

Java Script: An introduction to JavaScript–JavaScript DOM Model–Date and Objects,- Regular Expressions- Exception Handling-Validation-Built-in objects-Event Handling- DHTML with JavaScript.

Servlets: Java Servlet Architecture- Servlet Life Cycle- Form GET and POST actions- Session Handling- Understanding Cookies- Installing and Configuring Apache Tomcat Web Server;-DATABASE CONNECTIVITY: JDBC perspectives, JDBC program example

JSP: Understanding Java Server Pages-JSP Standard Tag Library(JSTL)-Creating HTML forms by embedding JSP code.

INTRODUCING JAVASCRIPT

JavaScript was developed by Netscape Communications Corporation, the maker of the Netscape web browser. JavaScript was the first web scripting language to be supported by browsers, and it is still by far the most popular.

JavaScript was originally called LiveScript and was first introduced in Netscape Navigator 2.0 in 1995. Netscape Navigator was an early browser that held a large share of the browser market. It was soon renamed JavaScript to indicate a marketing relationship with Sun's Java language.

JavaScript is almost as easy to learn as HTML, and it can be included directly in HTML documents.

What can you do with Javascript?

- Display messages to the user as part of a web page, in the browser's status line, or in alert boxes.
- Validate the contents of a form and make calculations (for example, an order form can automatically display a running total as you enter item quantities).
- Animate images or create images that change when you move the mouse over them.
- Create ad banners that interact with the user, rather than simply displaying a graphic.
- Detect the browser in use or its features and perform advanced functions only on browsers that support them.
- Detect installed plug-ins and notify the user if a plug-in is required.
- Modify all or part of a web page without requiring the user to reload it.
- Display or interact with data retrieved from a remote server.

JavaScript Browser Support

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. The procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera are listed below.

Where to put JavaScript?

JavaScript can be put in the <head> or in the <body> of an HTML document

- JavaScript *functions* should be defined in the <head>. This ensures that the function is loaded before it is needed
- JavaScript in the <body> will be executed as the page loads

JavaScript can be put in a separate .js file

- <script src="myJavaScriptFile.js"></script>

Put this HTML wherever you would put the actual JavaScript code

An external .js file lets you use the same JavaScript on multiple HTML pages

The external .js file cannot itself contain a <script> tag

JavaScript can be put in HTML *form object*, such as a button

This JavaScript will be executed when the form object is used

JavaScript Programs

A **computer program** is a list of "instructions" to be "executed" by the computer.

In a programming language, these program instructions are called **statements**.

JavaScript is a **programming language**.

JavaScript statements are separated by **semicolons**.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>JavaScript Statements</h1>
```

```
<p>Statements are separated by semicolons.</p>
```

```
<p>The variables x, y, and z are assigned the values 5, 6, and 11:</p>
```

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

```
<script>
```

```
var x = 5;
var y = 6;
var z = x + y;
document.getElementById("demo").innerHTML = z;
</script>
</body>
</html>
```

JAVASCRIPT STATEMENTS

JavaScript statements are composed of:

Values, Operators, Expressions, Keywords, and Comments.

JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values.

Fixed values are called **literals**. Variable values are called **variables**.

JavaScript Literals

- Boolean literals
- Floating-point literals
- Integers
- Object literals
- String literals

JAVASCRIPT VARIABLES

In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var** keyword to **define** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

```
var x;
x = 6;
```

Variables names must begin with a letter or underscore

Variable names are case-sensitive. All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

```
lastName = "Doe";
```

```
lastname = "Peterson";
```

Variables are *untyped* (they can hold values of any type)

Variables declared within a function are local to that function (accessible only within that function)

Variables declared outside a function are global (accessible from anywhere on the page)

JAVASCRIPT DATA TYPES


- JavaScript has three “primitive” types: number, string, and boolean
- Everything else is an object
- Numbers are always stored as floating-point values
 - Hexadecimal numbers begin with 0x
 - Some platforms treat 0123 as octal, others treat it as decimal
- Strings may be enclosed in single quotes or double quotes
 - Strings can contains \n (newline), \" (double quote), etc.
- Booleans are either true or false
 - 0, "0", empty strings, undefined, null, and NaN are false , other values are true

JAVASCRIPT OBJECTS

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods
	car.name = Fiat car.model = 500 car.weight = 850kg car.color = white	car.start() car.drive() car.brake() car.stop()

All cars have the same **properties**, but the property values differ from car to car.

All cars have the same **methods**, but the methods are performed at different times.

Objects are variables too. But objects can contain many values.

```
var car = {type:"Fiat", model:500, color:"white"};
```

The values are written as **name:value** pairs (name and value separated by a colon).

The name:values pairs (in JavaScript objects) are called **properties**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Accessing Object Properties

You can access object properties in two ways:

objectName.propertyName

or

objectName[propertyName]

Accessing Object Methods

You access an object method with the following syntax:

objectName.methodName()

JAVASCRIPT OPERATORS

- Arithmetic operators:

+ - * / % ++ --

- Comparison operators:

< <= == != >= >

- Logical operators:

&& || ! (&& and || are *short-circuit* operators)

- Bitwise operators:

& | ^ ~ << >> >>>

- Assignment operators:

+= -= *= /= %= <<= >>= >>>= &= ^= |=

- Associativity.

JAVASCRIPT CONTROL STATEMENTS (Syntax Refer Unit 1)

- Sequential execution

– Statements execute in the order they are written

- Transfer of control

– Next statement to execute may not be the next one in sequence

- Three control structures

– Sequence structure

– Selection structure

- if
- if...else
- switch

– Repetition structure

- while
- do...while
- for
- for...in

JAVASCRIPT FUNCTIONS

Defining functions

- All variables declared in function are called local
 - Do not exist outside current function
- Parameters
 - Also local variables
- Promotes reusability
 - Keep short
 - Name clearly

Format of a function definition

```
function function-name( parameter-list )  
{  
    declarations and statements  
}
```

- Function name any valid identifier
- Parameter list names of variables that will receive arguments
 - Must have same number as function call
 - May be empty
- Declarations and statements
 - Function body (“block” of code)

Returning control

- return statement
- Can return either nothing, or a value

return *expression*;

- No return statement same as return;
- Not returning a value when expected is an error

Recursive Function: Function Calling Themselves

Eg: function fact (a)

