be displayed in several ways, depending on the **type** attribute.

Here are some examples:

Type	Description
<code><input type="text"></code>	Defines a one-line text input field
<code><input type="radio"></code>	Defines a radio button (for selecting one of many choices)
<code><input type="submit"></code>	Defines a submit button (for submitting the form)

Text Input

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

Example

`<form>`

First name:`
`

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

Last name:`
`

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

`</form>`

Input Type Text

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

Example

```
<form>
First name:<br>
<input type="text" name="firstname"><br>
Last name:<br>
<input type="text" name="lastname">
</form>
```

Input Type Password

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

Example

```
<form>
User name:<br>
<input type="text" name="username"><br>
User password:<br>
<input type="password" name="psw">
</form>
```

characters in a password field are masked (shown as asterisks or circles).

Input Type Submit

`<input type="submit">` defines a button for **submitting** form data to a **form-handler**.

The form-handler is typically a server page with a script for processing input data.

The form-handler is specified in the form's **action** attribute:

Example

```
<form action="/action_page.php">
First name:<br>
<input type="text" name="firstname" value="Mickey"><br>
Last name:<br>
<input type="text" name="lastname" value="Mouse"><br><br>
<input type="submit" value="Submit">
</form>
```

If you omit the submit button's value attribute, the button will get a default text:

Example

```
<form action="/action_page.php">
  First name:<br>
  <input type="text" name="firstname" value="Mickey"><br>
  Last name:<br>
  <input type="text" name="lastname" value="Mouse"><br><br>
  <input type="submit">
</form>
```

The value Attribute

The **value** attribute specifies the initial value for an input field:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John">
</form>
```

HTML frames 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.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages –

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back* button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

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 – The <frame> tag deprecated in HTML5. Do not use this element.

Example

Following is the example to create three horizontal frames –

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset rows = "10%,80%,10%">
    <frame name = "top" src = "/html/top_frame.htm" />
    <frame name = "main" src = "/html/main_frame.htm" />
    <frame name = "bottom" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>

  </frameset>

</html>
```

Example

Let's put the above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically –

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Frames</title>
  </head>

  <frameset cols = "25%,50%,25%">
    <frame name = "left" src = "/html/top_frame.htm" />
    <frame name = "center" src = "/html/main_frame.htm" />
    <frame name = "right" src = "/html/bottom_frame.htm" />

    <noframes>
      <body>Your browser does not support frames.</body>
    </noframes>

  </frameset>

</html>
```

Division:–

A section in a document that will be displayed in blue:

```
<div style="color:#0000FF">
  <h3>This is a heading</h3>
```

`<p>This is a paragraph.</p>`
`</div>`

Definition and Usage

The `<div>` tag defines a division or a section in an HTML document.

The `<div>` tag is used to group block-elements to format them with CSS.

Global Attributes

The `<div>` tag also supports the [Global Attributes in HTML](#).

Event Attributes

The `<div>` tag also supports the [Event Attributes in HTML](#).

What is CSS

CSS is an acronym stands for Cascading Style Sheets. It is a style sheet language which is used to describe the look and formatting of a document written in markup language. It provides an additional feature to HTML. It is generally used with HTML to change the style of web pages and user interfaces. It can also be used with any kind of XML documents including plain XML, SVG and XUL.

CSS is used along with HTML and JavaScript in most websites to create user interfaces for web applications and user interfaces for many mobile applications.

What does CSS do

- You can add new looks to your old HTML documents.
 - You can completely change the look of your website with only a few changes in CSS code.
-

Why use CSS

These are the three major benefits of CSS:

1) Solves a big problem

Before CSS, tags like font, color, background style, element alignments, border and size had to be repeated on every web page. This was a very long process. For example: If you are developing a large website where fonts and color information are added on every single page, it will be become a long and expensive process. CSS was created to solve this problem. It was a W3C recommendation.

2) Saves a lot of time

CSS style definitions are saved in external CSS files so it is possible to change the entire website by changing just one file.

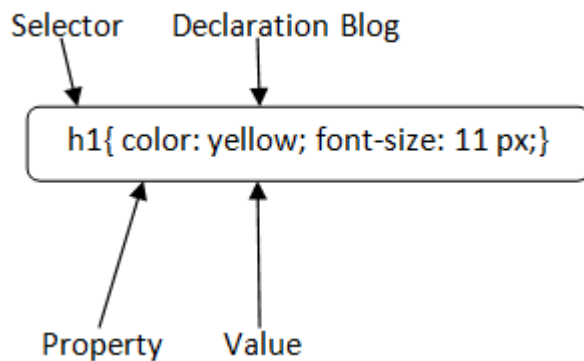
3) Provide more attributes

CSS provides more detailed attributes than plain HTML to define the look and feel of

the website.

CSS Syntax

A CSS rule set contains a selector and a declaration block.



Selector: Selector indicates the HTML element you want to style. It could be any tag like `<h1>`, `<title>` etc.

Declaration Block: The declaration block can contain one or more declarations separated by a semicolon. For the above example, there are two declarations:

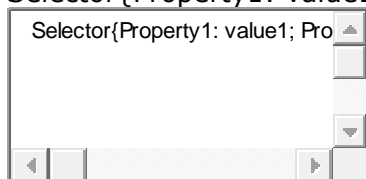
1. `color: yellow;`
2. `font-size: 11 px;`

Each declaration contains a property name and value, separated by a colon.

Property: A Property is a type of attribute of HTML element. It could be color, border etc.

Value: Values are assigned to CSS properties. In the above example, value "yellow" is assigned to color property.

1. `Selector{Property1: value1; Property2: value2;;}`



CSS Selector

CSS selectors are used *to select the content you want to style*. Selectors are the part of CSS rule set. CSS selectors select HTML elements according to its id, class, type, attribute etc.

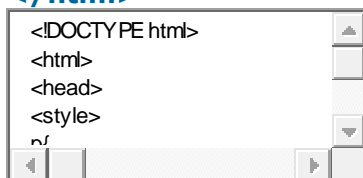
There are several different types of selectors in CSS.

1. CSS Element Selector
2. CSS Id Selector
3. CSS Class Selector
4. CSS Universal Selector
5. CSS Group Selector

1) CSS Element Selector

The element selector selects the HTML element by name.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p{`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<p>`This style will be applied on every paragraph.`</p>`
13. `<p id="para1">`Me too!`</p>`
14. `<p>`And me!`</p>`
15. `</body>`
16. `</html>`



Test it Now

Output:

This style will be applied on every paragraph.

Me too!

And me!

2) CSS Id Selector

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique within the page so it is chosen to select a single, unique element.

It is written with the hash character (#), followed by the id of the element.

Let's take an example with the id "para1".

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `#para1 {`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<p id="para1">Hello Javatpoint.com</p>`
13. `<p>This paragraph will not be affected.</p>`
14. `</body>`
15. `</html>`



Test it Now

Output:

This paragraph will not be affected.

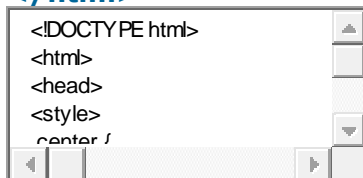
3) CSS Class Selector

The class selector selects HTML elements with a specific class attribute. It is used with a period character . (full stop symbol) followed by the class name.

Note: A class name should not be started with a number.

Let's take an example with a class "center".

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `.center {`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<h1 class="center">This heading is blue and center-aligned.</h1>`
13. `<p class="center">This paragraph is blue and center-aligned.</p>`
14. `</body>`
15. `</html>`



Test it Now

Output:

This heading is blue and center-aligned.

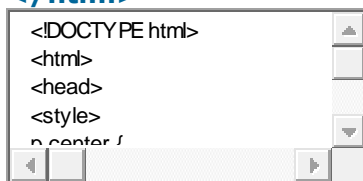
This paragraph is blue and center-aligned.

CSS Class Selector for specific element

If you want to specify that only one specific HTML element should be affected then you should use the element name with class selector.

Let's see an example.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p.center {`
6. `text-align: center;`
7. `color: blue;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<h1 class="center">`This heading is not affected`</h1>`
13. `<p class="center">`This paragraph is blue and center-aligned.`</p>`
14. `</body>`
15. `</html>`



Test it Now

Output:

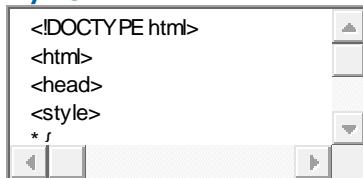
This heading is not affected

This paragraph is blue and center-aligned.

4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `* {`
6. `color: green;`
7. `font-size: 20px;`
8. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<h2>`This is heading`</h2>`
13. `<p>`This style will be applied on every paragraph.`</p>`
14. `<p id="para1">`Me too!`</p>`
15. `<p>`And me!`</p>`
16. `</body>`
17. `</html>`



Test it Now

Output:

This is heading

This style will be applied on every paragraph.

Me too!

And me!

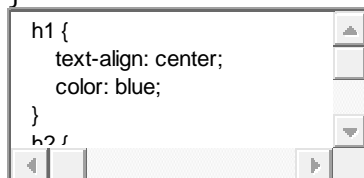
5) CSS Group Selector

The grouping selector is used to select all the elements with the same style definitions.

Grouping selector is used to minimize the code. Commas are used to separate each selector in grouping.

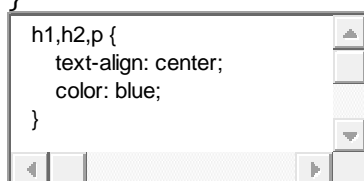
Let's see the CSS code without group selector.

1. h1 {
2. text-align: center;
3. color: blue;
4. }
5. h2 {
6. text-align: center;
7. color: blue;
8. }
9. p {
10. text-align: center;
11. color: blue;
12. }



As you can see, you need to define CSS properties for all the elements. It can be grouped in following ways:

1. h1,h2,p {
2. text-align: center;
3. color: blue;
4. }



Let's see the full example of CSS group selector.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `h1, h2, p {`
5. `text-align: center;`
7. `color: blue;`
3. `}`
9. `</style>`
10. `</head>`
11. `<body>`
12. `<h1>Hello Javatpoint.com</h1>`
13. `<h2>Hello Javatpoint.com (In smaller font)</h2>`
14. `<p>This is a paragraph.</p>`
15. `</body>`
16. `</html>`



Test it Now

Output:

Hello Javatpoint.com

Hello Javatpoint.com (In smaller font)

This is a paragraph.

How to add CSS

CSS is added to HTML pages to format the document according to information in the style sheet. There are three ways to insert CSS in HTML documents.

1. Inline CSS
2. Internal CSS
3. External CSS

1) Inline CSS

Inline CSS is used to apply CSS on a single line or element.

For example:

1. `<p style="color:blue">Hello CSS</p>`



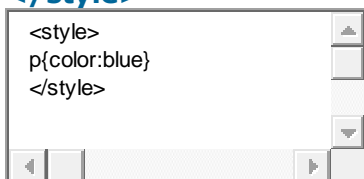
For more visit here: [Inline CSS](#)

2) Internal CSS

Internal CSS is used to apply CSS on a single document or page. It can affect all the elements of the page. It is written inside the style tag within head section of html.

For example:

1. `<style>`
2. `p{color:blue}`
3. `</style>`



For more visit here: [Internal CSS](#)

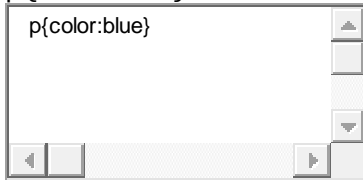
3) External CSS

External CSS is used to apply CSS on multiple pages or all pages. Here, we write

all the CSS code in a css file. Its extension must be .css for example style.css.

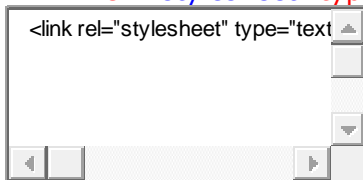
For example:

1. `p{color:blue}`



You need to link this style.css file to your html pages like this:

1. `<link rel="stylesheet" type="text/css" href="style.css">`



The link tag must be used inside head section of html.

Inline CSS

We can apply CSS in a single element by inline CSS technique.

The inline CSS is also a method to insert style sheets in HTML document. This method mitigates some advantages of style sheets so it is advised to use this method sparingly.

If you want to use inline CSS, you should use the style attribute to the relevant tag.

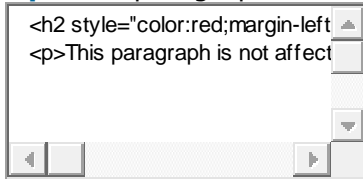
Syntax:

1. `<htmltag style="cssproperty1:value; cssproperty2:value;"> </htmltag>`



Example:

1. `<h2 style="color:red;margin-left:40px;">Inline CSS is applied on this heading.</h2>`
2. `<p>This paragraph is not affected.</p>`



Test it Now

Output:

Inline CSS is applied on this heading.

This paragraph is not affected.

Disadvantages of Inline CSS

- You cannot use quotations within inline CSS. If you use quotations the browser will interpret this as an end of your style value.
- These styles cannot be reused anywhere else.
- These styles are tough to be edited because they are not stored at a single place.
- It is not possible to style pseudo-codes and pseudo-classes with inline CSS.
- Inline CSS does not provide browser cache advantages.

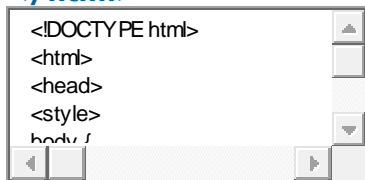
Internal CSS

The internal style sheet is used to add a unique style for a single document. It is defined in `<head>` section of the HTML page inside the `<style>` tag.

Example:

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`

5. body {
5. background-color: linen;
7. }
3. h1 {
9. color: red;
10. margin-left: 80px;
11. }
12. </style>
13. </head>
14. <body>
15. <h1>The internal style sheet is applied on this heading.</h1>
16. <p>This paragraph will not be affected.</p>
17. </body>
18. </html>



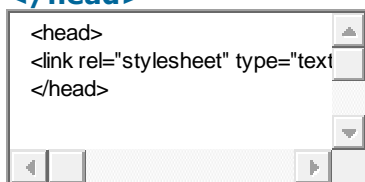
External CSS

The external style sheet is generally used when you want to make changes on multiple pages. It is ideal for this condition because it facilitates you to change the look of the entire web site by changing just one file.

It uses the <link> tag on every pages and the <link> tag should be put inside the head section.

Example:

1. <head>
2. <link rel="stylesheet" type="text/css" href="mystyle.css">
3. </head>

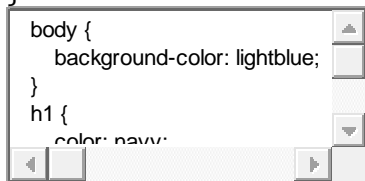


The external style sheet may be written in any text editor but must be saved with a .css extension. This file should not contain HTML elements.

Let's take an example of a style sheet file named "mystyle.css".

File: mystyle.css

1. body {
2. background-color: lightblue;
3. }
4. h1 {
5. color: navy;
5. margin-left: 20px;
7. }



Note: You should not use a space between the property value and the unit. For example: It should be margin-left:20px not margin-left:20 px.

CSS Font

CSS Font property is used to control the look of texts. By the use of CSS font property you can change the text size, color, style and more. You have already studied how to make text bold or underlined. Here, you will also know how to resize your font using percentage.

These are some important font attributes:

1. **CSS Font color:** This property is used to change the color of the text. (standalone attribute)
2. **CSS Font family:** This property is used to change the face of the font.
3. **CSS Font size:** This property is used to increase or decrease the size of the font.
4. **CSS Font style:** This property is used to make the font bold, italic or oblique.

5. **CSS Font variant:** This property creates a small-caps effect.
 6. **CSS Font weight:** This property is used to increase or decrease the boldness and lightness of the font.
-

1) CSS Font Color

CSS font color is a standalone attribute in CSS although it seems that it is a part of CSS fonts. It is used to change the color of the text.

There are three different formats to define a color:

- By a color name
- By hexadecimal value
- By RGB

In the above example, we have defined all these formats.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. body {
6.     font-size: 100%;
7. }
8. h1 { color: red; }
9. h2 { color: #9000A1; }
10. p { color:rgb(0, 220, 98); }
11.}
12.</style>
13.</head>
14.<body>
15.<h1>This is heading 1</h1>
16.<h2>This is heading 2</h2>
17.<p>This is a paragraph.</p>
18.</body>
19.</html>
```