```
{  
    if(a==1||a==0) return 1;  
    else return a*fact (a-1);  
}
```

Date Object

The JavaScript date object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

You can use different Date constructors to create date object. It provides methods to get and set day, month, year, hour, minute and seconds.

Constructor

You can use 4 variant of Date constructor to create date object.

Date()

Date(milliseconds)

Date(dateString)

Date(year, month, day, hours, minutes, seconds, milliseconds)

JavaScript Date Methods

Method	Description
getFullYear()	returns the year in 4 digit e.g. 2015. It is a new method and suggested than getYear() which is now deprecated.
getMonth()	returns the month in 2 digit from 0 to 11. So it is better to use getMonth()+1 in your code.
getDate()	returns the date in 1 or 2 digit from 1 to 31.
getDay()	returns the day of week in 1 digit from 0 to 6.
getHours()	returns all the elements having the given name value.
getMinutes()	returns all the elements having the given class name.
getSeconds()	returns all the elements having the given class name.

Simple Example:

```
Current Time: <span id="txt"></span>
<script>
var today=new Date();
var h=today.getHours();
var m=today.getMinutes();
var s=today.getSeconds();
document.getElementById('txt').innerHTML=h+":"+m+": "+s;
</script>
```

Great User Based on time

```
<script language="JavaScript">
var name=prompt("Enter Your Name, Please.", "name");
var today = new Date ()
var hrs = today.getHours();
document.writeln("<b>");
document.writeln("<b>");
if (hrs <= 12)
    document.write("Good Morning "+name+ "!");
else if (hrs <= 18)
    document.write("Good Afternoon "+name+ "!");
else
    document.write("Good Evening "+name+ "!"); document.writeln("<br />");
</script>
```

String

The **JavaScript string** is an object that represents a sequence of characters. There are 2 ways to create string in JavaScript

1. By string literal
var **stringname**="string value";
2. By string object (using new keyword)
var **stringname**=new String("string literal");

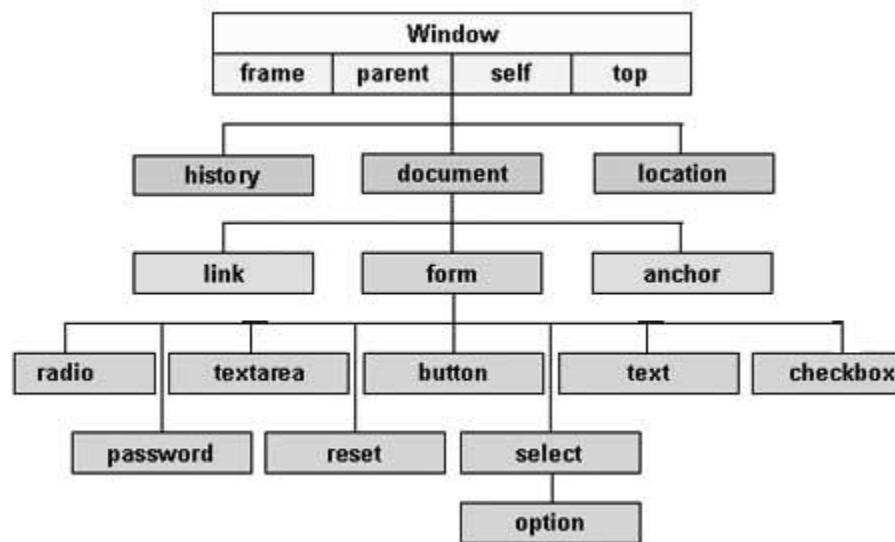
Let's see the list of JavaScript string methods with examples.

- o charAt(index)
- o concat(str)
- o indexOf(str)
- o lastIndexOf(str)
- o toLowerCase()
- o toUpperCase()
- o slice(beginIndex, endIndex)
- o trim()

A **Document object** represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

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



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
write("string")	writes the given string on the document.
writeln("string")	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.

The different node methods available through DOM manipulation are as follows:

`node.childNodes`

`node.firstChild`

`node.lastChild`

`node.parentNode`

`node.nextSibling`

`node.previousSibling`

Suppose you have this XHTML:

```
<ul id="list">
  <li><a href="link1.html" class="link_one">Link Number One</a></li>
  <li><a href="link2.html">Link Number Two</a></li>
  <li><a href="link3.html">Link Number Three</a></li>
  <li><a href="link4.html">Link Number Four</a></li>
</ul>
```

We can access the first link in our unordered list using any one of the following 3 sections of code:

```
var myLinkList = document.getElementById("list");
var myFirstLink = myLinkList.childNodes[0].childNodes[0];
alert(myFirstLink.className);
var myLinkList = document.getElementById("list");
var myFirstLink = myLinkList.firstChild.firstChild;
alert(myFirstLink.className);
```

createElement

This method does exactly what it says: it creates an element and allows you to place that new element anywhere in the DOM structure.

Using the list example from above, we could add a new list item using the following code:

```
var myNewListItem = document.createElement("li");
var myNewLink = document.createElement("a");
```

appendChild

Let's add the two elements we just created to our list of links using the **appendChild** method:

```
var myNewListItem = document.createElement("li");
var myNewLink = document.createElement("a");
```

removeChild

In order to remove the element that we just created, we would use the following code:

```
var myLinkList = document.getElementById("list");  
var myRemovedLink = myLinkList.lastChild;  
myLinkList.removeChild(myRemovedLink)
```

innerHTML is a way of accessing, for the purpose of writing to, the content inside of an XHTML element. It comes in handy with Ajax requests because you can choose an element on your page, then write the Ajax-loaded content straight into that element via **innerHTML**

```
var myContentHolder = document.getElementById("content");  
myContentHolder.innerHTML = "<p>This is the dynamic content created by the innerHTML  
property</p>";
```

Window Object

The **window object** represents a window in browser. An object of window is created automatically by the browser. Window is the object of browser, **it is not the object of javascript**. The javascript objects are string, array, date etc

Methods of window object

Method	Description
alert()	displays the alert box containing message with ok button.
confirm()	displays the confirm dialog box containing message with ok and cancel button.
prompt()	displays a dialog box to get input from the user.
open()	opens the new window.
close()	closes the current window.
setTimeout()	performs action after specified time like calling function, evaluating expressions et

Event

- Events are the actions that occur as a result of browser activities or user interactions with the web pages.
 - Such as the user performs an action (mouse click or enters data)
 - We can validate the data entered by a user in a web form
 - Communicate with Java applets and browser plug-ins

Event Categories

- Keyboard and mouse events
 - Capturing a mouse event is simple
- Load events
 - The page first appears on the screen: “Loads”, leaves: “Unloads”, ...
- Form-related events
 - onFocus() refers to placing the cursor into the text input in the form.
 - Others -Errors, window resizing.

Event Handler

- When an event occurs, a code segment is executed in response to a specific event is called “event handler”.
- Event handler names are quite similar to the name of events they handle.
- E.g the event handler for the “click” event is “onClick”.
- <HTMLtag eventhandler=“JavaScript Code”>

Event Handler	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.

Event bubbling

- The process whereby events fired on *child* elements “bubble” up to their *parent* elements
- When an event is fired on an element, it is first delivered to the element’s event handler (if any), then to the parent element’s event handler (if any)

Regular Expression is an object that describes a pattern of characters.

The JavaScript **RegExp** class represents regular expressions, and both **String** and **RegExp** define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax

A regular expression could be defined with the **RegExp ()** constructor, as follows –

```
var pattern = new RegExp(pattern, attributes);
```

or simply

```
var pattern = /pattern/attributes;
```

Here is the description of the parameters –

- **pattern** – A string that specifies the pattern of the regular expression or another regular expression.
- **attributes** – An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

Brackets

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
[...]	Any one character between the brackets.
[^...]	Any one character not between the brackets.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character

Write a JavaScript for form validation name length, email and phone number etc

```
<html>
<head>
<title>Form Validation</title>
<script>
function validate()
{
    var emailID = document.myForm.EMail.value;
    var y=document.myForm.ph.value;
    atpos = emailID.indexOf("@");
    dotpos = emailID.lastIndexOf(".");
    var r=/\d{2}-\d{3}-\d{8}/;
    m=r.test(y);
    if( document.myForm.Name.value == "" )
    alert( "Please provide your name!" );
    else if( document.myForm.EMail.value == "" )
    alert( "Please provide your Email!" );
    else if (atpos < 1 || ( dotpos - atpos < 2 ))
    alert("Please enter correct email ID")
    else if(document.myForm.Zip.value
    ==""||document.myForm.Zip.value.length!=5)
    alert( "Please provide a zip in the format #####." )
    else if( document.myForm.Country.value == "-1" )
    alert( "Please provide your country!" );
    else if(!m)
    alert( "Please provide Correct Phone Number!" );

}

</script>
</head>
<body>
<form name="myForm" onsubmit="validate();">
<table cellpadding="2" cellspacing="2" border="1">
<tr>
<td align="right">Name</td>
<td><input type="text" name="Name" /></td>
</tr>
<tr>
<td align="right">EMail</td>
<td><input type="text" name="EMail" /></td>
</tr>
<tr>
<td align="right">Zip Code</td>
<td><input type="text" name="Zip" /></td>
</tr>
<tr>
```



```

<td align="right">Country</td>
<td>
  <select name="Country">
    <option value="-1" selected>[choose yours]</option>
    <option value="1">USA</option>
    <option value="2">UK</option>
    <option value="3">INDIA</option>
  </select>
</td>
</tr>
<tr><td>Phone(XX-XXX-XXXXXX)</td>
  <td><input type="text" name="ph" /></td></tr>
<tr>
  <td align="right"></td>

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

```

Ouput

The screenshot shows a web browser window with multiple tabs. The active tab displays a form titled 'Form Validation'. The form has the following fields and values:

Name	mohan
EMail	itmoha@gmail.com
Zip Code	631601
Country	USA
Phone(XX-XXX-XXXXXX)	91-044-37
Submit	

A JavaScript alert box is displayed over the form, with the message: "This page says: Please provide a zip in the format #####." The browser's address bar shows the URL: file:///C:/Users/my-pc/Desktop/val.htm?Name=mohan&EMail=itmoha%40gmail.com&Zip=63160&Country=1&ph=12.

Write a JavaScript i)sum of n natural number ii)sum of n odd numbers

```
<html>
<head>
<script type = "text/javascript">
    var num = window.prompt("Enter the number:", "");
    var n = parseInt(num);
    sum=sumnaturalno(n);
    sum1=sumnoddno(n);
document.writeln("<br>The sum of Numer is "+sum +"<br>The sum of odd
Numer is"+sum1 );
function sumnoddno(n)
{
    var i;
    sum1=0;
    for(i = 1;i <= n; i++)
    {
        if(i%2!=0)
        {
            document.writeln("<br> The Numer is "+i);
            sum1=sum1+i;
        }
    }
    return sum1;
}
function sumnaturalno(n)
{
    var i;
    sum=0;
    for(i = 1;i <= n; i++)
    {
        document.writeln("<br> The Numer is "+i);
        sum=sum+i;
    }
    return sum;
}
</script>
</head>
</html>
```

Exception Handling

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable thing

try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The **finally** statement lets you execute code, after try and catch, regardless of the result:

The JavaScript statements **try** and **catch** come in pairs:

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}  
finally {  
    Block of code to be executed regardless of the try / catch result  
}
```

The throw Statement

The **throw** statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
throw "Too big"; // throw a text
```