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
```

Test it Now

Output:

This is heading 1

This is heading 2

This is a paragraph.

2) CSS Font Family

CSS font family can be divided in two types:

- Generic family: It includes Serif, Sans-serif, and Monospace.
- Font family: It specifies the font family name like Arial, New Times Roman etc.

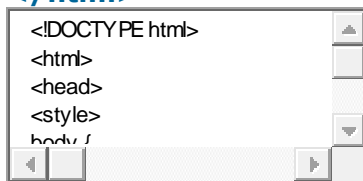
Serif: Serif fonts include small lines at the end of characters. Example of serif: Times new roman, Georgia etc.

Sans-serif: A sans-serif font doesn't include the small lines at the end of characters. Example of Sans-serif: Arial, Verdana etc.



1. `<!DOCTYPE html>`
2. `<html>`

3. **<head>**
4. **<style>**
5. body {
5. font-size: 100%;
7. }
3. h1 { font-family: sans-serif; }
9. h2 { font-family: serif; }
10. p { font-family: monospace; }
11. }
12. **</style>**
13. **</head>**
14. **<body>**
15. **<h1>**This heading is shown in sans-serif.**</h1>**
16. **<h2>**This heading is shown in serif.**</h2>**
17. **<p>**This paragraph is written in monospace.**</p>**
18. **</body>**
19. **</html>**



Test it Now

Output:

This heading is shown in sans-serif.

This heading is shown in serif.

This paragraph is written in monospace.

3) CSS Font Size

CSS font size property is used to change the size of the font.

These are the possible values that can be used to set the font size:

Font Size Value	Description
xx-small	used to display the extremely small text size.
x-small	used to display the extra small text size.
small	used to display small text size.
medium	used to display medium text size.
large	used to display large text size.
x-large	used to display extra large text size.
xx-large	used to display extremely large text size.
smaller	used to display comparatively smaller text size.
larger	used to display comparatively larger text size.
size in pixels or %	used to set value in percentage or in pixels.

1. `<html>`
2. `<head>`
3. `<title>`Practice CSS font-size property`</title>`
4. `</head>`
5. `<body>`
5. `<p style="font-size:xx-small;">` This font size is extremely small.`</p>`
7. `<p style="font-size:x-small;">` This font size is extra small`</p>`
3. `<p style="font-size:small;">` This font size is small`</p>`
9. `<p style="font-size:medium;">` This font size is medium. `</p>`
10. `<p style="font-size:large;">` This font size is large. `</p>`
11. `<p style="font-size:x-large;">` This font size is extra large. `</p>`
12. `<p style="font-size:xx-large;">` This font size is extremely large. `</p>`
13. `<p style="font-size:smaller;">` This font size is smaller. `</p>`
14. `<p style="font-size:larger;">` This font size is larger. `</p>`

15. `<p style="font-size:200%;">` This font size is set on 200%. `</p>`

16. `<p style="font-size:20px;">` This font size is 20 pixels. `</p>`

17. `</body>`

18. `</html>`



Test it Now

Output:

This font size is extremely small.

This font size is extra small

This font size is small

This font size is medium.

This font size is large.

This font size is extra large.

This font size is
extremely large.

This font size is smaller.

This font size is larger.

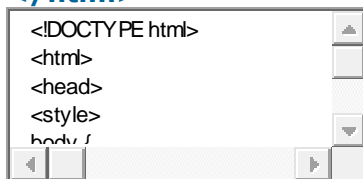
This font size is set on 200%.

This font size is 20 pixels.

4) CSS Font Style

CSS Font style property defines what type of font you want to display. It may be italic, oblique, or normal.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `body {`
6. `font-size: 100%;`
7. `}`
8. `h2 { font-style: italic; }`
9. `h3 { font-style: oblique; }`
10. `h4 { font-style: normal; }`
11. `}`
12. `</style>`
13. `</head>`
14. `<body>`
15. `<h2>`This heading is shown in italic font.`</h2>`
16. `<h3>`This heading is shown in oblique font.`</h3>`
17. `<h4>`This heading is shown in normal font.`</h4>`
18. `</body>`
19. `</html>`



Test it Now

Output:

This heading is shown in italic font.

This heading is shown in oblique font.

This heading is shown in normal font.

5) CSS Font Variant

CSS font variant property specifies how to set font variant of an element. It may be normal and small-caps.

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<style>`
5. `p { font-variant: small-caps; }`
5. `h3 { font-variant: normal; }`
7. `</style>`
3. `</head>`
9. `<body>`
10. `<h3>`This heading is shown in normal font.`</h3>`
11. `<p>`This paragraph is shown in small font.`</p>`
12. `</body>`
13. `</html>`



Test it Now

Output:

This heading is shown in normal font.

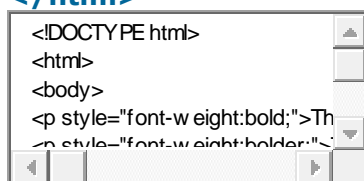
THIS PARAGRAPH IS SHOWN IN SMALL FONT.

6) CSS Font Weight

CSS font weight property defines the weight of the font and specify that how bold a font is. The possible values of font weight may be normal, bold, bolder, lighter or number (100, 200..... upto 900).

1. `<!DOCTYPE html>`

2. `<html>`
3. `<body>`
4. `<p style="font-weight:bold;">This font is bold.</p>`
5. `<p style="font-weight:bolder;">This font is bolder.</p>`
5. `<p style="font-weight:lighter;">This font is lighter.</p>`
7. `<p style="font-weight:100;">This font is 100 weight.</p>`
3. `<p style="font-weight:200;">This font is 200 weight.</p>`
9. `<p style="font-weight:300;">This font is 300 weight.</p>`
10. `<p style="font-weight:400;">This font is 400 weight.</p>`
11. `<p style="font-weight:500;">This font is 500 weight.</p>`
12. `<p style="font-weight:600;">This font is 600 weight.</p>`
13. `<p style="font-weight:700;">This font is 700 weight.</p>`
14. `<p style="font-weight:800;">This font is 800 weight.</p>`
15. `<p style="font-weight:900;">This font is 900 weight.</p>`
16. `</body>`
17. `</html>`



Test it Now

Output:

This font is bold.

This font is bolder.

This font is lighter.

This font is 100 weight.

This font is 200 weight.

This font is 300 weight.

This font is 400 weight.

This font is 500 weight.

This font is 600 weight.

This font is 700 weight.

This font is 800 weight.

This font is 900 weight.

Java script

JavaScript is *an object-based scripting language* that is lightweight and cross-platform.

JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

Where JavaScript is used

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time

- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)

Displaying clocks etc.

JavaScript Example

1. `<h2>Welcome to JavaScript</h2>`
2. `<script>`
3. `document.write("Hello JavaScript by JavaScript");`

`</script>`

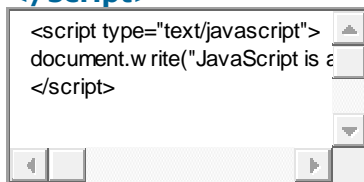
JavaScript Example

1. JavaScript Example
2. Within body tag
3. Within head tag

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let's create the first JavaScript example.

1. `<script type="text/javascript">`
2. `document.write("JavaScript is a simple language for javatpoint learners");`
3. `</script>`



Test it Now

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through

JavaScript. We will learn about document object in detail later.

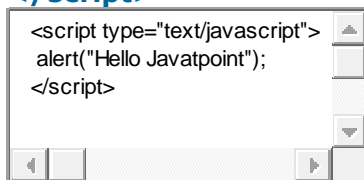
3 Places to put JavaScript code

1. Between the body tag of html
 2. Between the head tag of html
 3. In .js file (external javascript)
-

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

1. `<script type="text/javascript">`
2. `alert("Hello Javatpoint");`
3. `</script>`



[Test it Now](#)

2) JavaScript Example : code between the head tag

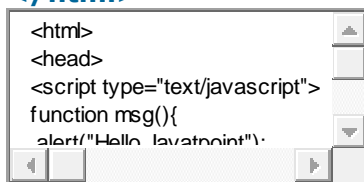
Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function `msg()`. To create function in JavaScript, you need to write function with `function_name` as given below.

To call function, you need to work on event. Here we are using `onclick` event to

call msg() function.

1. `<html>`
2. `<head>`
3. `<script type="text/javascript">`
4. `function msg(){`
5. `alert("Hello Javatpoint");`
6. `}`
7. `</script>`
8. `</head>`
9. `<body>`
10. `<p>Welcome to JavaScript</p>`
11. `<form>`
12. `<input type="button" value="click" onclick="msg()"/>`
13. `</form>`
14. `</body>`
15. `</html>`



External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

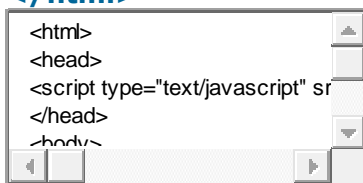
1. function msg(){
2. alert("Hello Javatpoint");
3. }



Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>



Document Object Model

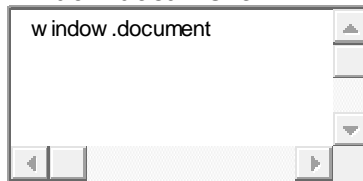
1. Document Object
2. Properties of document object
3. Methods of document object
4. Example of document object

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

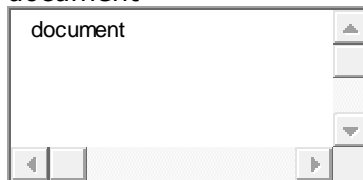
As mentioned earlier, it is the object of window. So

1. window.document



Is same as

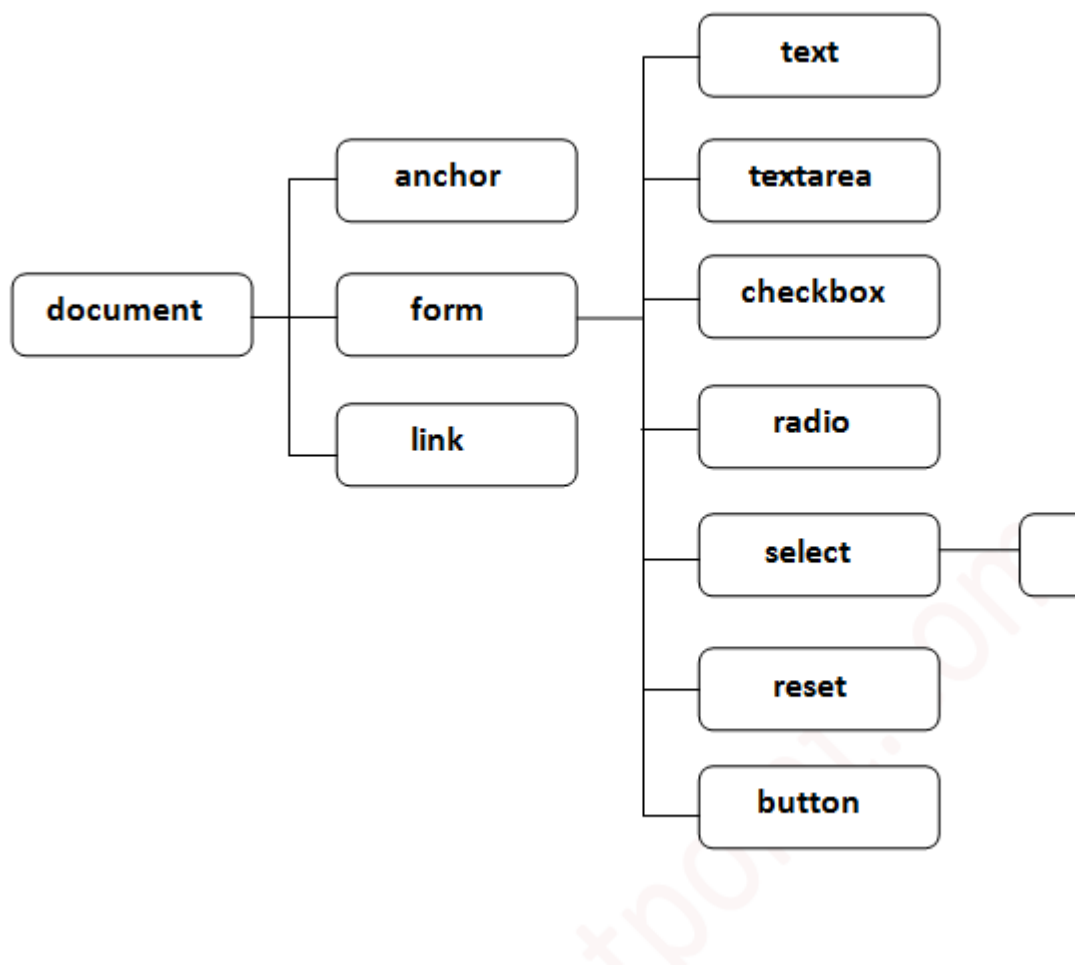
1. document



According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

Method	Description
<code>write("string")</code>	writes the given string on the document.
<code>writeln("string")</code>	writes the given string on the document with newline character at the end.

<code>getElementById()</code>	returns the element having the given id value.
<code>getElementsByName()</code>	returns all the elements having the given name value.
<code>getElementsByTagName()</code>	returns all the elements having the given tag name.
<code>getElementsByClassName()</code>	returns all the elements having the given class name.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **`document.form1.name.value`** to get the value of name field.

Here, **`document`** is the root element that represents the html document.

`form1` is the name of the form.

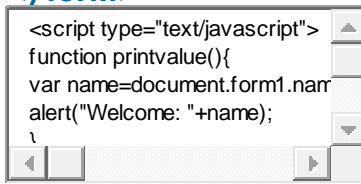
`name` is the attribute name of the input text.

`value` is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

1. `<script type="text/javascript">`
2. `function printvalue(){`
3. `var name=document.form1.name.value;`
4. `alert("Welcome: "+name);`
5. `}`
6. `</script>`
- 7.
8. `<form name="form1">`
9. Enter Name: `<input type="text" name="name"/>`

10. `<input type="button" onclick="printvalue()" value="print name"/>`
11. `</form>`



Output of the above example

Enter Name:

Javascript - document.getElementById() method

1. `getElementById()` method
2. Example of `getElementById()`

The **document.getElementById()** method returns the element of specified id.

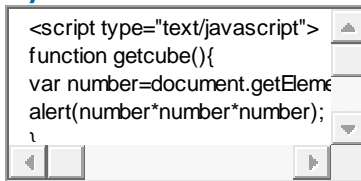
In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use `document.getElementById()` method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of `document.getElementById()` method that prints cube of the given number.

1. `<script type="text/javascript">`
2. `function getcube(){`
3. `var number=document.getElementById("number").value;`
4. `alert(number*number*number);`
5. `}`
6. `</script>`
7. `<form>`
8. Enter No: `<input type="text" id="number" name="number"/>
`

9. `<input type="button" value="cube" onclick="getcube()"/>`

10. `</form>`