```
throw 500; // throw a number
```

```
<html>
```

```
<body><p>Please input a number between 5 and 10:</p>
```

```
<input id="demo" type="text">
```

```
<button type="button" onclick="myFunction()">Test Input</button>
```

```
<p id="message"></p><script>
```

```
function myFunction() {
```

```
    var message, x;
```

```
    message = document.getElementById("message");
```

```
    message.innerHTML = "";
```

```
    x = document.getElementById("demo").value;
```

```
    try {
```

```
        if(x == "") throw "empty";
```

```
        if(isNaN(x)) throw "not a number";
```

```
        x = Number(x);
```

```
        if(x < 5) throw "too low";
```

```
        if(x > 10) throw "too high";
```

```
    }
```

```
    catch(err) {
```

```
        message.innerHTML = "Input is " + err;
```

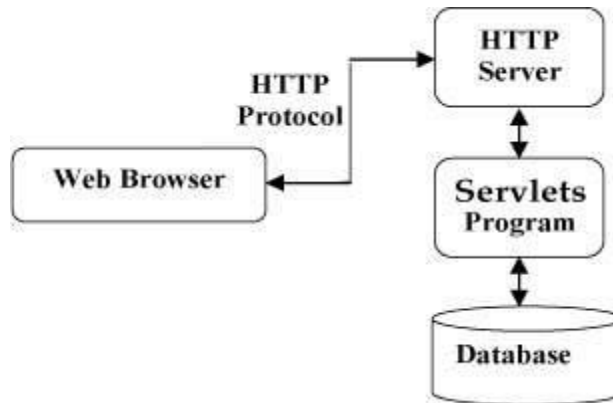
```
    }</script></body></html>
```

SERVLETS:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Servlets Architecture:

Following diagram shows the position of Servlets in a Web Application.



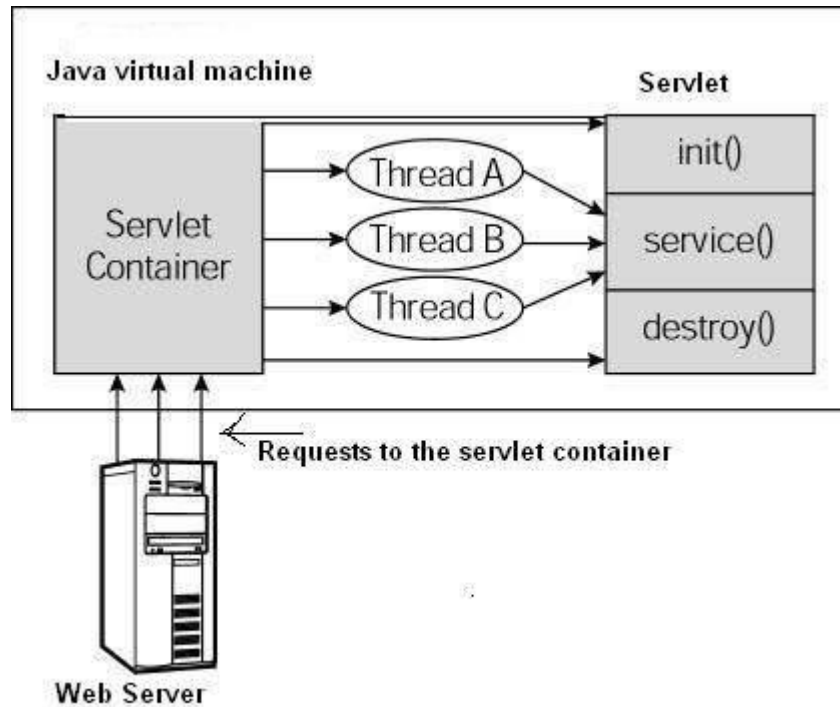
Servlets Tasks:

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

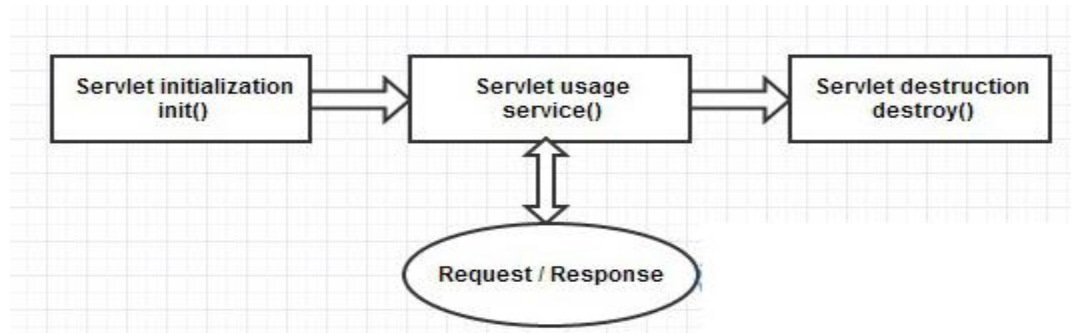
Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http**



A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet



The servlet is initialized by calling the `init ()` method.

The servlet calls `service()` method to process a client's request.

The `service ()` method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls `doGet`, `doPost`, `doPut`, `doDelete`, etc. methods as appropriate.

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

{

}

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException  
  
{  
  
// Servlet code  
  
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
  
// Servlet code  
  
}
```

The destroy() method :The destroy() method is called only once at the end of the life cycle of a servlet.

```
import java.io.*;  
import javax.servlet.*;  
import javax.servlet.http.*;  
/ Extend HttpServlet class  
public class HelloWorld extends HttpServlet  
{  
    private String message;  
    public void init() throws ServletException  
    {  
        // Do required initialization  
        message = "Hello World";  
    }  
}
```

```

public void doGet(HttpServletRequest r1, HttpServletResponse r2) throws ServletException,
IOException
{
    // Actual logic goes here.
    PrintWriter out = r2.getWriter();
    out.println("<h1>" + message + "</h1>");
}

```

```

public void destroy()
{
    // do nothing.
}

```

web.xml file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```

<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>

```



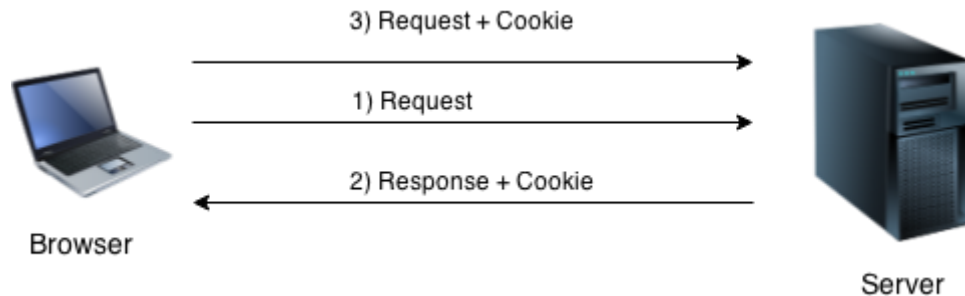
Cookies in Servlet

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

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

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

- Non-persistent cookie-valid for single session
- Persistent cookie-valid for multiple session

Advantage of Cookies

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Cookie class

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

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

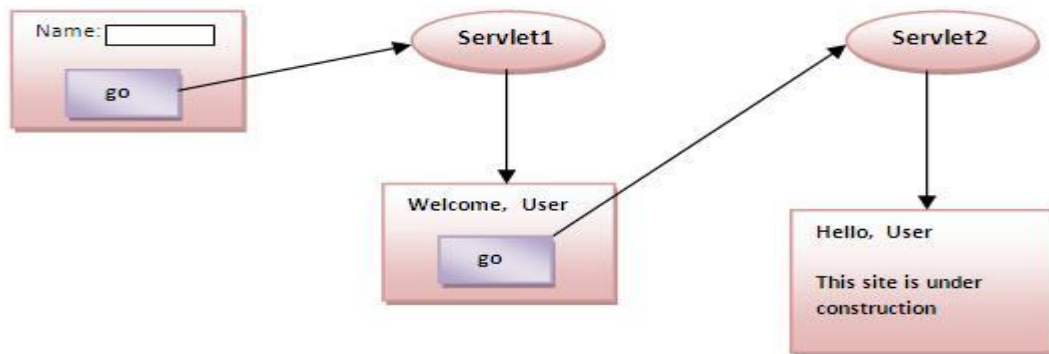
Useful Methods of Cookie class

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

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```

<form action="/FirstServlet" method="post">
  Name:<input type="text" name="userName"/><br/>
  <input type="submit" value="go"/>
</form>

```

FirstServlet.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest r1, HttpServletResponse r2) throws ServletException,
  IOException
  {
    PrintWriter out = response.getWriter();
    String n=r2.getParameter("userName");
    out.print("Welcome "+n);
    Cookie ck=new Cookie("uname",n);//creating cookie object
    r2.addCookie(ck);//adding cookie in the response
    //creating submit button
    out.print("<form action='/SecondServlet'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();

  }
}

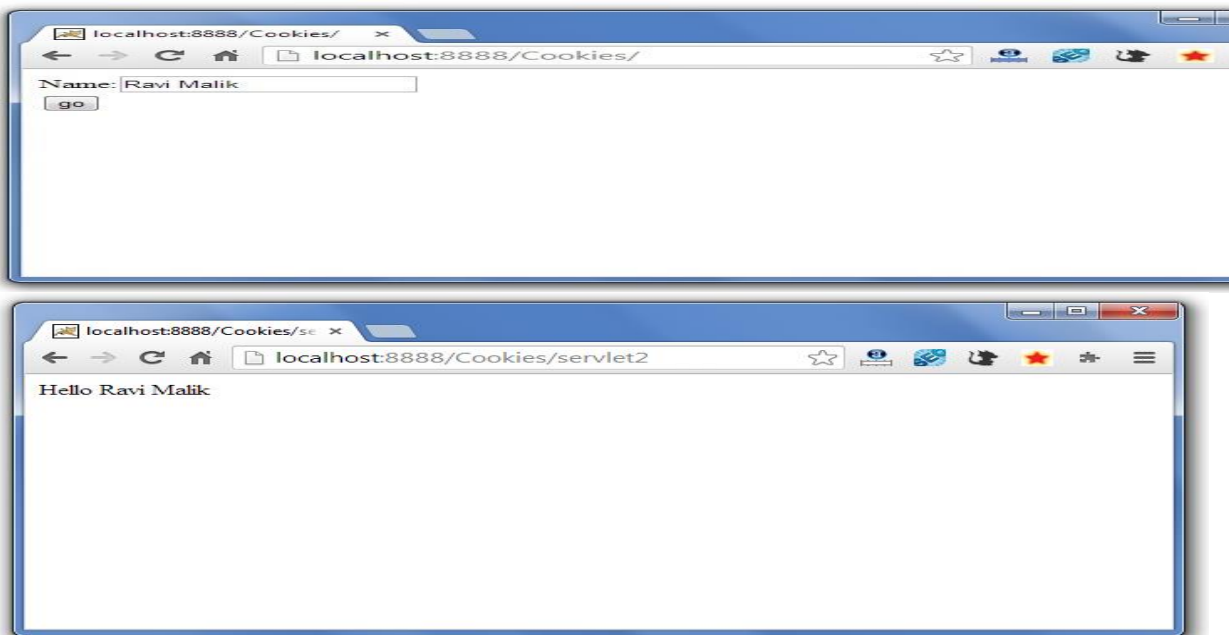
```

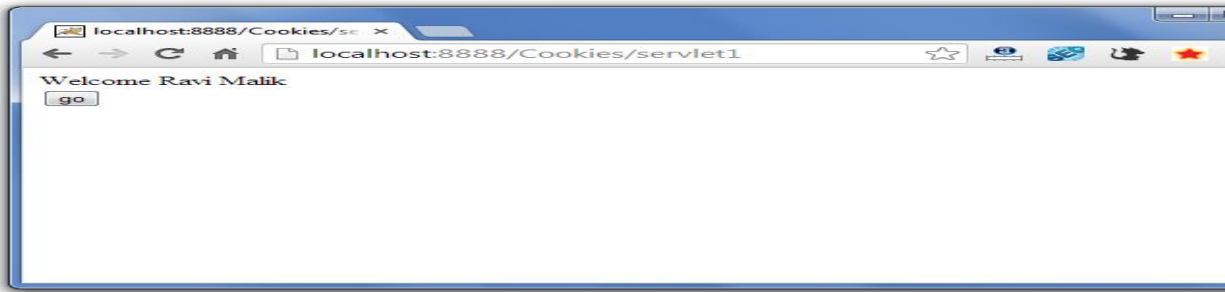
SecondServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SecondServlet extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException
    {
        PrintWriter out = response.getWriter();
        Cookie ck[]=request.getCookies();
        out.print("Hello "+ck[0].getValue());
        out.close();
    }
}
```

Output



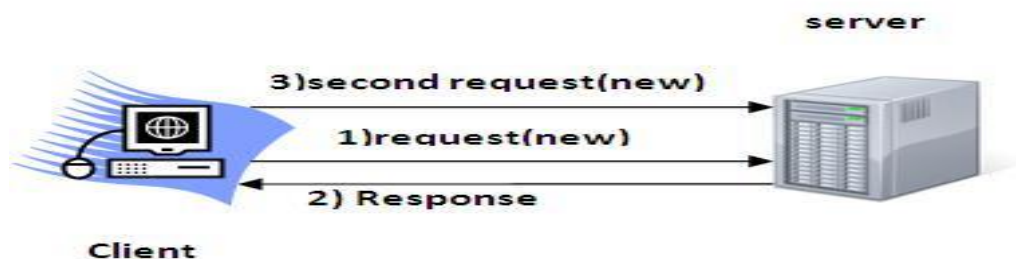


Session simply means a particular interval of time.

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

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Method	Description
<code>public HttpSession getSession()</code>	Will cause one session to be created.
<code>public HttpSession getSession(boolean)</code>	true = will cause one to be created; false = will return null (no session)
<code>public String getRequestedSessionId()</code>	Gets the ID assigned by the server to the session
<code>public Boolean isRequestedSessionIdValid()</code>	Returns true if the request contains a valid session ID
<code>public Boolean</code>	Returns true if the session ID was sent as part

isRequestedSessionIdFromCookie()	of a cookie
public Boolean isRequestedSessionIdFromURL()	Returns true if the session ID was sent through URL rewriting