```
<script type="text/javascript">
function getcube(){
var number=document.getElem
alert(number*number*number);
}
```

Output of the above example

Enter No:

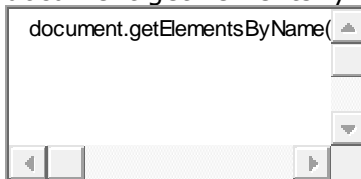
Javascript `document.getElementsByName()` method

1. `getElementsByName()` method
2. Example of `getElementsByName()`

The **`document.getElementsByName()`** method returns all the element of specified name.

The syntax of the `getElementsByName()` method is given below:

1. `document.getElementsByName("name")`



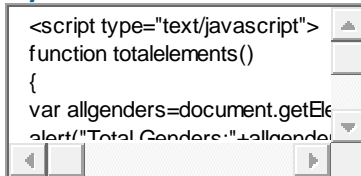
```
document.getElementsByName(
```

Here, name is required.

Example of `document.getElementsByName()` method

In this example, we going to count total number of genders. Here, we are using `getElementsByName()` method to get all the genders.

1. `<script type="text/javascript">`
2. `function totalelements()`
3. `{`
4. `var allgenders=document.getElementsByName("gender");`
5. `alert("Total Genders:"+allgenders.length);`
5. `}`
7. `</script>`
3. `<form>`
9. Male: `<input type="radio" name="gender" value="male">`
10. Female: `<input type="radio" name="gender" value="female">`
- 11.
12. `<input type="button" onclick="totalelements()" value="Total Genders">`
13. `</form>`



Output of the above example

Male: ☐ Female: ☐

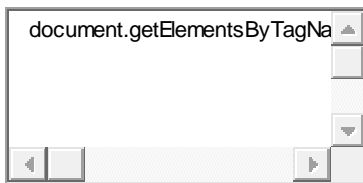
Bottom of FoJavascript - document.getElementsByTagName() method

1. `getElementsByTagName()` method
2. Example of `getElementsByTagName()`

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the `getElementsByTagName()` method is given below:

1. `document.getElementsByTagName("name")`

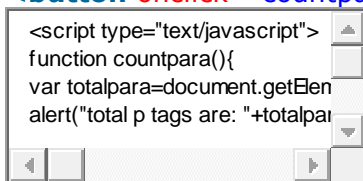


Here, name is required.

Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

1. `<script type="text/javascript">`
2. `function countpara(){`
3. `var totalpara=document.getElementsByTagName("p");`
4. `alert("total p tags are: "+totalpara.length);`
5. `}`
7. `</script>`
3. `<p>This is a pragraph</p>`
9. `<p>Here we are going to count total number of paragraphs by getElementByTagName() method.</p>`
10. `<p>Let's see the simple example</p>`
11. `<button onclick="countpara()">count paragraph</button>`



Output of the above example

This is a paragraph

Here we are going to count total number of paragraphs by getElementByTagName() method.

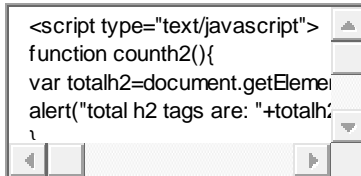
Let's see the simple example

count paragraph

Another example of document.getElementsByTagName() method

In this example, we going to count total number of h2 and h3 tags used in the document.

1. `<script type="text/javascript">`
2. `function counth2(){`
3. `var totalh2=document.getElementsByTagName("h2");`
4. `alert("total h2 tags are: "+totalh2.length);`
5. `}`
6. `function counth3(){`
7. `var totalh3=document.getElementsByTagName("h3");`
8. `alert("total h3 tags are: "+totalh3.length);`
9. `}`
10. `</script>`
11. `<h2>This is h2 tag</h2>`
12. `<h2>This is h2 tag</h2>`
13. `<h3>This is h3 tag</h3>`
14. `<h3>This is h3 tag</h3>`
15. `<h3>This is h3 tag</h3>`
16. `<button onclick="counth2()">count h2</button>`
17. `<button onclick="counth3()">count h3</button>`



Output of the above example

This is h2 tag

This is h2 tag

This is h3 tag

This is h3 tag

This is h3 tag

count h2 count h3

Note: Output of the given examples may differ on this page because it will count the total number of para , total number of h2 and total number of h3 tags used in this document

Javascript - innerHTML

1. javascript innerHTML
2. Example of innerHTML property

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

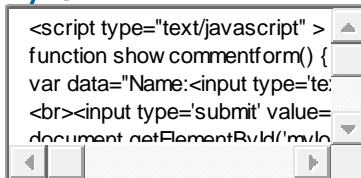
Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifying this position by calling the document.getElementById() method.

1. `<script type="text/javascript" >`
2. `function showcommentform() {`
3. `var data="Name:<input type='text' name='name'>
Comment:
<textarea rows='5' cols='80'></textarea>`
4. `
<input type='submit' value='Post Comment'>";`
5. `document.getElementById('mylocation').innerHTML=data;`
5. `}`

7. `</script>`
3. `<form name="myForm">`
9. `<input type="button" value="comment" onclick="showcommentform()">`
10. `<div id="mylocation"></div>`
11. `</form>`



[Test it Now](#)

Output of the above example

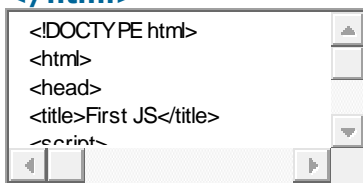
Show/Hide Comment Form Example using innerHTML

1. `<!DOCTYPE html>`
2. `<html>`
3. `<head>`
4. `<title>First JS</title>`
5. `<script>`
5. `var flag=true;`
7. `function commentform(){`
3. `var cform="<form action='Comment'>Enter Name:
<input type='text' name='name'/>
`
9. `Enter Email:
<input type='email' name='email'/>
Enter Comment:
`
10. `<textarea rows='5' cols='70'></textarea>
<input type='submit' value='Post Comment'/></form>";`
11. `if(flag){`
12. `document.getElementById("mylocation").innerHTML=cform;`
13. `flag=false;`
14. `}else{`

```

15. document.getElementById("mylocation").innerHTML="";
16. flag=true;
17. }
18. }
19. </script>
20. </head>
21. <body>
22. <button onclick="commentform()">Comment</button>
23. <div id="mylocation"></div>
24. </body>
25. </html>

```



Javascript - innerText

1. javascript innerText
2. Example of innerText property

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

```

1. <script type="text/javascript" >
2. function validate() {
3.   var msg;
4.   if(document.myForm.userPass.value.length>5){
5.     msg="good";
6.   }
7.   else{

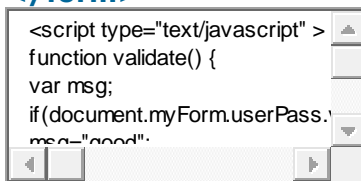
```



```

3. msg="poor";
9. }
10. document.getElementById('mylocation').innerText=msg;
11. }
12.
13. </script>
14. <form name="myForm">
15. <input type="password" value="" name="userPass" onkeyup="validate()">
16. Strength: <span id="mylocation">no strength</span>
17. </form>

```



Test it Now

Output of the above example

Strength: no strength

JavaScript Form Validation

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

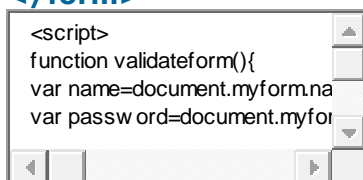
Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

JavaScript form validation example

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

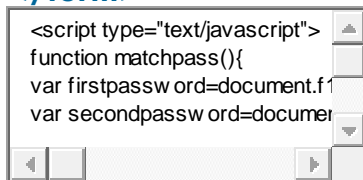
1. **<script>**
2. function validateform(){
3. var name=document.myform.name.value;
4. var password=document.myform.password.value;
- 5.
6. if (name==null || name==""){
7. alert("Name can't be blank");
8. return false;
9. }else if(password.length<6){
10. alert("Password must be at least 6 characters long.");
11. return false;
12. }
13. }
14. **</script>**
15. **<body>**
16. **<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >**
17. Name: **<input type="text" name="name">
**
18. Password: **<input type="password" name="password">
**
19. **<input type="submit" value="register">**
20. **</form>**



Test it Now

JavaScript Retype Password Validation

1. `<script type="text/javascript">`
2. `function matchpass(){`
3. `var firstpassword=document.f1.password.value;`
4. `var secondpassword=document.f1.password2.value;`
- 5.
6. `if(firstpassword==secondpassword){`
7. `return true;`
8. `}`
9. `else{`
10. `alert("password must be same!");`
11. `return false;`
12. `}`
13. `}`
14. `</script>`
- 15.
16. `<form name="f1" action="register.jsp" onsubmit="return matchpass()">`
17. Password: `<input type="password" name="password" />
`
18. Re-enter Password: `<input type="password" name="password2" />
`
19. `<input type="submit">`
20. `</form>`



[Test it Now](#)

JavaScript Number Validation

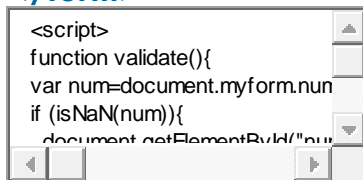
Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

1. `<script>`
2. `function validate(){`
3. `var num=document.myform.num.value;`

```

4.  if (isNaN(num)){
5.    document.getElementById("numloc").innerHTML="Enter Numeric value only"
      ;
6.    return false;
7.  }else{
8.    return true;
9.  }
10. }
11. </script>
12. <form name="myform" onsubmit="return validate()" >
13. Number: <input type="text" name="num"><span id="numloc"></span><
      br/>
14. <input type="submit" value="submit">
15. </form>

```



Test it Now

JavaScript validation with image

Let's see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```

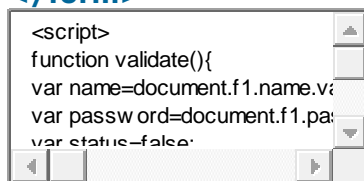
1. <script>
2. function validate(){
3.   var name=document.f1.name.value;
4.   var password=document.f1.password.value;
5.   var status=false;
6.
7.   if(name.length<1){
8.     document.getElementById("nameloc").innerHTML=
9.     " <img src='unchecked.gif'/> Please enter your name";
10.    status=false;
11.  }else{
12.    document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>

```

```

";
13. status=true;
14. }
15. if(password.length<6){
16. document.getElementById("passwordloc").innerHTML=
17. " <img src='unchecked.gif'/> Password must be at least 6 char long";
18. status=false;
19. }else{
20. document.getElementById("passwordloc").innerHTML=" <img src='checked.gi
    f'/>";
21. }
22. return status;
23. }
24. </script>
25.
26. <form name="f1" action="#" onsubmit="return validate()">
27. <table>
28. <tr><td>Enter Name:</td><td><input type="text" name="name"/>
29. <span id="nameloc"></span></td></tr>
30. <tr><td>Enter Password:</td><td><input type="password" name="pass
    word"/>
31. <span id="passwordloc"></span></td></tr>
32. <tr><td colspan="2"><input type="submit" value="register"/></td></tr>
    <
33. </table>
34. </form>

```



Test it Now

Output:

Enter Name:

Enter Password:

register

JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
1. <script>
2. function validateemail()
3. {
4.   var x=document.myform.email.value;
5.   var atposition=x.indexOf("@");
6.   var dotposition=x.lastIndexOf(".");
7.   if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
8.     alert("Please enter a valid e-
       mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
9.     return false;
10.  }
11. }
12. </script>
13. <body>
14. <form name="myform" method="post" action="#" onsubmit="return validateemail();">
15. Email: <input type="text" name="email"><br/>
16.
17. <input type="submit" value="register">
18. </form>
```