Please enter your username and password

```
<form action="/ServletSession/Login" method="POST">
  <p><input type="text" name="username" length="40">
  <p><input type="password" name="password" length="40">
  <p><input type="submit" value="Submit">
</form>
```

```
public class LoginSES extends HttpServlet {
    @Override
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
    {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        // Get the session - if no session exists create one
        HttpSession session = request.getSession(true);
        // Set some attribute values to the session
        // In this case user and password from the request and client
        session.setAttribute("username", username);
        session.setAttribute("password", password);

        try {
            response.setContentType("text/html");
            PrintWriter writer = response.getWriter();
            writer.println("<html><body>");
            writer.println("Thank you, " + username +
                ". You are now logged into the system");
            // Encodes the specified URL by including the session ID in it,
            // or, if encoding is not needed, returns the URL unchanged
            String newURL = response.encodeURL("/ServletSession/GetSession");
            // Return a <a> tag with the new url
            writer.println("Click <a href=\"" + newURL +
                "\">here</a> for another servlet");
            writer.println("</body></html>");
        }
    }
}
```

```

writer.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

Installing Apache Tomcat and configuring server

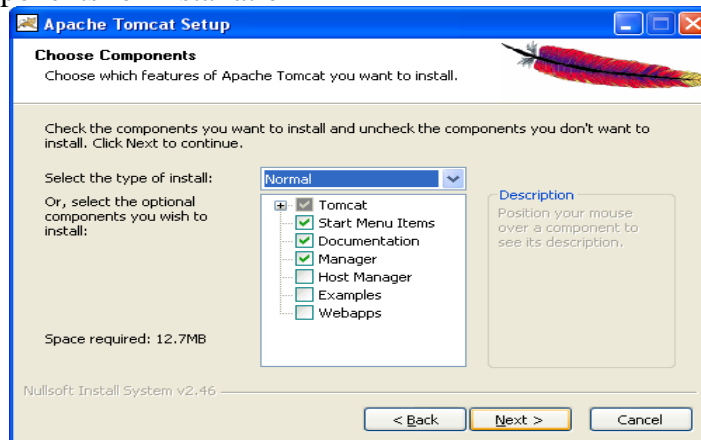
The following is the steps to be followed in installing and configuring a server (Apache Tomcat 5.5.3)

- ☐ Download the Server from <http://tomcat.apache.org/download-55.cgi>
- ☐ Then double click on the **apache-tomcat-5.5.31.exe** application the following installing and configuring screens will appear.

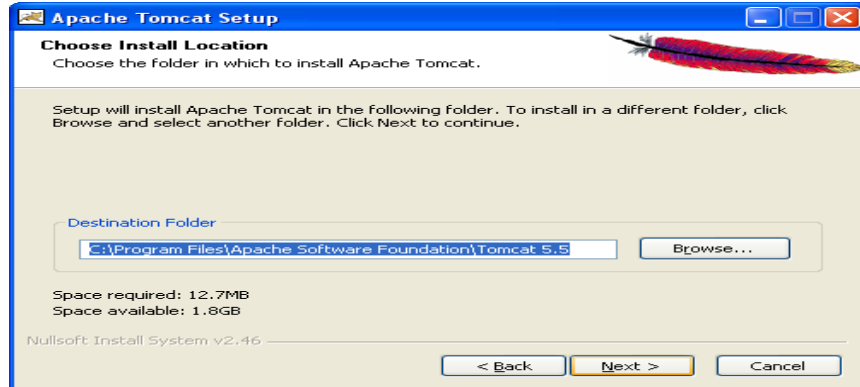
1.Beginning of Installation



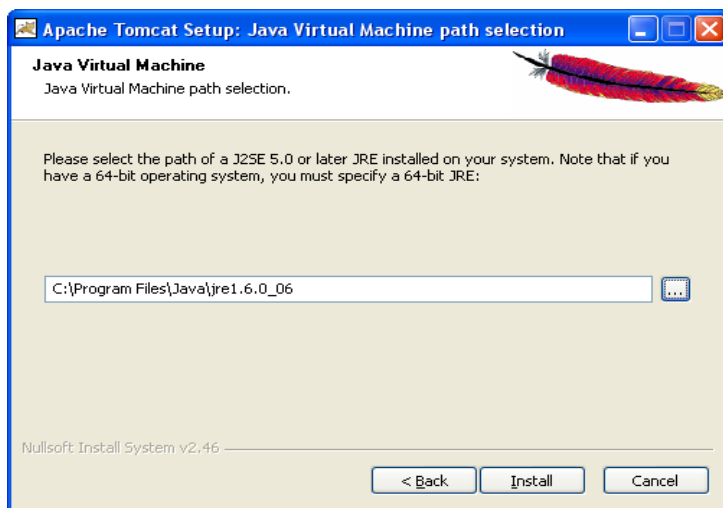
2.Choosing the Components for Installation



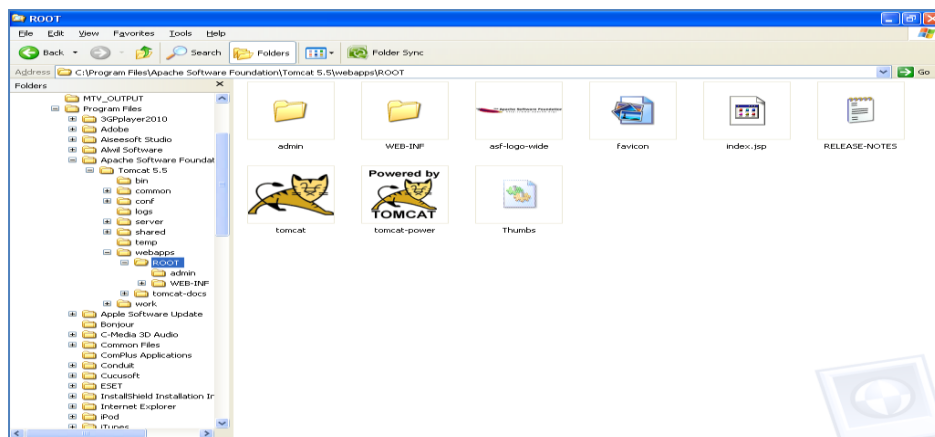
3. Select The Installation Path



4. Configuring the jre folder into the server



5. Now the final installation is completed and the installed files is viewed in the destination folder (refer following Screen)



The JDBC API is comprised of two Java packages:

java.sql and javax.sql.

The following are core JDBC classes, interfaces, and exceptions in the java.sql package:

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

1. Register the driver class
2. Creating connection
3. Creating statement
4. Executing queries
5. Closing connection

1) Register the driver class

The `forName ()` method of `Class` class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of `forName()` method

`public static void forName(String className)throws ClassNotFoundException`

Example to register the `OracleDriver` class

`Class.forName("oracle.jdbc.driver.OracleDriver");`

2) Create the connection object

The `getConnection()` method of `DriverManager` class is used to establish connection with the database.

Syntax of `getConnection()` method

`public static Connection getConnection(String url)throws SQLException`

`public static Connection getConnection(String url,String name,String password) throws SQLException`

Example to establish connection with the Oracle database

`Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","password");`

JDBC driver names and database URL.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	<code>jdbc:mysql://hostname/ databaseName</code>
ORACLE	oracle.jdbc.driver.OracleDriver	<code>jdbc:oracle:thin:@hostname:port Number:databaseName</code>

MsAccess	sun.jdbc.odbc.JdbcOdbcDriver	jdbc:odbc:dsname
----------	------------------------------	------------------

3) Create the Statement object

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

Syntax of createStatement() method

public Statement createStatement()throws SQLException

Example to create the statement object

Statement stmt=con.createStatement();

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database.

This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

public ResultSet executeQuery(String sql)throws SQLException

Example to execute query

ResultSet rs=stmt.executeQuery("select * from emp");

```
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) Close the connection object

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

Syntax of close() method

public void close()throws SQLException

Example to close connection

con.close();

Simple Example :

```
create table Employees (
```

```
    id int not null,
```

```
    age int not null,
```

```
    first varchar (255),
```

```
    last varchar (255)
```

```
);
```

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
```

```
Query OK, 1 row affected (0.05 sec)
```

```
mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
```

```
Query OK, 1 row affected (0.00 sec)
```

```
// Loading required libraries
```

```
import java.io.*;
```

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

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.sql.*;
```

```
public class DatabaseAccess extends HttpServlet{
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
```

```
        // JDBC driver name and database URL
```

```
        static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
```

```
        static final String DB_URL="jdbc:mysql://localhost/TEST";
```

```
        // Database credentials
```

```
        static final String USER = "root";
```

```
        static final String PASS = "password";
```

```

// Set response content type
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Database Result";

String docType =
    "<!doctype html public "-//w3c//dtd html 4.0 " + "transitional//en">\n";

out.println(docType +
    "<html>\n" +
    "<head><title>" + title + "</title></head>\n" +
    "<body bgcolor = \"#f0f0f0\">\n" +
    "<h1 align = \"center\">" + title + "</h1>\n");
try {
    // Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    // Open a connection
    Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);

    // Execute SQL query
    Statement stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    ResultSet rs = stmt.executeQuery(sql);

    // Extract data from result set
    while(rs.next()){
        //Retrieve by column name
        int id = rs.getInt("id");
        int age = rs.getInt("age");
        String first = rs.getString("first");
        String last = rs.getString("last");

        //Display values
        out.println("ID: " + id + "<br>");
        out.println(", Age: " + age + "<br>");
        out.println(", First: " + first + "<br>");
        out.println(", Last: " + last + "<br>");
    }
}

```

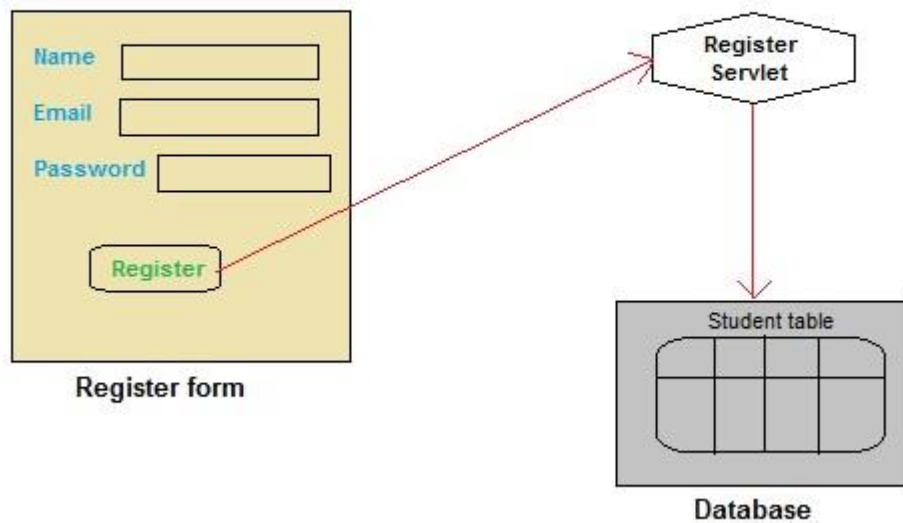
```

        out.println("</body></html>");

        // Clean-up environment
        rs.close();
        stmt.close();
        conn.close();
    } catch(SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch(Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try {
            if(stmt!=null)
                stmt.close();
        } catch(SQLException se2) {
        } // nothing we can do
        try {
            if(conn!=null)
                conn.close();
        } catch(SQLException se) {
            se.printStackTrace();
        } //end finally try
    } //end try
}
}

```

Simple Example For Insertion



Create a Table in your Database

create table Student

```
(  
    name varchar(60),  
    email varchar(60),  
    pass varchar(100)  
)
```

index.html

```
<html>  
  <head>  
    <title>Register form</title>  
  </head>  
  <body>  
    <form method="post" action="register">  
      Name:<input type="text" name="name" /><br/>  
      Email ID:<input type="text" name="email" /><br/>  
      Password:<input type="text" name="pass" /><br/>  
      <input type="submit" value="register" />  
    </form>  
  </body>  
</html>
```

Register.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class Register extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String pass = request.getParameter("pass");
        try{

            //loading drivers for mysql
            Class.forName("com.mysql.jdbc.Driver");
            //creating connection with the database
            Connection con=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/test","username","password");

            PreparedStatement ps=con.prepareStatement
                ("insert into Student values(?,?,?)");

            ps.setString(1, name);
            ps.setString(2, email);
            ps.setString(3, pass);
            int i=ps.executeUpdate();

            if(i>0)
            {
                out.println("You are sucessfully registered");
            }
        }
        catch(Exception se)
        {
            se.printStackTrace();
        }
    }
}
```

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

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

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

- 1) Extension to Servlet
- 2) Easy to maintain
- 3) Fast Development: No need to recompile and redeploy
- 4) Less code than Servlet

JSP Scripting elements

Java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

`<% java source code %>`

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```


JSP expression tag

The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%= statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>

<body>

Current Time: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname"><br/>

<input type="submit" value="go">

</form>

</body>

</html>
```

File: welcome.jsp

```
<html>

<body>

<%= "Welcome "+request.getParameter("uname") %>

</body>

</html>
```

JSP Declaration Tag

- The JSP declaration tag is used to declare fields and methods.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
- So it doesn't get memory at each request.
- Syntax of JSP declaration tag
- The syntax of the declaration tag is as follows:

<%! field or method declaration %>

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare	The jsp declaration tag can declare variables as

variables not methods.	well as methods.
The declaration of scriptlet tag is placed inside the <code>_jspService()</code> method.	The declaration of jsp declaration tag is placed outside the <code>_jspService()</code> method.

Example of JSP declaration tag that declares field

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

index.jsp

```
<html>

<body>

<%! int data=50; %>

<%= "Value of the variable is:"+data %>

</body>

</html>
```

Example of JSP declaration tag that declares method

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

index.jsp

```
<html>

<body>

<%!

int cube(int n){

return n*n*n*; }

%>

<%= "Cube of 3 is:"+cube(3) %>

</body>
```

</html>

JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplifies the JSP.
2. **Code Reusability** We can use the JSTL tags in various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL Tags

There JSTL mainly provides 5 types of tags:

Tag Name	Description
<u>Core tags</u>	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is http://java.sun.com/jsp/jstl/core . The prefix of core tag is c .
<u>Function tags</u>	The functions tags provide support for string manipulation and string length. The url for the functions tags is http://java.sun.com/jsp/jstl/functions and prefix is fn .
<u>Formatting tags</u>	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is http://java.sun.com/jsp/jstl/fmt and prefix is fmt .
<u>XML tags</u>	The xml sql tags provide flow control, transformation etc. The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x .
<u>SQL tags</u>	The JSTL sql tags provide SQL support. The url for the sql tags is http://java.sun.com/jsp/jstl/sql and prefix is sql .