```
<script>
function validateemail()
{
var x=document.myform.email.v
var atposition=x.indexOf("@");
```

HTML/DOM events for JavaScript

HTML or DOM events are widely used in JavaScript code. JavaScript code is executed with HTML/DOM events. So before learning JavaScript, let's have some idea about events.

Events	Description
onclick	occurs when element is clicked.
ondblclick	occurs when element is double-clicked.
onfocus	occurs when an element gets focus such as button input, textarea etc.
onblur	occurs when form loses the focus from an element.
onsubmit	occurs when form is submitted.
onmouseover	occurs when mouse is moved over an element.
onmouseout	occurs when mouse is moved out from an element (after moved over).
onmousedown	occurs when mouse button is pressed over an element.
onmouseup	occurs when mouse is released from an element (after mouse is pressed).
onload	occurs when document, object or frameset is loaded.

onunload	occurs when body or frameset is unloaded.
onscroll	occurs when document is scrolled.
onresize	occurs when document is resized.
onreset	occurs when form is reset.
onkeydown	occurs when key is being pressed.
onkeypress	occurs when user presses the key.
onkeyup	occurs when key is released.

COLLECTIONS

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

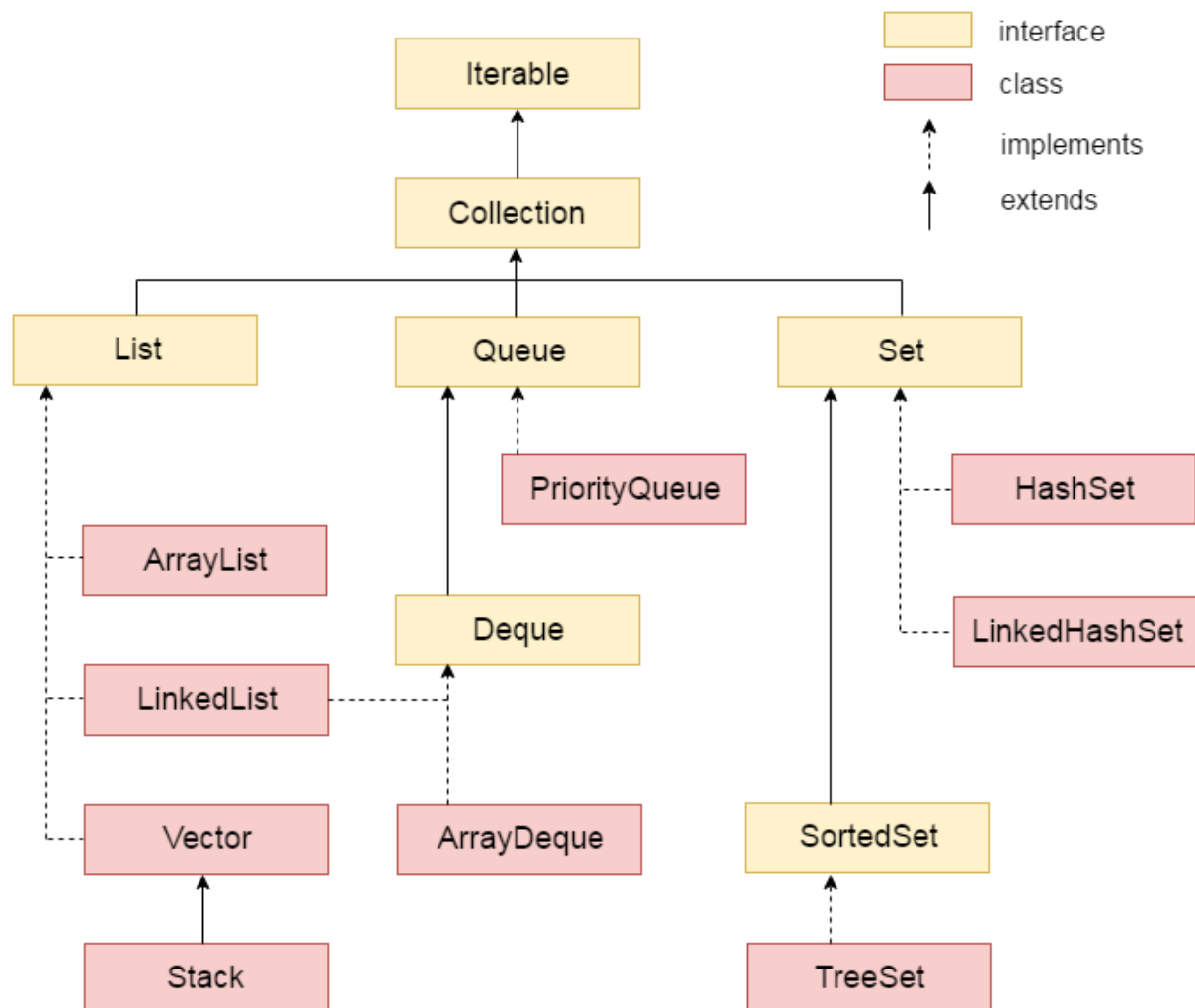
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm.

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework. The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection	is used to insert the specified collection elements in the

	c)	invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

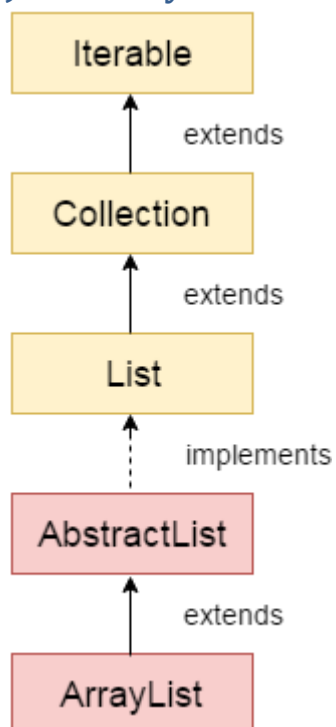
No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.

- | | | |
|---|-----------------------------------|---|
| 2 | <code>public Object next()</code> | It returns the element and moves the cursor pointer to the next element. |
| 3 | <code>public void remove()</code> | It removes the last elements returned by the iterator. It is rarely used. |
-

What we are going to learn in Java Collections Framework

1. [ArrayList class](#)
2. [LinkedList class](#)
3. [List interface](#)
4. [HashSet class](#)
5. [LinkedHashSet class](#)
6. [TreeSet class](#)
7. [PriorityQueue class](#)
8. [Map interface](#)
9. [HashMap class](#)
10. [LinkedHashMap class](#)
11. [TreeMap class](#)
12. [Hashtable class](#)
13. [Sorting](#)
14. [Comparable interface](#)
15. [Comparator interface](#)
16. [Properties class in Java](#)

Java ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

1. `public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable`

Constructors of Java ArrayList

<code>ArrayList()</code>	It is used to build an empty array list.
<code>ArrayList(Collection c)</code>	It is used to build an array list that is initialized with the elements of the collection c.
<code>ArrayList(int capacity)</code>	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

Method	Description
<code>void add(int index, Object element)</code>	It is used to insert the specified element at the specified position index in a list.
<code>boolean addAll(Collection c)</code>	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

<code>void clear()</code>	It is used to remove all of the elements from this list.
<code>int lastIndexOf(Object o)</code>	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
<code>Object[] toArray()</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code>Object[] toArray(Object[] a)</code>	It is used to return an array containing all of the elements in this list in the correct order.
<code>boolean add(Object o)</code>	It is used to append the specified element to the end of a list.
<code>boolean addAll(int index, Collection c)</code>	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
<code>Object clone()</code>	It is used to return a shallow copy of an ArrayList.
<code>int indexOf(Object o)</code>	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
<code>void trimToSize()</code>	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.

Let's see the old non-generic example of creating java collection.

1. `ArrayList al=new ArrayList();//creating old non-generic arraylist`

Let's see the new generic example of creating java collection.

1. `ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist`

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

For more information of java generics, click here [Java Generics Tutorial](#).

Java ArrayList Example

```
import java.util.*;

class TestCollection1{

    public static void main(String args[]){

        ArrayList<String> list=new ArrayList<String>();//Creating arraylist

        list.add("Ravi");//Adding object in arraylist

        list.add("Vijay");

        list.add("Ravi");

        list.add("Ajay");

        //Traversing list through Iterator

        Iterator itr=list.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

Output:-

```
Ravi
Vijay
Ravi
Ajay
```

Two ways to iterate the elements of collection in java

There are two ways to traverse collection elements:

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

Iterating Collection through for-each loop

```
import java.util.*;
```



```
class TestCollection2{

    public static void main(String args[]){

        ArrayList<String> al=new ArrayList<String>();

        al.add("Ravi");

        al.add("Vijay");

        al.add("Ravi");

        al.add("Ajay");

        for(String obj:al)

            System.out.println(obj);

    }

}
```

OUTPUT:-

```
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in array list.

```
class Student{

    int rollno;

    String name;

    int age;

    Student(int rollno,String name,int age){

        this.rollno=rollno;

        this.name=name;

        this.age=age;

    }

}
```

```

    }

import java.util.*;

public class TestCollection3{

    public static void main(String args[]){

        //Creating user-defined class objects

        Student s1=new Student(101,"Sonoo",23);

        Student s2=new Student(102,"Ravi",21);

        Student s2=new Student(103,"Hanumat",25);

        //creating arraylist

        ArrayList<Student> al=new ArrayList<Student>();

        al.add(s1);//adding Student class object

        al.add(s2);

        al.add(s3);
        //Getting Iterator
        Iterator itr=al.iterator();
        //traversing elements of ArrayList object
        while(itr.hasNext()){
            Student st=(Student)itr.next();
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
    }

}

```

Output:-

```

101 Sonoo 23
102 Ravi 21
103 Hanumat 25

```

Example of addAll(Collection c) method

```

import java.util.*;

class TestCollection4{

    public static void main(String args[]){

```

```

ArrayList<String> al=new ArrayList<String>();

al.add("Ravi");

al.add("Vijay");

al.add("Ajay");

ArrayList<String> al2=new ArrayList<String>();

al2.add("Sonoo");

al2.add("Hanumat");

al.addAll(al2);//adding second list in first list

Iterator itr=al.iterator();

while(itr.hasNext()){

    System.out.println(itr.next());

}

}

}

```

Output:-

```

Ravi
Vijay
Ajay
Sonoo
Hanumat

```

Example of removeAll() method

```

import java.util.*;

class TestCollection5{

    public static void main(String args[]){

        ArrayList<String> al=new ArrayList<String>();

        al.add("Ravi");

        al.add("Vijay");

        al.add("Ajay");
    }
}

```

```

ArrayList<String> al2=new ArrayList<String>();

al2.add("Ravi");

al2.add("Hanumat");

al.removeAll(al2);

System.out.println("iterating the elements after removing the elements of al2...");

Iterator itr=al.iterator();

while(itr.hasNext()){

    System.out.println(itr.next());

}

}

}

```

Output:-

```

    iterating the elements after removing the elements of al2...
    Vijay
    Ajay

```

Example of retainAll() method

```

import java.util.*;

class TestCollection6{

    public static void main(String args[]){

        ArrayList<String> al=new ArrayList<String>();

        al.add("Ravi");

        al.add("Vijay");

        al.add("Ajay");

        ArrayList<String> al2=new ArrayList<String>();

        al2.add("Ravi");
    }
}

```

```

al2.add("Hanumat");

al.retainAll(al2);

System.out.println("iterating the elements after retaining the elements of al2...");

Iterator itr=al.iterator();

while(itr.hasNext()){

    System.out.println(itr.next());

}

}

}

```

Output:-

```

    iterating the elements after retaining the elements of al2...
    Ravi

```

Java ArrayList Example: Book

Let's see an ArrayList example where we are adding books to list and printing all the books.

```

import java.util.*;

class Book {

    int id;

    String name,author,publisher;

    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {

        this.id = id;

        this.name = name;

        this.author = author;

        this.publisher = publisher;

        this.quantity = quantity;

    }
}

```

```

    }

    public class ArrayListExample {

    public static void main(String[] args) {

        //Creating list of Books

        List<Book> list=new ArrayList<Book>();

        //Creating Books

        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill"
,4);

        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

        //Adding Books to list

list.add(b1);

        list.add(b2);

        list.add(b3);

        //Traversing list

        for(Book b:list){

            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

        }

    }

}

```

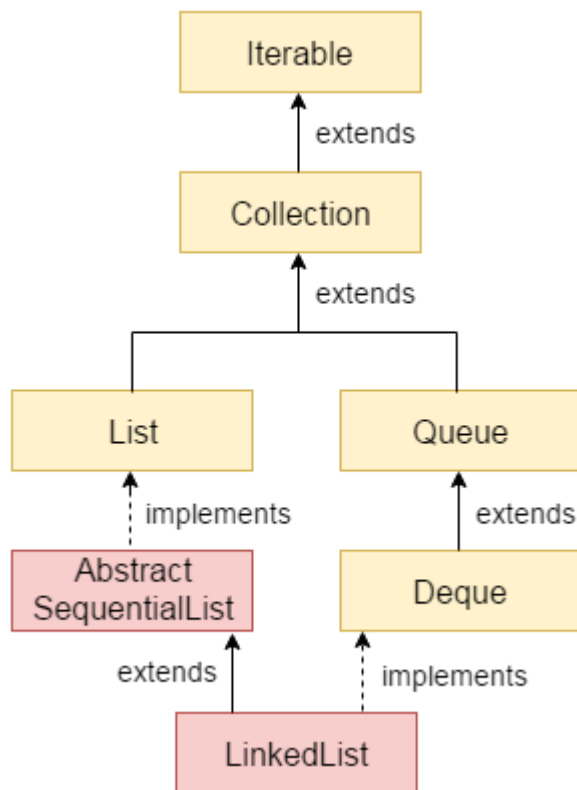
Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communication& Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6

```

Java LinkedList class



Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Hierarchy of LinkedList class

As shown in above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.



fig- doubly linked list

LinkedList class declaration

Let's see the declaration for java.util.LinkedList class.

1. `public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable`

Constructors of Java LinkedList

Constructor	Description
<code>LinkedList()</code>	It is used to construct an empty list.
<code>LinkedList(Collection c)</code>	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Methods of Java LinkedList

Method	Description
<code>void add(int index, Object element)</code>	It is used to insert the specified element at the specified position index in a list.
<code>void addFirst(Object o)</code>	It is used to insert the given element at the beginning of a list.
<code>void addLast(Object o)</code>	It is used to append the given element to the end of a list.
<code>int size()</code>	It is used to return the number of elements in a list
<code>boolean add(Object o)</code>	It is used to append the specified element to the end of a list.
<code>boolean contains(Object o)</code>	It is used to return true if the list contains a specified element.
<code>boolean remove(Object o)</code>	It is used to remove the first occurrence of the specified element in a list.
<code>Object getFirst()</code>	It is used to return the first element in a list.
<code>Object getLast()</code>	It is used to return the last element in a list.
<code>int indexOf(Object o)</code>	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
<code>int lastIndexOf(Object o)</code>	It is used to return the index in a list of the last occurrence of the

specified element, or -1 if the list does not contain any element.

Java LinkedList Example

```
import java.util.*;

public class TestCollection7{

    public static void main(String args[]){

        LinkedList<String> al=new LinkedList<String>();

        al.add("Ravi");

        al.add("Vijay");

        al.add("Ravi");

        al.add("Ajay");

        Iterator<String> itr=al.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

Java LinkedList Example: Book

```
import java.util.*;
```

```

class Book {

    int id;

    String name,author,publisher;

    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {

        this.id = id;

        this.name = name;

        this.author = author;

        this.publisher = publisher;

        this.quantity = quantity;

    }

}

public class LinkedListExample {

    public static void main(String[] args) {

        //Creating list of Books

        List<Book> list=new LinkedList<Book>();

        //Creating Books

        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc
Graw Hill",4);

        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

        //Adding Books to list

        list.add(b1);

        list.add(b2);

        list.add(b3);

        //Traversing list

```

```

        for(Book b:list){

            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

        }

    }

}

```

Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6

```

Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses dynamic array to store the elements.	LinkedList internally uses doubly linked list to store the elements.
2) Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is faster than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can act as a list only because it implements List only.	LinkedList class can act as a list and queue both because it implements List and Deque interfaces.
4) ArrayList is better for storing and accessing data.	LinkedList is better for manipulating data.

Example of ArrayList and LinkedList in Java

```

import java.util.*;
class TestArrayLinked{
    public static void main(String args[]){

        List<String> al=new ArrayList<String>();//creating arraylist
    }
}

```

```

al.add("Ravi");//adding object in arraylist
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");

List<String> al2=new LinkedList<String>();//creating linkedlist
al2.add("James");//adding object in linkedlist
al2.add("Serena");
al2.add("Swati");
al2.add("Junaid");

System.out.println("arraylist: "+al);
System.out.println("linkedlist: "+al2);
}
}

```

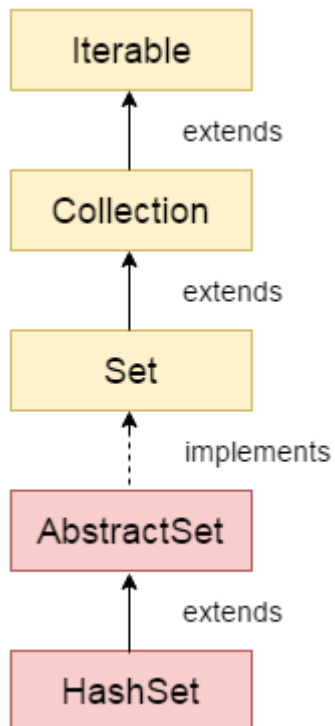
Output:

```

arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]

```

Java HashSet class



Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing**.
- HashSet contains unique elements only.

Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

Let's see the declaration for java.util.HashSet class.

1. `public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable`

Constructors of Java HashSet class:

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

Methods of Java HashSet class:

Method	Description
void clear()	It is used to remove all of the elements from this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean add(Object o)	It is used to adds the specified element to this set if it is not already present.

<code>boolean isEmpty()</code>	It is used to return true if this set contains no elements.
<code>boolean remove(Object o)</code>	It is used to remove the specified element from this set if it is present.
<code>Object clone()</code>	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
<code>Iterator iterator()</code>	It is used to return an iterator over the elements in this set.
<code>int size()</code>	It is used to return the number of elements in this set.

Java HashSet Example

```
import java.util.*;

class TestCollection9{

    public static void main(String args[]){

        //Creating HashSet and adding elements

        HashSet<String> set=new HashSet<String>();

        set.add("Ravi");

        set.add("Vijay");

        set.add("Ravi");

        set.add("Ajay");

        //Traversing elements

        Iterator<String> itr=set.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

Output:-

Ajay
Vijay
Ravi

Java HashSet Example: Book

Let's see a HashSet example where we are adding books to set and printing all the books.

```
import java.util.*;

class Book {

    int id;

    String name,author,publisher;

    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {

        this.id = id;

        this.name = name;

        this.author = author;

        this.publisher = publisher;

        this.quantity = quantity;

    }

}

public class HashSetExample {

    public static void main(String[] args) {

        HashSet<Book> set=new HashSet<Book>();

        //Creating Books

        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4)
        ;

        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

        //Adding Books to HashSet

        set.add(b1);
```



```

        set.add(b2);

        set.add(b3);

        //Traversing HashSet

        for(Book b:set){

            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

        }

    }

}

```

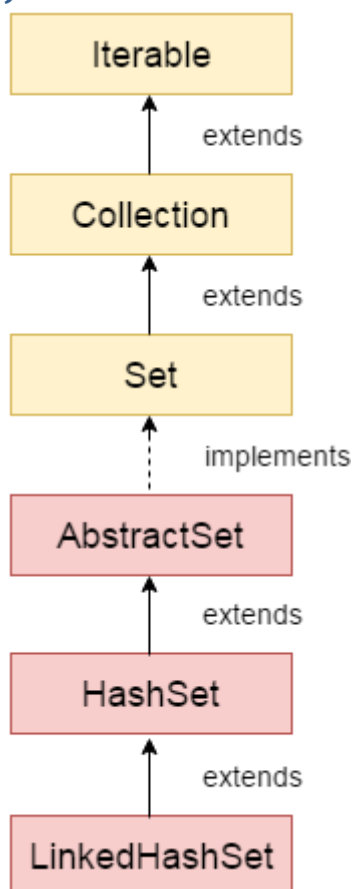
Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley

```

Java LinkedHashSet class



Java `LinkedHashSet` class is a Hash table and Linked list implementation of the set interface. It inherits `HashSet` class and implements `Set` interface.

The important points about Java `LinkedHashSet` class are:

- Contains unique elements only like `HashSet`.
- Provides all optional set operations, and permits null elements.
- Maintains insertion order.

Hierarchy of `LinkedHashSet` class

The `LinkedHashSet` class extends `HashSet` class which implements `Set` interface. The `Set` interface inherits `Collection` and `Iterable` interfaces in hierarchical order.

`LinkedHashSet` class declaration

Let's see the declaration for `java.util.LinkedHashSet` class.

1. `public class LinkedHashSet<E> extends HashSet<E> implements Set<E>, Cloneable, Serializable`

Constructors of Java `LinkedHashSet` class

Constructor	Description
<code>HashSet()</code>	It is used to construct a default <code>HashSet</code> .
<code>HashSet(Collection c)</code>	It is used to initialize the hash set by using the elements of the collection <code>c</code> .
<code>LinkedHashSet(int capacity)</code>	It is used initialize the capacity of the <code>linkedhashset</code> to the given integer value capacity.
<code>LinkedHashSet(int capacity, float fillRatio)</code>	It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument.

Example of `LinkedHashSet` class:

```
import java.util.*;

class TestCollection10{

    public static void main(String args[]){

        LinkedHashSet<String> al=new LinkedHashSet<String>();
```

```

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

    System.out.println(itr.next());

}

}

}

```

Output:-

```

Ravi
Vijay
Ajay

```

Java LinkedHashSet Example: Book

```

import java.util.*;

class Book {

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

    this.id = id;

    this.name = name;

    this.author = author;

    this.publisher = publisher;

```

```

        this.quantity = quantity;
    }
}

public class LinkedHashSetExample {

    public static void main(String[] args) {

        LinkedHashSet<Book> hs=new LinkedHashSet<Book>();

        //Creating Books

        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);

        //Adding Books to hash table

        hs.add(b1);

        hs.add(b2);

        hs.add(b3);

        //Traversing hash table

        for(Book b:hs){

            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

        }

    }

}

```

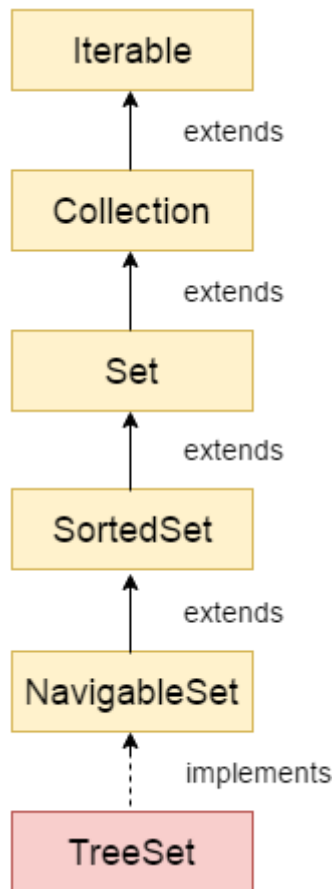
Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6

```

Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quite fast.
- Maintains ascending order.

Hierarchy of TreeSet class

As shown in above diagram, Java TreeSet class implements NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

1. `public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable`

Constructors of Java TreeSet class

Constructor	Description
<code>TreeSet()</code>	It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.
<code>TreeSet(Collection c)</code>	It is used to build a new tree set that contains the elements of the collection c.
<code>TreeSet(Comparator comp)</code>	It is used to construct an empty tree set that will be sorted according to given comparator.
<code>TreeSet(SortedSet ss)</code>	It is used to build a TreeSet that contains the elements of the given SortedSet.

Methods of Java TreeSet class

Method	Description
<code>boolean addAll(Collection c)</code>	It is used to add all of the elements in the specified collection to this set.
<code>boolean contains(Object o)</code>	It is used to return true if this set contains the specified element.
<code>boolean isEmpty()</code>	It is used to return true if this set contains no elements.
<code>boolean remove(Object o)</code>	It is used to remove the specified element from this set if it is present.
<code>void add(Object o)</code>	It is used to add the specified element to this set if it is not already present.
<code>void clear()</code>	It is used to remove all of the elements from this set.
<code>Object clone()</code>	It is used to return a shallow copy of this TreeSet instance.
<code>Object first()</code>	It is used to return the first (lowest) element currently in this sorted set.
<code>Object last()</code>	It is used to return the last (highest) element currently in this sorted

set.

int size()

It is used to return the number of elements in this set.

Java TreeSet Example

```
import java.util.*;

class TestCollection11{

    public static void main(String args[]){

        //Creating and adding elements

        TreeSet<String> al=new TreeSet<String>();

        al.add("Ravi");

        al.add("Vijay");

        al.add("Ravi");

        al.add("Ajay");

        //Traversing elements

        Iterator<String> itr=al.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

[Test it Now](#)

Output:

```
Ajay
Ravi
Vijay
```

Java TreeSet Example: Book

Let's see a TreeSet example where we are adding books to set and printing all the books. The elements in TreeSet must be of Comparable type. String and Wrapper classes are Comparable

by default. To add user-defined objects in TreeSet, you need to implement Comparable interface.

```
import java.util.*;

class Book implements Comparable<Book>{

    int id;

    String name,author,publisher;

    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {

        this.id = id;

        this.name = name;

        this.author = author;

        this.publisher = publisher;

        this.quantity = quantity;

    }

    public int compareTo(Book b) {

        if(id>b.id){

            return 1;

        }else if(id<b.id){

            return -1;

        }else{

            return 0;

        }

    }

}

public class TreeSetExample {

    public static void main(String[] args) {
```



```

Set<Book> set=new TreeSet<Book>();

//Creating Books

Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

Book b2=new Book(233,"Operating System","Galvin","Wiley",6);

Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill"
,4);

//Adding Books to TreeSet

set.add(b1);

set.add(b2);

set.add(b3);

//Traversing TreeSet

for(Book b:set){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

}

}

}

```

Output:

```

101 Data Communications & Networking Forouzan Mc Graw Hill 4
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6

```

Java Queue Interface

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

Queue Interface declaration

1. `public interface Queue<E> extends Collection<E>`

Methods of Java Queue Interface

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

PriorityQueue class

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner. It inherits AbstractQueue class.

PriorityQueue class declaration

Let's see the declaration for java.util.PriorityQueue class.

1. `public class PriorityQueue<E> extends AbstractQueue<E> implements Serializable`

Java PriorityQueue Example

```
import java.util.*;

class TestCollection12{

    public static void main(String args[]){

        PriorityQueue<String> queue=new PriorityQueue<String>();

        queue.add("Amit");

        queue.add("Vijay");

        queue.add("Karan");

        queue.add("Jai");
```

```

queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}

```

Output:-

```

head:Amit
head:Amit
iterating the queue elements:
Amit
Jai
Karan
Vijay
Rahul
after removing two elements:
Karan
Rahul
Vijay

```

Java PriorityQueue Example: Book

Let's see a PriorityQueue example where we are adding books to queue and printing all the

books. The elements in PriorityQueue must be of Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in PriorityQueue, you need to implement Comparable interface.

```
import java.util.*;

class Book implements Comparable<Book>{

    int id;

    String name,author,publisher;

    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {

        this.id = id;

        this.name = name;

        this.author = author;

        this.publisher = publisher;

        this.quantity = quantity;

    }

    public int compareTo(Book b) {

        if(id>b.id){

            return 1;

        }else if(id<b.id){

            return -1;

        }else{

            return 0;

        }

    }

}

public class LinkedListExample {
```

```

public static void main(String[] args) {

    Queue<Book> queue=new PriorityQueue<Book>();

    //Creating Books

    Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

    Book b2=new Book(233,"Operating System","Galvin","Wiley",6);

    Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill"
,4);

    //Adding Books to the queue

    queue.add(b1);

    queue.add(b2);

    queue.add(b3);

    System.out.println("Traversing the queue elements:");

    //Traversing queue elements

    for(Book b:queue){

        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

    queue.remove();

    System.out.println("After removing one book record:");

    for(Book b:queue){

        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

    }

}

}

```

Output:

```

Traversing the queue elements:
101 Data Communications & Networking Forouzan Mc Graw Hill 4
233 Operating System Galvin Wiley 6
121 Let us C Yashwant Kanetkar BPB 8

```

After removing one book record:
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6

Java Deque Interface

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for "**double ended queue**".

Deque Interface declaration

1. `public interface Deque<E> extends Queue<E>`

Methods of Java Deque Interface

Method	Description
<code>boolean add(object)</code>	It is used to insert the specified element into this deque and return true upon success.
<code>boolean offer(object)</code>	It is used to insert the specified element into this deque.
<code>Object remove()</code>	It is used to retrieves and removes the head of this deque.
<code>Object poll()</code>	It is used to retrieves and removes the head of this deque, or returns null if this deque is empty.
<code>Object element()</code>	It is used to retrieves, but does not remove, the head of this deque.
<code>Object peek()</code>	It is used to retrieves, but does not remove, the head of this deque, or returns null if this deque is empty.

ArrayDeque class

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

The important points about ArrayDeque class are:

- Unlike Queue, we can add or remove elements from both sides.
- Null elements are not allowed in the ArrayDeque.
- ArrayDeque is not thread safe, in the absence of external synchronization.
- ArrayDeque has no capacity restrictions.

- ArrayDeque is faster than LinkedList and Stack.

ArrayDeque Hierarchy

The hierarchy of ArrayDeque class is given in the figure displayed at the right side of the page.

ArrayDeque class declaration

Let's see the declaration for java.util.ArrayDeque class.

1. `public class ArrayDeque<E> extends AbstractCollection<E> implements Deque<E>, Cloneable, Serializable`

Java ArrayDeque Example

```
import java.util.*;

public class ArrayDequeExample {

    public static void main(String[] args) {

        //Creating Deque and adding elements

        Deque<String> deque = new ArrayDeque<String>();

        deque.add("Ravi");

        deque.add("Vijay");

        deque.add("Ajay");

        //Traversing elements

        for (String str : deque) {

            System.out.println(str);

        }

    }

}
```

Output:

```
Ravi
Vijay
```

Ajay

Java ArrayDeque Example: offerFirst() and pollLast()

```
import java.util.*;

public class DequeExample {

    public static void main(String[] args) {

        Deque<String> deque=new ArrayDeque<String>();

        deque.offer("arvind");

        deque.offer("vimal");

        deque.add("mukul");

        deque.offerFirst("jai");

        System.out.println("After offerFirst Traversal...");

        for(String s:deque){

            System.out.println(s);

        }

        //deque.poll();

        //deque.pollFirst();//it is same as poll()

        deque.pollLast();

        System.out.println("After pollLast() Traversal...");

        for(String s:deque){

            System.out.println(s);

        }

    }

}
```

Output:

```
After offerFirst Traversal...
jai
arvind
```



```
vimal  
mukul  
After pollLast() Traversal...  
jai  
arvind  
vimal
```

Java ArrayDeque Example: Book

```
import java.util.*;  
  
class Book {  
  
    int id;  
  
    String name,author,publisher;  
  
    int quantity;  
  
    public Book(int id, String name, String author, String publisher, int quantity) {  
  
        this.id = id;  
  
        this.name = name;  
  
        this.author = author;  
  
        this.publisher = publisher;  
  
        this.quantity = quantity;  
  
    }  
  
}  
  
public class ArrayDequeExample {  
  
    public static void main(String[] args) {  
  
        Deque<Book> set=new ArrayDeque<Book>();  
  
        //Creating Books  
  
        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);  
  
        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill"  
,4);  
  
        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
```

```

//Adding Books to Deque

set.add(b1);

set.add(b2);

set.add(b3);

//Traversing ArrayDeque

for(Book b:set){

    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

}

}

}

```

Output:

```

101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6

```

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

ArrayList	Vector
1) ArrayList is not synchronized .	Vector is synchronized .
2) ArrayList increments 50% of current array size if number of element exceeds from its capacity.	Vector increments 100% means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is not a legacy class, it is introduced in JDK 1.2.	Vector is a legacy class.
4) ArrayList is fast because it is non-synchronized.	Vector is slow because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.

5) ArrayList uses **Iterator** interface to traverse the elements. But it can use Iterator also.