JSTL Core Tags

The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

JSTL Core Tags List

Tags	Description
<u>c:out</u>	It display the result of an expression, similar to the way <code><%=...%></code> tag work.
<u>c:import</u>	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
<u>c:set</u>	It sets the result of an expression under evaluation in a 'scope' variable.
<u>c:remove</u>	It is used for removing the specified scoped variable from a particular scope.
<u>c:catch</u>	It is used for Catches any Throwable exceptions that occurs in the body.
<u>c:if</u>	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
<u>c:choose</u> , <u>c:when</u> , <u>c:otherwise</u>	It is the simple conditional tag that includes its body content if the evaluated condition is true.
<u>c:forEach</u>	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
<u>c:forEachTokens</u>	It iterates over tokens which is separated by the supplied delimiters.
<u>c:param</u>	It adds a parameter in a containing 'import' tag's URL.
<u>c:redirect</u>	It redirects the browser to a new URL and supports the context-relative URLs.
<u>c:url</u>	It creates a URL with optional query parameters.

JSTL Function Tags

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is: `<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>`

JSTL Function Tags List

JSTL Functions	Description
<u>fn:contains()</u>	It is used to test if an input string containing the specified substring in a program.
<u>fn:containsIgnoreCase()</u>	It is used to test if an input string contains the specified substring as a case insensitive way.
<u>fn:endsWith()</u>	It is used to test if an input string ends with the specified suffix.
<u>fn:escapeXml()</u>	It escapes the characters that would be interpreted as XML markup.
<u>fn:indexOf()</u>	It returns an index within a string of first occurrence of a specified substring.
<u>fn:trim()</u>	It removes the blank spaces from both the ends of a string.
<u>fn:startsWith()</u>	It is used for checking whether the given string is started with a particular string value.
<u>fn:split()</u>	It splits the string into an array of substrings.
<u>fn:toLowerCase()</u>	It converts all the characters of a string to lower case.
<u>fn:toUpperCase()</u>	It converts all the characters of a string to upper case.
<u>fn:substring()</u>	It returns the subset of a string according to the given start and end position.
<u>fn:substringAfter()</u>	It returns the subset of string after a specific substring.
<u>fn:substringBefore()</u>	It returns the subset of string before a specific substring.
<u>fn:length()</u>	It returns the number of characters inside a string, or the number of items in a collection.
<u>fn:replace()</u>	It replaces all the occurrence of a string with another string

sequence.

JSTL Formatting tags

The formatting tags provide support for message formatting, number and date formatting etc. The url for the formatting tags is <http://java.sun.com/jsp/jstl/fmt> and prefix is **fmt**.

The JSTL formatting tags are used for internationalized web sites to display and format text, the time, the date and numbers. The syntax used for including JSTL formatting library in your JSP is: `<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>`

Formatting Tags	Descriptions
fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
fmt:formatNumber	It is used to format the numerical value with specific format or precision.
fmt:parseDate	It parses the string representation of a time and date.
fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
fmt:message	It display an internationalized message.
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.

Creating HTML forms by embedding JSP code.

Index.html

```
<html>
<title>calculator</title>
<head><h1><center>Basic Calculator</center></h1></head>
<body>
<center>
<form action="calculator.jsp" method="get">
<label for="num1"><b>Number 1</b></label>
<input type="text" name="num1"><br><br>
<label for="num2"><b>Number 2</b></label>
<input type="text" name="num2"><br><br>
<input type="radio" name="r1" value="Add">+
<input type="radio" name="r1" value="Sub">-<br>
<input type="radio" name="r1" value="mul">*
<input type="radio" name="r1" value="div">/<br><br>
<input type="submit" value="submit">
</center>
</body>
</html>
```

calculator.jsp

```
<html>
<title>calculator</title>
<head></head>
<body>
<% @page language="java"%>
<%
    int num1 = Integer.parseInt(request.getParameter("num1"));
    int num2 = Integer.parseInt(request.getParameter("num2"));

    String operation = request.getParameter("r1");

    if(operation.equals("Add")){
        int add=num1+num2;
        out.println("Addition is: "+add);
    }
    else if(operation.equals("Sub")){
        int sub=num1-num2;
        out.println("Substraction is: "+sub);
    }
}
```



```

    }
    else if(operation.equals("mul")){
        int mul=num1*num2;
        out.println("multiplication is: "+mul);
    }
    else if(operation.equals("div"))
    {
        int div = num1/num2;
        if(num1>=num2)
            out.println("division is: "+div);
        else
            out.println("The division cannot be performed");
    }
}
%>
</body>
</html>

```

Output

Basic Calculator

Number 1

Number 2

☒ +

☐ -

☐ *

☐ /

Addition is: 10

Part-A

1. Compare get request type and post request type?
2. Write a java Script to Great User based on time using IF-ELSE condition
3. Write the code segment to store current server time in session using Servlet API.
4. Where You Apply Date Object and method in java script? Give an example
5. List benefits of using JavaScript code in an HTML document.
6. Analyze on the statement."Each object of a class has its own instance of static member variable."
7. Define JavaScript statement with an example.
8. Can you list the different methods defined in document and window object of JavaScript.
9. Summarize briefly about the interaction between a webserver and a servlet.
10. Predict the need for client and server side scripting.
11. Create and write the code to return the full URL of a Document.
12. Show how is session tracking is achieved by the URL rewriting?
13. Compare get request type and post request type.
14. CompareServletcontext and Servletconfig.
15. Show the use of 'param' variable in JSP.
16. Define JSTL and List Core Actions of JSTL?
17. Analyze about java scriptlet
18. Give types of directive in JSP.
19. List any three advantages of java servlet over JSP
20. Rewrite the code segment to store current server time in session using Java Servlet API.

Part-B

1. Write a Java script Program to validate a form using Window, document, form and Regular Expression object list their property and methods
2. Create a Scientific Calculator using HTML and JSP
3. Consider a java script and HTML to create a page with two panes. One is left Pane Should have a text area where HTML code can be typed by the user .The pane on the right side should display the preview of the HTML code typed by the user,
4. A database has a table Employee with two columns Employee Id and Name. Write a JDBC program access the database that can query and print all entries in the table
5. Create a Java Script Function for i) Biggest of Three Number ii) Factorial using Recursion iii)Reverse A Number iv)sum of Digits
6. Create a JSP program to find simple interest and display the result in client.
7. Write Short notes on i) Scriptlets ii) Implicit JSP Objects. iii) Expression. iv) Declarations.
8. Create a java script to display the welcome message using the alert whenever button of a html form is pressed and Demonstrate on DHTML with JavaScript
9. Create a java servlet program for cookie Tracking for a request
10. Describe how to implement conditional statements and loops in JavaScript
11. Quote a brief note on Array declaration in JavaScript
12. Give various Operators in JavaScript.
13. Discuss the concepts of Event Handling