Example of Java ArrayList

```
import java.util.*;

class TestArrayList21{

    public static void main(String args[]){

        List<String> al=new ArrayList<String>();//creating arraylist

        al.add("Sonoo");//adding object in arraylist

        al.add("Michael");

        al.add("James");

        al.add("Andy");

        //traversing elements using Iterator

        Iterator itr=al.iterator();

        while(itr.hasNext()){

            System.out.println(itr.next());

        }

    }

}
```

Output:

```
Sonoo
Michael
James
Andy
```

Example of Java Vector

Let's see a simple example of java Vector class that uses Enumeration interface.

```
import java.util.*;

class TestVector1{

    public static void main(String args[]){

        Vector<String> v=new Vector<String>();//creating vector

        v.add("umesh");//method of Collection

        v.addElement("irfan");//method of Vector

        v.addElement("kumar");

        //traversing elements using Enumeration

        Enumeration e=v.elements();

        while(e.hasMoreElements()){

            System.out.println(e.nextElement());

        }

    }

}
```

Output:

```
umesh
irfan
kumar
```


UNIT-II [XML]

3.1 XML and its goals :

XML means extensible Markup Language

→ Markup is a set of instructions, often called tags, which can be added to text files, when a file is processed by a suitable application, the tags are used to control the structure or presentation of data contained in the file.

①

Example of markups

1. Microsoft RTF (Rich Text Format)
2. ADOBE PDF (Portable document format)
3. HTML (HyperText Markup Language)

- These describe how data looks but give no information about what it is
- XML is used to describe the structure of a document not the way that is presented.
- XML is a recommendation of the world wide web consortium^(W3C)
- XML was developed by IBM in 1998. XML is derived from Standard Generalized Markup Language (SGML), which is a system for defining the markup language.
- XML is a case-sensitive meta-markup language because it allows you to create custom-defined markup languages
- You can use XML to design domain-specific language that describe the information in a document
- You can create XML documents using text editor and saved with ".xml" extension.

- XML is used to add structural and formatting information to data.

Areas of XML :-

1. XML structuring the data for storage where a ~~relational set~~ relational database is inappropriate.
2. Structuring the data for presentation on webpages

Advantages of XML :-

1. It acts as a mini database
2. XML is used to create new markup languages
3. Using XML, You can create your own tags
4. XML is used to create structured data.
5. XML is used to transfer data across applications on web
6. XML is language is independent and platform independent

Features of XML :-

1. Self-describing data: Each element and tag in an XML document is self-explanatory. You can easily identify the use of webpage and data contained in the webpage ~~by~~ by looking at code.
2. User-friendly:- You can create your own tags to create a webpage and need not remember any pre-defined tags.
3. Viewing data from a standard Tool: You can invoke

an XML document in the Internet Explorer, Firefox and all other web browsers used to view data.

4. Uses DTD :- XML uses DTD for defining the elements of XML as well as other languages such as HTML. ②

5. Content independent :- XML is content independent and does not require you to learn any special programming code to use this language. XML enables you to create your own context-free tags.

6. Platform independent :- XML is platform independent language that can be viewed/run almost all platforms.

7. Metadata :- XML enables you to provide details about information structure to websites.


GOALS of XML :-

1. XML should support many number of applications
2. XML should be directly usable over the internet
3. XML should be easier to write programs
4. XML should be used with SGML without any conflict
5. XML documents should be easily and clearly understandable by humans
6. The design of XML documents should be faster to generate
7. The XML document should be clearly and correctly designed

8. The XML document should be simple and easy to create
9. The optional features in XML is kept at absolute minimum (i.e ideally zero)
10. Minimum importance should be given to XML document compactness.

3.2 : XML components / Building Blocks of XML :-

Any XML document is just composed of 6 things.

1. Tags
2. Elements
3. Attributes
4. PCDATA
5. CDATA
6. Entities
7. Control information 

1. Tags :- Tags are used to specify a text string that represents an element in an XML document

ex: `<author> Winston </author>`

Empty tags: In XML, elements can be empty when they do not contain any parsed character data. In the case of an empty elements, the empty tags are must use a forward slash (/) after the name of the element

ex: `<fax />`

Note: All the tags must be closed by an exact equivalent

because XML is case-sensitive language.

- `<author>` tag should be closed by `</author>` tag
not by `</Author>` ③

Nesting of Tags: Even simple XML document has nesting of tags. They must be nested properly and closed in the reverse order in which they were opened.

2. Attributes: - XML attributes store additional information about the data. i.e. metadata. (data about data)

- You can use attributes to store IDs, URLs, references and other information not directly relevant to the end user.
- Attributes simply describes the properties of elements.

3. Elements: - XML elements are used to store data. To create an XML element, you need to follow these naming conventions

* An element must begin with

- Letters
- Numbers
- Underscore

* An element can have special characters such as hyphens (-), underscore (_), or dot (.).

* There can not be any blank space in tag names.

- Elements can be represented with tags and empty tags

`<author> XYZ </author>`
└──────────────────┘
Simple element

4. PCDATA :- PCDATA means Parsed Character Data. It is the data which will be suitably parsed by a given parser. In general, PCDATA may be a simple XML file or markup.

5. CDATA :- CDATA means Character Data. This is just opposite to PCDATA. i.e. there is no entity ~~markup~~ expansion. The CDATA is not parsed at all.

6. Entities :- An Entity is a thing which is used as a part of the document but which is not a simple element.

ex: An image or encrypted digital signature.

7. Control Information :- There are three types of control information

7.1 Comments : Comments are used to describe a document in order to have a proper documentation of the coding. These are ignored by the XML parsers and applications using the XML document. In XML comments are begin with

<!-- and end with -->

ex: <!-- This is a comment -->

- Same type of comment is used for both XML source files and in DTD files.

7.2 Processing Instructions : They used to control applications.

ex: <?xml version="1.0" ?> → XML Prolog

This instruction must be the first statement in your XML document. This instruction tells the applications that data in the file follows the rules of XML version "1.0" (4)

7.3 Document Type Definition (DTD) :- Each valid XML document has associated with DTD. This DTD usually held in a separate file, so it can be used with many documents. The DTD file holds the rules of grammar for a particular XML data structure. Those rules are used by the validating parsers to check that is not only the valid XML file, but it also obeys its own internal rules.

ex: `<!DOCTYPE Course SYSTEM "courses.dtd">`

This declaration tells the parser that the XML file is of the courses and it uses a DTD which stored in a file called "Courses.dtd"

There are two types of DTDs

1. PUBLIC
2. SYSTEM

- PUBLIC DTDs are predefined DTDs. They are confirmed to the international standards such as those W3C used in HTML.
- SYSTEM DTDs: Any DTD which is developed by you or for you.

CREATING XML DOCUMENT :- XML is based on containment model in which each XML element contains

text or other XML elements, called child elements, within it. When you create an XML code for a webpage, you need to ensure that the element do not contain both data and child elements.

- When you create an XML document, you need to ensure that the document is well-formed
- A well-formed document must have
 1. A root element
 2. An opening tag and closing tag
 3. An appropriate nesting
 4. Values always enclosed in the quotation marks.

Syntax

```
<?xml version="1.0" encoding="UTF-8"?>  
  <root>  
    <child>  
      <Subchild> text </Subchild>  
    </child>  
  </root>
```

For example: To create a structure for book, you need a book title, author name and book contents. contents are further divided into sections.

- webpage about book can be structured as follows

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

```

<BookTitle> XML Bible </BookTitle>
<Author> Sam Peter </Author>
<Contents>
    <chapter1> XML Basics </chapter1>
    <chapter2> XML DTDs </chapter2>
    <chapter3> XML Schemas </chapter3>
    <chapter4> XML Namespaces </chapter4>
    <chapter5> XSL Transformations </chapter5>
</Contents>
</Book>

```

⑤

- In the above code, the first instruction is called Processing instruction or XML Prolog, which tells the applications how to handle XML. It also serves as version declaration and says that file is XML which confirms the rules of XML version 1.0.
- The rule states that the parser must halt when it finds an error and that it may return a message back to the calling application.

3.3 Valid or Well-formed:- A well-formed and valid XML document is one which follows all of the below mentioned rules of XML i.e

1. Tags are matched and don't overlap.
2. Empty elements must be ended properly.
3. The document contains XML declaration

4. A valid XML document has its own DTD.

3.4 DOCUMENT TYPE DEFINITION (DTD) :-

The XML has neither meaning nor context without a grammar against which it can be validated. The grammar is called DTD. DTDs are used to give structure to an XML document and valid XML document must have a DTD attached to it. DTD files can be created in text editor and are saved with a "dtd" extension.

— A DTD can be embedded internally or externally (i.e. an external file attached to an XML document)

— Syntax for internal DTD :

`<!DOCTYPE ROOTELEMENT [ELEMENT Declarations]>`

— Syntax for external DTD :

`<!DOCTYPE ROOTELEMENT SYSTEM/PUBLIC "DTD filename">`

— In the above syntax

- * The `<!DOCTYPE` indicates to the parser that, this is a DTD file
- * `ROOTELEMENT` indicates beginning of XML document.
- * Element declarations specifies information about all of the parent and child elements of the XML document.
- * `SYSTEM/PUBLIC` indicates location of the DTD file to the web browser
- * DTD filename : Specifies name of the ^{DTD} file.

Declaring elements in DTD:- The XML document is composed of number of elements in which each of the elements may itself be made from other elements and some of the elements may contain attributes. This structure must be reflected in DTD (X)

- The first node of the XML document is called "root node". Root node contains all other elements of document and each XML document must have exactly one root node.
- Each element can either be a container which holds further elements or it can define data.

Syntax

(6)

`<!ELEMENT elementname type >`

- In the above syntax, *) element name is the name of the element/tag used to markup the element in XML doc.

*) The "type" specifies the type of the element, also known as the Content Specification model of the element

- An XML supports four types of elements. They are

1. Empty
2. Element only
3. Mixed
4. Any.

1. Empty: The Empty element does not contain any data. Empty elements provide information through attributes. Empty elements are declared as

<!ELEMENT elementname EMPTY>

2. Element only :- This element contains only child elements and no character data. You can declare an element as "Element only" by specifying the content model for the element.

— The content model is a list of child elements and element declaration symbols.

— The element declaration symbols used to specify the content model are

1. Parenthesis () : Enclose a sequence of child nodes / elements

2. Comma (,) : Separates items in the order in which they must appear.

3. Pipe (|) : Separates the list of items where only one item can appear at a time

4. No Symbol : It must appear exactly once

5. Question mark (?) : Items must appear exactly once or not at all.

6. Star (*) : The item can appear any number of times.

7. plus (+) : The item must appear at least once.

ex: the content model for Course

<!ELEMENT course (name, (lmsum | installment),
description?, module *) >

3. Mixed elements :- This element contains combination of child elements and character data. The simplest mixed element is one that contains only character data.

Syntax :

`<ELEMENT elementname (#PCDATA) >`

PCDATA indicates the element contains parsed character data.

4. ANY Element :- The ANY element has virtually no structure and can contain character data.

Syntax :

`<ELEMENT elementname ANY>` (7)

DECLARING ATTRIBUTES IN DTD :-

Attributes specify additional information about element and are used inside the tag in the ^{form of} name-value pairs

Syntax :

`<!ATTLIST elementname attributename attribute
type default >`

In the above syntax,

- element name is the name of the element for which the attribute is defined
- attribute name is the name of the attribute
- attribute type defines the type of attribute
- Default specifies the default value of the attribute or a symbol that indicates the use of attribute.
- The values that default can take
 1. #REQUIRED: indicates that the attribute is required.
 2. #IMPLIED: indicates that the attribute is optional.

3. #FIXED: indicates that the attribute has a fixed value
4. Default: default value of the attribute
5. CDATA: denotes unparsed character data / plain text data
6. Enumerated: denotes a series of string values
7. NOTATION: denotes a notation declared somewhere else in DTD
8. ENTITY: denotes an external binary entity.
9. ENTITIES: denotes multiple ^{external} binary entities separated by whitespace.
10. ID: denotes a unique identifier
11. NMTOKEN: denotes a name consisting of XML token characters such as letters, numbers, periods, dashes, colons and underscores.
12. NMTOKENS: denotes a series / multiple names consisting of XML token characters.

ex:

```
<!ATTLIST qty amount CDATA #REQUIRED
          unit CDATA "gms" >
```

ENTITIES:- Entities are used to represent complex data. Entity is container which will be filled with some form of content. this content may be include in the XML file

- ① as an internal entity
- ② as an external entity (stored in another file)

Internal Entities: are used to create a small pieces of data which you want to use repeatedly throughout your

Schema.

Syntax : `<!ENTITY entityname "replacement text">`

`<!ENTITY pos "pinch of salt" >`

<code>&amp;</code>	<code>- &</code>
<code>&lt;</code>	<code>- <</code>
<code>&gt;</code>	<code>- ></code>
<code>&pos</code>	<code>- ('')</code>
<code>&quot;</code>	<code>- (")</code>

Usage:

`<item>` finally add a `&pos;` `</item>`

When an entity is included, the name is preceded by an ampersand (&) and followed by a semicolon (;)

External Entities :- Almost anything which is data can be included in your XML as an external entity

for example; to add an image to your XML file ⑧

`<!ENTITY myimage SYSTEM "xyz.jpg" NDATA JPG>`

When an XML parser does not understand your entity, a helper application must be specified, the data will then be passed to this helper for processing.

`<!NOTATION JPG SYSTEM "mspaint" >`

that passes the images to a paint program for viewing.

* Consider the following code that illustrates INTERNAL DTD for Courses.

courses.xml:

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

`<!DOCTYPE courses [`

`<!ELEMENT courses (course)* >`

`<!ELEMENT course (name, (lumpsum?, installment?),
description?, modules*) >`

`<!ELEMENT name (#PCDATA) >`

```

<ELEMENT lumpsum (#PCDATA)>
<ELEMENT installment (#PCDATA)>
<ELEMENT description (#PCDATA)>
<ELEMENT modules (#PCDATA)> ]>

```

```

<courses>

```

```

  <course>

```

```

    <name> M.Tech </name>

```

```

    <lumpsum> 42500 </lumpsum>

```

```

    <modules> computer science </modules>

```

```

    <modules> web programming </modules>

```

```

  </course>

```

```

  <course>

```

```

    <name> B.Tech </name>

```

```

    <installment> 20000 </installment>

```

```

    <installment> 13000 </installment>

```

```

    <description> undergraduate course </description>

```

```

    <modules> computer basics </modules>

```

```

    <modules> software engg. </modules>

```

```

  </course>

```

```

</courses>

```

— . —

~~XML SCHEMAS~~ Limitations of DTD :-

1. DTD itself is not in XML format - more work for parsers
2. Does not express datatypes (weak data typing)
3. No namespace support
4. Document can override external DTD definitions

5. No Document Object model support (DOM)

3.5 XML Schemas :-

- The purpose of XML Schemas are to define a legal building blocks of XML document just like a DTD.
- XML Schemas are successors of DTDs
- XML Schemas are XML documents.
- XML Schemas will be in most web applications as a replacement for DTDs. (9)
- An XML Schema defines
 - An element that appears in a document
 - attributes that appears in a document
 - which elements are child elements
 - Order of child elements
 - no. of child elements
 - whether an element is empty or can include text
 - Data types for elements and attributes
 - Default and fixed values for elements and attributes

Advantages / Reasons for use Schemas :

XML Schemas are

- extensible to future additions
- richer and more powerful than DTDs
- written in XML
- support data types
- support namespaces.

Syntax:

```
<xsd:element name='ele-name' type='dataType'
minOccurs='+ve integer' maxOccurs='+ve integer|
unbounded' />
```

Where name is the name of the element being declared.

-type is the dataType of the element being declared.

-minOccurs specifies the minimum number of times an element can appear. (10)

-maxOccurs specifies the maximum number of times an element can appear; unbounded means an element can appear any number of times.

ex:

```
<xsd:element name='pname' type='xsd:string' />
```

Defining ComplexType Element :- The ComplexType is one that contains other elements, attributes and mixed content. To declare a ComplexType,

Syntax:

```
<xsd:complexType>
```

skeleton of the complexType.

```
</xsd:complexType>
```

ex:

```
<xsd:element name="address">
```

```
<xsd:complexType>
```

```
<xsd:sequence>
```

```
<xsd:element name="pno" type="xsd:string" />
```

```
<xsd:element name="street" type="xsd:string" />
```

```
<xsd:element name="city" type="xsd:string" />
```

```
</xsd:sequence>
```

</xsd:complexType>
</xsd:element>

A Simple XML Schema Program for

Defining attributes : - Attributes carry additional information regarding elements.

Syntax :

<xsd:attribute name = 'attribute name'
type = 'datatype' />

* A Simple XML Schema that describes product information.
Product.xsd :

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

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

<xsd:element name = "product" >

<xsd:complexType >

<xsd:sequence >

<xsd:element name = "pname" type = "xs:string" />

<xsd:element name = "desc" type = "xs:string" />

<xsd:element name = "price" type = "xs:^{float}integer" />

<xsd:element name = "qty" type = "xs:integer" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:schema>

3.5 XSL (Presenting XML) :-

- XSL stands for Extensible Stylesheet Language
- CSS was designed for styling HTML pages and can also be used to style XML pages.
- XSL was specifically designed to style XML pages and is much more sophisticated than CSS.
- XSL can convert an XML file into another XML with different. The most common type of XSL processing is to convert XML file in HTML file which can be displayed by web browser. (11)
- XSL is the bridge between XML and HTML.
➔
- XSL consists of 3 languages. They are:

1. XSLT (XSL Transformations): An XML based language that allows you to transform an XML document into another XML document. (possibly HTML documents).

2. XPath (XML Path): - XPath is a language to select parts of XML to transform with XSLT such as elements and attributes

3. XSL-fo (Formatting Objects): It is replacement for CSS. objects that specify how data is to be displayed.

How does it work :-

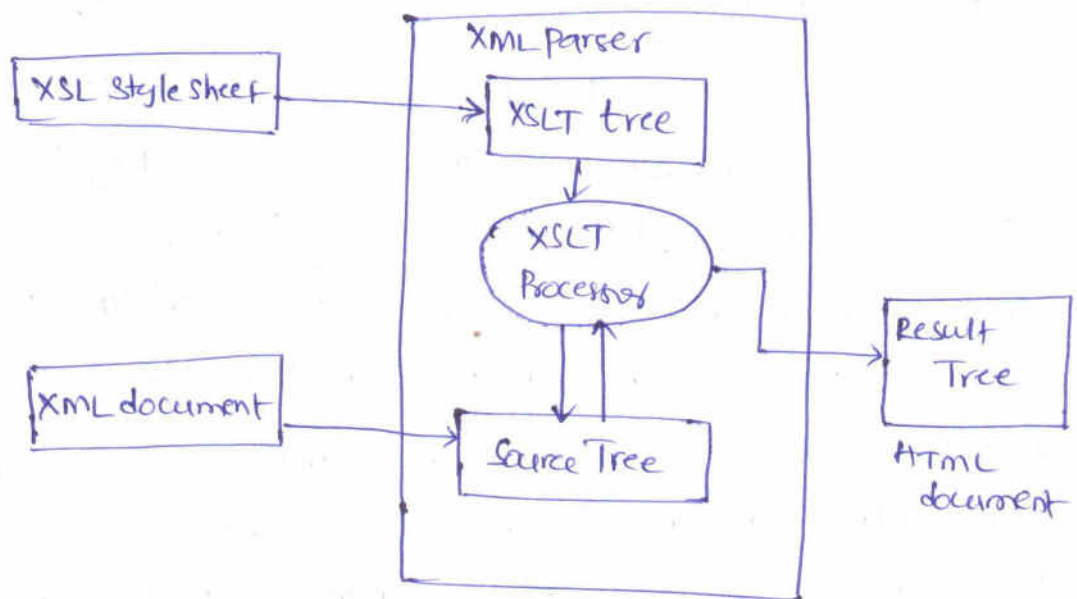


fig: working of XSLT Processor

1. The XML document is parsed into XML Source Tree
 2. the XML parser ~~parse~~ parses the XSL stylesheet and create a 'XSLT tree' based on the elements and ~~element~~ attributes used in an XSL document
 3. Use Xpaths to define templates that matches the parts of the source tree
 4. Use XSL processor to transform the matched and put the transformed into result tree.
 5. the result tree is output as a ~~xml~~ result document (HTML document)
- Parts of the source document that not matched by a template are typically copied as unchanged.

The XSL documents must be written ~~on~~ on simple text editor and save with ".xsl" extension

You can add XSL document to XML file by adding the following statement:

`<?xml-stylesheet type="text/xsl" href="filename.xsl" ?>`

XSL Elements :- XSLT provides a number of elements for selecting and formatting data from XML document and few are used to control the processor.

→ the form of XSL Element which select data.

`<xsl:element select = "value" >`

the value is a pattern which matches a node or set of nodes with in XML document

→ Some of the elements provided by XSLT for formatting data are :

- * Stylesheet
- * value-of
- * for-each
- * sort
- * text.

(12)

1. Stylesheet : An XSL stylesheet contains instructions for transforming XML document. This can be included by using :

`<xsl:stylesheet xmlns : "http://www.w3.org/1999/XSL/Transform" version = "1.0" >`

the style sheet element is root element for all XSLT stylesheet

2. xsl:value-of :- the value-of element displays the value of specified element or attribute

Syntax

`<xsl:value-of select = "ele-name/attr-name" />`

→ element name is the name of the element for

for which the value is to be displayed

ex:

```
<xsl:value-of select="Title"/>
```

3. xsl:foreach :- for-each element applies a template repeatedly for each node.

Syntax

```
<xsl:for-each select='pattern' >
```

action to be performed

```
</xsl:for-each>
```

In the above syntax: the pattern can be one of the following:

- * element
- * parent/child
- * ancestor//child

Example:

```
<xsl:for-each select="Title" >
```

```
  <font color="blue" >
```

```
    <xsl:value-of select='Pname' />
```

```
  </font>
```

```
  <xsl:value-of select="Price" />
```

```
</xsl:for-each>
```

XSL Templates :- A template describes how an XML element and its contents are converted into a format that can be displayed in the browser.

— A template consists of two parts

1. A pattern that identifies an element in XML document

2. Action or processing code that shows the transformation of the resulting element.

1. template element:-
Syntax:-

`<xsl:template match="pattern" >`

action to be taken

(13)

`</xsl:template >`

the template element is used to define a template for desired output. the pattern can have multiple values.

They are:

`/` - Starts performing action from root node.

`*` - matches any element in XML document

ele-name - Performs the action when the named element is encountered.

2. xsl:apply-templates: The apply-templates element is used to instructs the XSLT Processor to find an appropriate node/template and performs the specified task on each element

Syntax

`<xsl:apply-templates [select=pattern] >`

3. xsl:attribute: creates an attribute node. this attribute is then applied to the output document

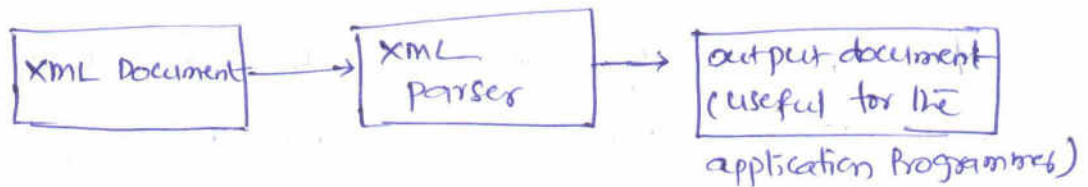
4. xsl:if: boolean conditions can be tested

5. xsl:when: used in conditional testing

6. xsl:cdata: a CDATA section is added to the ~~xml~~ output document.

3.6 XML Parsers : DOM and SAX :-

- Parsing is the process of reading and validating a program written in one format and converting it into desired format.
- Parsing of XML is process of reading and validating an XML document and converting it into desired format. The program that does this job is called an XML Parser or XML Processor.



- The Parsing Process follows the below steps:
 1. XML Parser reads the XML document from the disk and validates it.
 2. Then the XML Parser converts the XML file into an object.
 3. The object is accessed by the application program.
 4. Finally, the application program produces the desired output.

- Two models are commonly used for Parsers

1. DOM parser (Document Object Model)
2. SAX Parser (Simple API for XML)

- These are mostly used by the Java Programmers. Suppose if the application programs are written in Java technology. When an XML program is presented to a Java program as

as an object, the application programmers are confused from where it should start parsing and stop it. ¶

— There are two possibilities to solve this :

1. Present the document in bits and pieces, as and when we encounter a certain sections or portions of the document

2. To present the entire document tree at once, where Java program has to think of this document tree one object.

— The first approach, where it goes through XML document node by node till end of the XML file is called (SAX) Simple API for XML (14)

— The second approach, where an entire document is read into the memory as an object and parses its contents as per the requirement is called DOM (Document Object Model)

DOCUMENT OBJECT MODEL (DOM) :-

The DOM is a platform-independent and language-neutral interface that allows documents and scripts to dynamically access and update the contents and style of the document.

— DOM is a collection of nodes with different types of data where nodes are information units that arrange data in an XML document.

— DOM is used to describe the nodes and the relationship between nodes.

→ DOM works like an API and is divided into 3 levels

Level 1: provides basic functionality for navigating and manipulating HTML and XML documents such as methods for adding, moving and reading information.

Level 2: It introduces support for namespaces and for stylesheets such as CSS and storing data in a hierarchical structure.

Level 3: It produces complete mapping between DOM and XML.

- It reads the entire document and builds DOM tree
- It provides random access to any nodes in the document DOM tree.
- Allows programs and scripts to build documents, navigate their structure and modify or delete elements and content.

XML DOM :-

- The XML DOM is a standard object application programming interface (API) that gives developers a process to control the structure, content and format of XML document.
- When a DOM parser successfully parses an XML document, the parser creates a tree structure in memory that contains document data.
- A DOM Tree has a single root node, that contains all other nodes in the document.

- Each node is an object that has properties, methods and events
- Properties are associated with node provide access to the node's name, value and child nodes.
- Methods allow developers to create, append and delete nodes, load XML document etc and so on.
- Let us explain the XML DOM tree with example. consider the following simple XML document that contains the specification of a hard disk

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

```
<Store>
```

(15)

```
<HDD type="SATA">
```

```
<make> SeaGate </make>
```

```
<Capacity unit="GB"> 500 </Capacity>
```

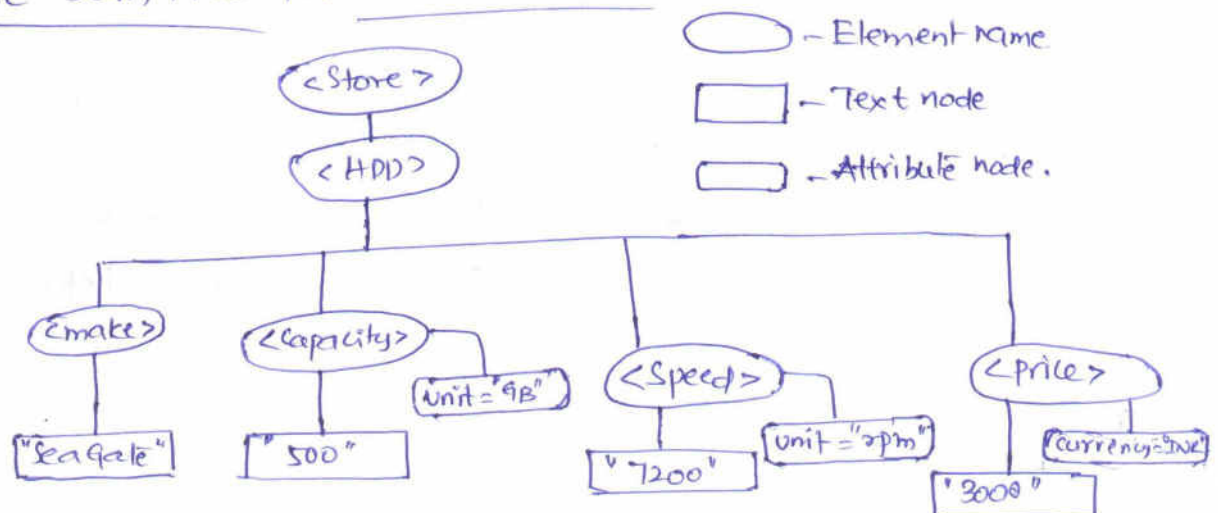
```
<Speed unit="rpm"> 7200 </Speed>
```

```
<Price currency="INR"> 3000 </Price>
```

```
</HDD>
```

```
</Store>
```

- the DOM Tree for above document



- Here Store is root node, HDD is child node where make, price, capacity and speed are siblings.

* Differences between DOM and SAX :-

DOM	SAX
1. It is a Tree-model parser.	1. It is event-based parser.
2. Dom parser stores an entire xml document in memory before parsing	2. SAX parser can read xml document node by node.
3. DOM is read and write.	3. SAX is read-only.
4. Dom is preferred when xml file is small in size.	4. SAX is preferred when xml file is large in size.
5. Backward and forward searching is possible in Dom	5. SAX reads the file from top to bottom and backward navigation is not possible
6. It takes more amount of memory	6. It takes less amount of memory
7. DOM is slow at runtime	7. SAX is fast and efficient
8. It uses the following packages in java. <code>import java.xml.parsers.*;</code> <code>import org.w3c.dom.*;</code>	8. It uses the following packages in java. <code>import org.xml.sax.*;</code> <code>import org.xml.sax.helpers.*;</code>

AJAX

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML, CSS etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

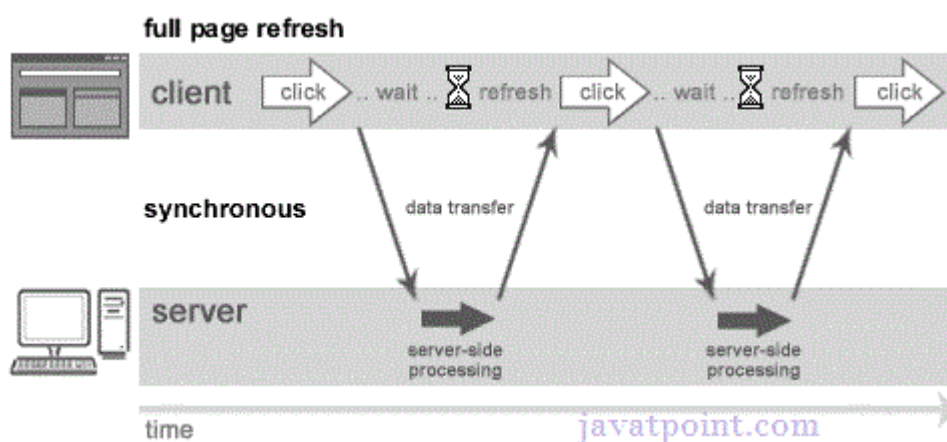
Where it is used?

There are too many web applications running on the web that are using ajax technology like **gmail, facebook, twitter, google map, youtube** etc.

Before understanding AJAX, let's understand classic web application model and ajax web application model first.

Synchronous (Classic Web-Application Model)

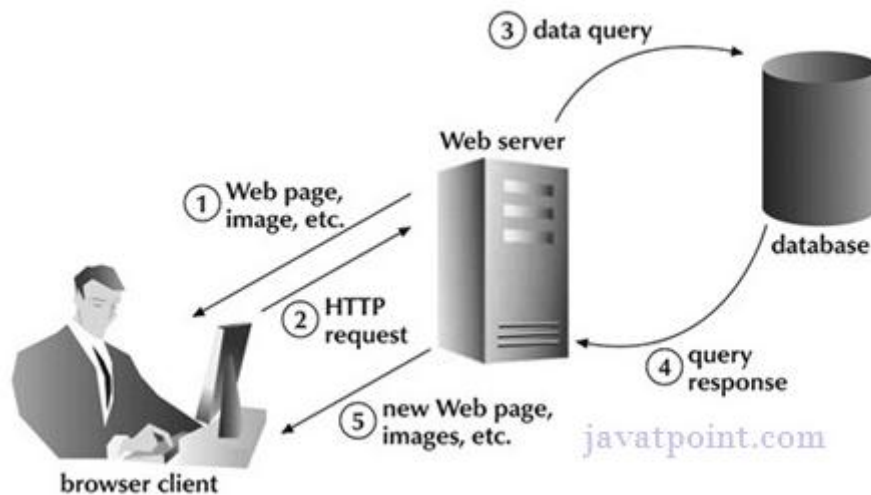
A synchronous request blocks the client until operation completes i.e. browser is not unresponsive. In such case, javascript engine of the browser is blocked.



As you can see in the above image, full page is refreshed at request time and user is blocked until request completes.

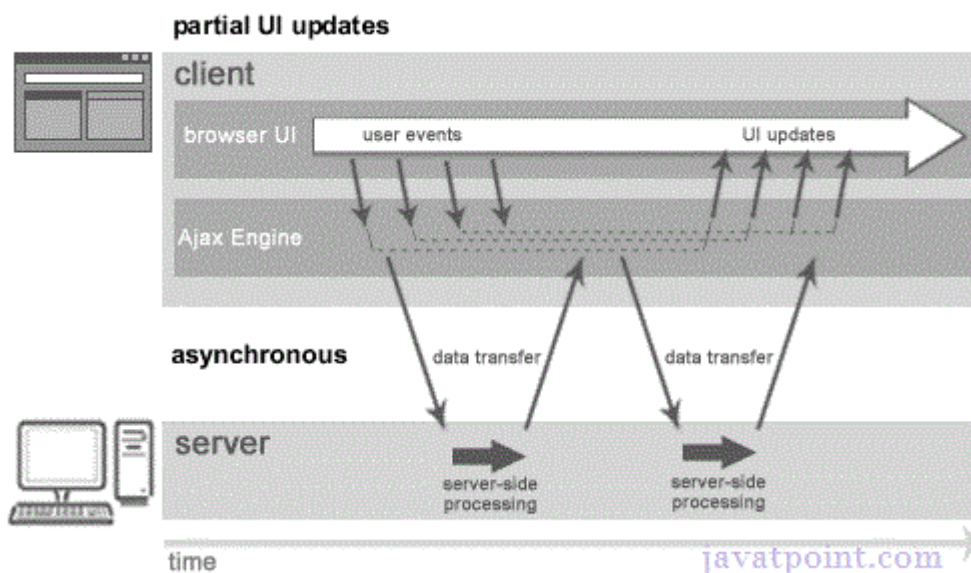
Let's understand it another way.

Let's understand it another way.



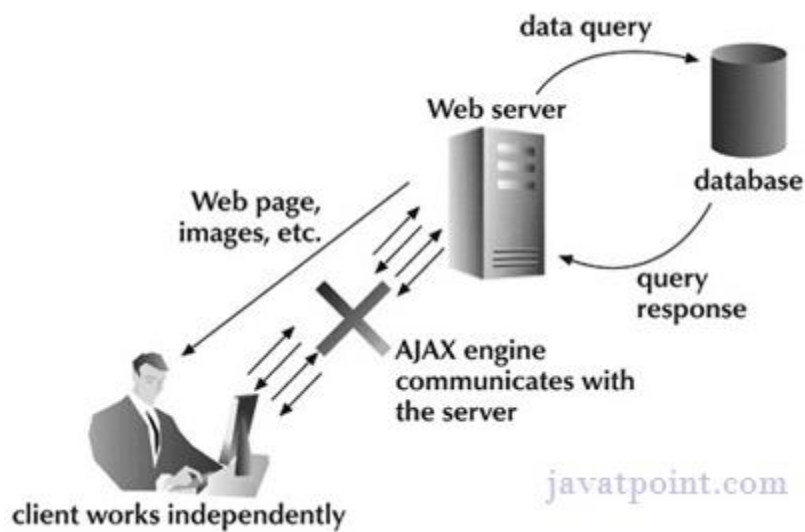
Asynchronous (AJAX Web-Application Model)

An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



As you can see in the above image, full page is not refreshed at request time and user gets response from the ajax engine.

Let's try to understand asynchronous communication by the image given below.



AJAX Technologies

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:

- HTML/XHTML and CSS
- DOM
- XML or JSON
- XMLHttpRequest
- JavaScript

HTML/XHTML and CSS

These technologies are used for displaying content and style. It is mainly used for presentation.

DOM

It is used for dynamic display and interaction with data.

XML or JSON

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest

For asynchronous communication between client and server. For more visit next page.

JavaScript

It is used to bring above technologies together.

Independently, it is used mainly for client-side validation.

XMLHttpRequest

An object of XMLHttpRequest is used for asynchronous communication between client and server.

It performs following operations:

1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

Properties of XMLHttpRequest object

The common properties of XMLHttpRequest object are as follows:

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
	represents the state of the request. It ranges from 0 to 4.
readyState	0 UNOPENED open() is not called. 1 OPENED open is called but send() is not called. 2 HEADERS_RECEIVED send() is called, and headers and status are

available.

3 LOADING Downloading data; responseText holds the data.

4 DONE The operation is completed fully.

responseText returns response as text.

responseXML returns response as XML

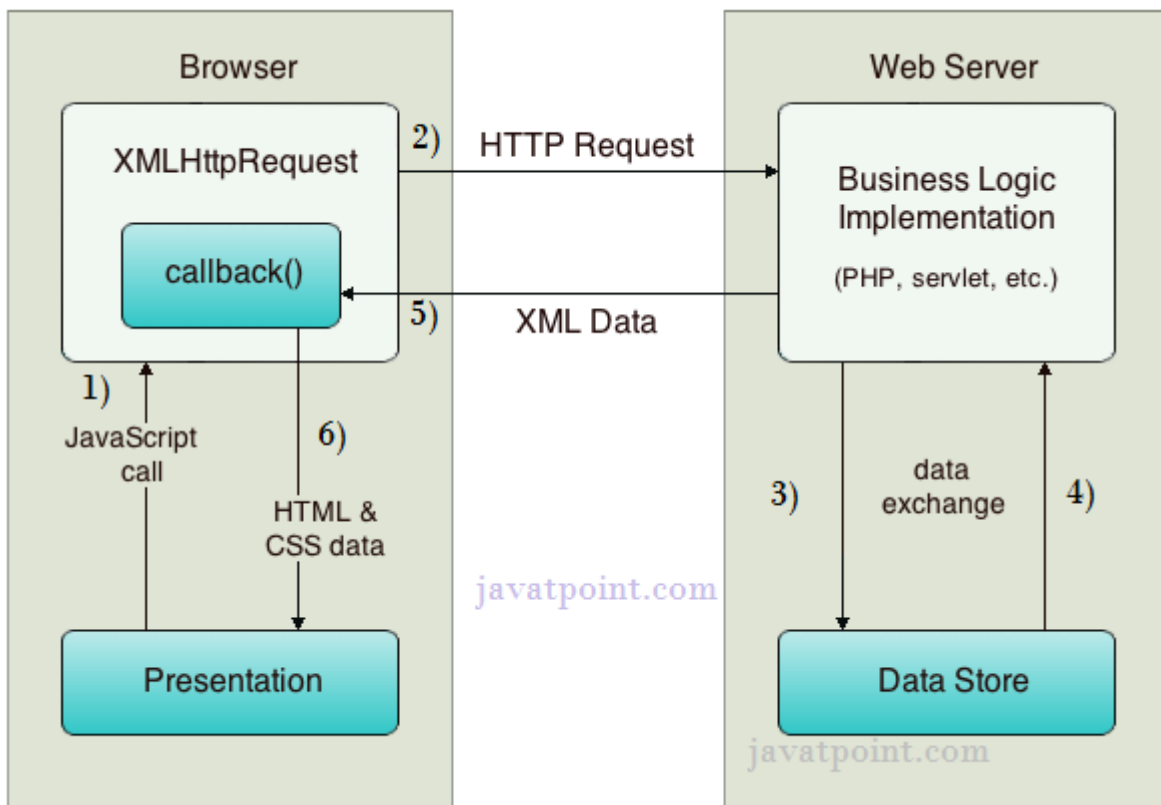
Methods of XMLHttpRequest object

The important methods of XMLHttpRequest object are as follows:

Method	Description
void open(method, URL)	opens the request specifying get or post method and url.
void open(method, URL, async)	same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	same as above but specifies username and password.
void send()	sends get request.
void send(string)	send post request.
setRequestHeader(header,value)	it adds request headers.

How AJAX works?

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

Web Service Components

There are three major web service components.

1. SOAP
2. WSDL
3. UDDI

SOAP

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.

SOAP is a W3C recommendation for communication between two applications.

SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

Advantages of Soap Web Services

WS Security: SOAP defines its own security known as WS Security.

Language and Platform independent: SOAP web services can be written in any programming language and executed in any platform.

Disadvantages of Soap Web Services

Slow: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.

WSDL dependent: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

WSDL

WSDL is an acronym for Web Services Description Language.

WSDL is a xml document containing information about web services such as method name, method parameter and how to access it.

WSDL is a part of UDDI. It acts as a interface between web service applications.

WSDL is pronounced as wiz-dull.

Features of WSDL

- WSDL is an XML-based protocol for information exchange in decentralized and distributed environments.
- WSDL definitions describe how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of Universal Description, Discovery, and Integration (UDDI), an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'.

WSDL Elements

A WSDL document contains the following elements:

- **Definition** : It is the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements described here.
- **Data types** : The data types to be used in the messages are in the form of XML schemas.
- **Message** : It is an abstract definition of the data, in the form of a message presented either as an entire document or as arguments to be mapped to a method invocation.
- **Operation** : It is the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message.
- **Port type** : It is an abstract set of operations mapped to one or more end-points, defining the collection of operations for a binding; the collection of operations, as it is abstract, can be mapped to multiple transports through various bindings.
- **Binding** : It is the concrete protocol and data formats for the operations and messages defined for a particular port type.
- **Port** : It is a combination of a binding and a network address, providing the target address of the service communication.
- **Service** : It is a collection of related end-points encompassing the service definitions in the file; the services map the binding to the port and include any extensibility definitions.

In addition to these major elements, the WSDL specification also defines the following utility elements:

- **Documentation**: This element is used to provide human-readable documentation and can be included inside any other WSDL element.
- **Import** : This element is used to import other WSDL documents or XML Schemas.

UDDI

- UDDI stands for **Universal Description, Discovery, and Integration**.
- UDDI is a specification for a distributed registry of web services.
- UDDI is a platform-independent, open framework.
- UDDI can communicate via SOAP, CORBA, Java RMI Protocol.
- UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative, enabling businesses to discover each other and define how they interact over the Internet.

UDDI has two sections:

- A registry of all web service's metadata, including a pointer to the WSDL description of a service.

- A set of WSDL port type definitions for manipulating and searching that registry.

A business or a company can register three types of information into a UDDI registry. This information is contained in three elements of UDDI.

These three elements are:

- White Pages,
- Yellow Pages, and
- Green Pages.

White Pages

White pages contain:

- Basic information about the company and its business.
- Basic contact information including business name, address, contact phone number, etc.
- A Unique identifiers for the company tax IDs. This information allows others to discover your web service based upon your business identification.

Yellow Pages

- Yellow pages contain more details about the company. They include descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.
- Yellow pages uses commonly accepted industrial categorization schemes, industry codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want.

Green Pages

Green pages contains technical information about a web service. A green page allows someone to bind to a Web service after it's been found. It includes:

- The various interfaces
- The URL locations
- Discovery information and similar data required to find and run the Web service.

NOTE : UDDI is not restricted to describing web services based on SOAP. Rather, UDDI can be used to describe any service, from a single webpage or email address all the way up to SOAP, CORBA, and Java RMI services.

UDDI - Technical Architecture

The UDDI technical architecture consists of three parts:

UDDI Data Model

UDDI Data Model is an XML Schema for describing businesses and web services. The data model is described in detail in the "UDDI Data Model" chapter.

UDDI API Specification

It is a specification of API for searching and publishing UDDI data.

UDDI Cloud Services

These are operator sites that provide implementations of the UDDI specification and synchronize all data on a scheduled basis.

The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system built from multiple nodes having their data synchronized through replication.

The current cloud services provide a logically centralized, but physically distributed, directory. It means the data submitted to one root node will automatically be replicated across all the other root nodes. Currently, data replication occurs every 24 hours.

UDDI cloud services are currently provided by Microsoft and IBM. Ariba had originally planned to offer an operator as well, but has since backed away from the commitment. Additional operators from other companies, including Hewlett-Packard, are planned for the near future.

It is also possible to set up private UDDI registries. For example, a large company may set up its own private UDDI registry for registering all internal web services. As these registries are not automatically synchronized with the root UDDI nodes, they are not considered as a part of the UDDI cloud.

UDDI - With WSDL

The UDDI data model defines a generic structure for storing information about a business and the web services it publishes. The UDDI data model is completely extensible, including several repeating sequence structures of information.

However, WSDL is used to describe the interface of a web service. WSDL is fairly straightforward to use with UDDI.

- WSDL is represented in UDDI using a combination of *businessService*, *bindingTemplate*, and *tModel* information.
- As with any service registered in UDDI, generic information about the service is stored in the *businessService* data structure, and information specific to how and where the service is accessed is stored in one or more associated *bindingTemplate*

structures. Each *bindingTemplate* structure includes an element that contains the network address of the service and has associated with it one or more *tModel* structures that describe and uniquely identify the service.

- When UDDI is used to store WSDL information, or pointers to WSDL files, the *tModel* should be referred to by convention as type *wsdlSpec*, meaning that the *overviewDoc* element is clearly identified as pointing to a WSDL service interface definition.
- For UDDI, WSDL contents are split into two major elements the interface file and the implementation file.

The Hertz reservation system web service provides a concrete example of how UDDI and WSDL works together. Here is the <tModel> for this web service: