

Internet

The Internet, sometimes called simply "the Net," is a worldwide system of computer networks - a network of networks in which users at any one computer can, if they have permission, get information from any other computer (and sometimes talk directly to users at other computers). The Internet is a global [network](#) connecting millions of [computers](#). More than 190 countries are linked into exchanges of [data](#), news and opinions. It was conceived by the Advanced Research Projects Agency (ARPA) of the U.S. government in 1969 and was first known as the [ARPANet](#). The original aim was to create a network that would allow users of a research computer at one university to "talk to" research computers at other universities. A side benefit of ARPANet's design was that, because messages could be routed or rerouted in more than one direction, the network could continue to function even if parts of it were destroyed in the event of a military attack or other disaster.

Today, the Internet is a public, cooperative and self-sustaining facility accessible to hundreds of millions of people worldwide. Physically, the Internet uses a portion of the total resources of the currently existing public telecommunication networks. Technically, what distinguishes the Internet is its use of a set of protocols called [TCP/IP](#) (for Transmission Control Protocol/Internet Protocol). Two recent adaptations of Internet technology, the [intranet](#) and the [extranet](#), also make use of the TCP/IP protocol.

For most Internet users, electronic mail ([email](#)) practically replaced the postal service for short written transactions. People communicate over the Internet in a number of other ways including Internet Relay Chat ([IRC](#)), [Internet telephony](#), [instant messaging](#), video chat or [social media](#).

The most widely used part of the Internet is the [World Wide Web](#) (often abbreviated "WWW" or called "the Web"). Its outstanding feature is [hypertext](#), a method of instant cross-referencing. In most Web sites, certain words or phrases appear in text of a different color than the rest; often this text is also underlined. When you select one of these words or phrases, you will be transferred to the site or page that is relevant to this word or phrase. Sometimes there are buttons, images, or portions of images that are "clickable." If you move the pointer over a spot on a Web site and the pointer changes into a hand, this indicates that you can click and be transferred to another site.

Using the Web, you have access to billions of pages of information. Web browsing is done with a Web [browser](#), the most popular of which are [Chrome](#), [Firefox](#) and [Internet Explorer](#). The appearance of a particular Web site may vary slightly depending on the browser you use. Also, later versions of a particular browser are able to render more "bells and whistles" such as animation, [virtual reality](#), sound, and music files, than earlier versions.

The Internet has continued to grow and evolve over the years of its existence. [IPv6](#), for example, was designed to anticipate enormous future expansion in the number of available [IP addresses](#). In a related development, the Internet of Things ([IoT](#)) is the burgeoning environment in which almost any entity or object can be provided with a [unique identifier](#) and the ability to transfer data automatically over the Internet.

Is Web and Internet the Same?

The *Internet* is **not** synonymous with *World Wide Web*. The Internet is a massive network of networks, a networking infrastructure. It connects millions of computers together globally, forming a network in which any computer can communicate with any other computer as long as they are both connected to the Internet. The World Wide Web, or simply Web, is a way of accessing information over the medium of the Internet. It is an information-sharing model that is built on top of the Internet. In conclusion Internet is infrastructure and web is one of the service on that infrastructure.

So who actually owns the Internet?

There are two answers to this question:

1. Nobody
2. Lots of people

If you think of the Internet as a unified, single entity, then no one owns it. There are organizations that determine the Internet's structure and how it works, but they don't have any ownership over the Internet itself. No government can lay claim to owning the Internet, nor can any company. The Internet is like the telephone system -- no one owns the whole thing.

From another point of view, thousands of people and organizations own the Internet. The Internet consists of lots of different bits and pieces, each of which has an owner. Some of these owners can control the quality and level of access you have to the Internet. They might not own the entire system, but they can impact your Internet experience.

Internet Browser or Browser

Alternatively referred to as a web browser, a browser is a software program created as a simplified means to present and explore content on the World Wide Web. These pieces of content, including pictures, videos, and web pages, are connected using hyperlinks and classified with Uniform Resource Identifiers (URLs).

There have been many different web browsers that have come and gone over the years; the first, named WorldWideWeb (later changed to Nexus) was invented by Tim Berners-Lee in 1990. Examples of some current Internet browsers

- Google Chrome
- Microsoft Edge
- Microsoft Internet Explorer
- Mozilla Firefox
- Opera
- Apple Safari

Getting around in a browser

Each browser has a navigation toolbar that helps you find your way around the Internet. As seen in the images below, the navigation toolbar has undergone significant changes to streamline its appearance and functionality. However, is likely never going to lose the navigation arrows and address bar.

Microsoft Internet Explorer 3.02



Microsoft Internet Explorer 7



Microsoft Internet Explorer 9



Microsoft Internet Explorer 11






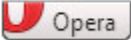
Overview of browser bar buttons, menus, and functions

As we mentioned in the previous section, over time, many Internet browser buttons and options have either been moved or done away with completely. Consequently, some of the options mentioned below may not be immediately visible on your browser.

Tip: In certain browsers, pressing the Alt key on your keyboard shows hidden options.

Settings (Menu)

Nearly all modern browsers today have moved advanced options and features in the upper right or left-hand corner of the browser window. Each browser's menu button is different:

For example, Internet Explorer uses , Chrome uses , Firefox uses , and Opera uses .

Back

The back button visits the previous page that referred you to the page you are currently viewing. Typically this button resembles an arrow pointing to the left.

Forward

The forward button moves you forward a page. It only works if you have previously used the back button. If you have not gone back and your browser shows a forward button, it will be grayed out.

Stop

The stop button no longer exists in the majority of modern web browsers. However, its function (to stop a web page from loading) may still be executed by pressing the Esc key.

Refresh (Reload)

As you browse the Internet the browser cache's data it downloads, meaning they store some or all parts of each page you visit on your computer. This feature is useful as it allows users to not have to download the full page each time they visit the same site. On some sites, you may want to refresh the page to get the latest version; e.g. on a news site. The Refresh button can also be used to reload a page that has failed to load because of an error.

Tip: Press the F5 key or Ctrl + R to refresh the page from the keyboard.

Home

The Home button is used to return users to their default web page; the same page that loads when the browser is first opened.

Search

In the past, the Search button was used to open a user's default search page or execute a search on the text found in the address or URL text field. Today's browsers have what is called an Omnibox, which is a search function built into the address bar.

Full Screen

This function is used to make the browser window a full screen; temporarily removing the toolbar, buttons, and address bar. Often, this view mode may be toggled on and off by pressing the F11 key on your keyboard.

History

This feature allows users to view which pages that have been visited since the browser history was last cleared or created. All your saved pages are stored in your Internet cache.

Tip: The shortcut key for history for most browsers is Ctrl+H.

Favorites (Bookmarks)

This folder stores websites or pages chosen by the user. The term "Favorites" is used with Microsoft Internet Explorer; in other browsers this may be known as bookmarks or a hotlist.

Print

This feature, although no longer a button and may be accessed through the main settings menu or pressing Ctrl + P on the keyboard.

Font (Size)

This button no longer exists but was used to increase or decrease the size of a font; an option now covered by the zoom feature.

Zoom

On modern browsers, holding down the Ctrl key and pressing either the + or - key zooms in or out, increasing and decreasing the size of font and images. To reset the zoom function to its default size, press Ctrl + 0 (zero) at the same time.

Mail

Used to open a user's preferred e-mail program. Today, this option is no longer found in browsers.

Edit

Used to open and edit the web page you are currently viewing in an HTML editor. Today, no longer found in browsers.

What is website?

A site (location) on the World Wide Web. A website, also written as web site, or simply site, is a set of related web pages typically served from a single web domain. Each Web site contains a home page, which is the first document users see when they enter the site. The site might also contain additional documents and files. Each site is owned and managed by an individual, company or organization. A website is hosted on at least one web server, accessible via a network such as the Internet or a private local area network through an Internet address known as a uniform resource locator (URL). All publicly accessible websites collectively constitute the World Wide Web.

Web pages, which are the building blocks of websites, are documents, typically written in plain text interspersed with formatting instructions of Hypertext Markup Language (HTML, XHTML). They may incorporate elements from other websites with suitable markup anchors. Web pages are accessed and transported with the Hypertext Transfer Protocol (HTTP), which may optionally employ encryption (HTTP Secure, HTTPS) to provide security and privacy for the user of the webpage content. The user's application, often a web browser, renders the page content according to its HTML markup instructions onto a display terminal.

The pages of a website can usually be accessed from a simple Uniform Resource Locator (URL) called the web address. The URLs of the pages organize them into a hierarchy, although hyper-linking between them conveys the reader's perceived site structure and guides the reader's navigation of the site which generally includes a home page with most of the links to the site's web content, and a supplementary about, contact and link page.

What is web server?

Definition: A web server is a computer that runs websites. It's a computer program that distributes web pages as they are request. **The basic objective of the web server is to store, process and deliver web pages to the users.** This intercommunication is done using Hypertext Transfer Protocol (HTTP).

What is localhost?

In computer networking, localhost is a hostname that means this computer or this host. It may be used to access the network services that are running on the host via its loopback network interface. Using the loopback interface bypasses and does not require any local network interface hardware. The local loopback mechanism may be useful for testing software during development, independently of any networking configurations. For example, if a computer has been configured to provide a website, directing a locally running web browser to `http://localhost` may display its home page. On most computer systems, localhost resolves to the IP address 127.0.0.1

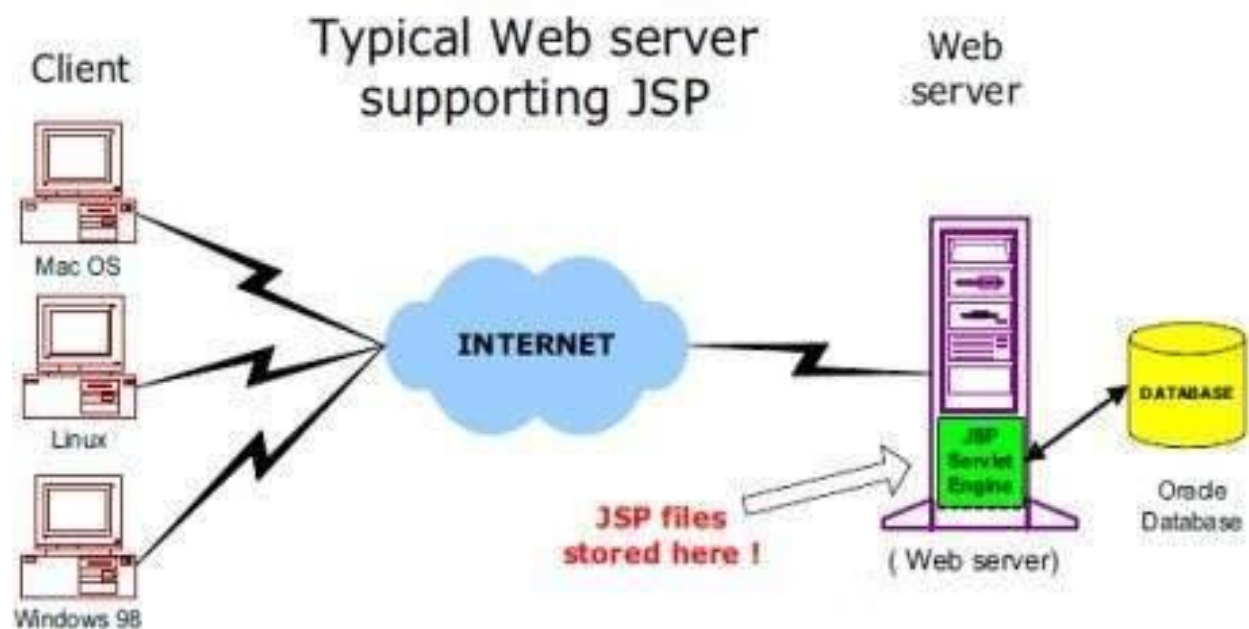
What is Java Server Pages?

- **Java Server Pages (JSP)** is a technology for developing **web pages that support dynamic content**.
- Java Server Pages (JSP) is a server side programming technology that enables creation of dynamic, platform independent method for building web based application.
- JSP helps in building web pages that supports dynamic content.
- JSP helps developers to **insert java code** in HTML pages by making use of special **JSP tags**, most of which start with **<%** and end with **%>**.
- A Java Server Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. **Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.**
- Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, passing control between pages and sharing information between requests, pages etc.

The web server needs a JSP engine i.e. container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. Apache has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web Application.

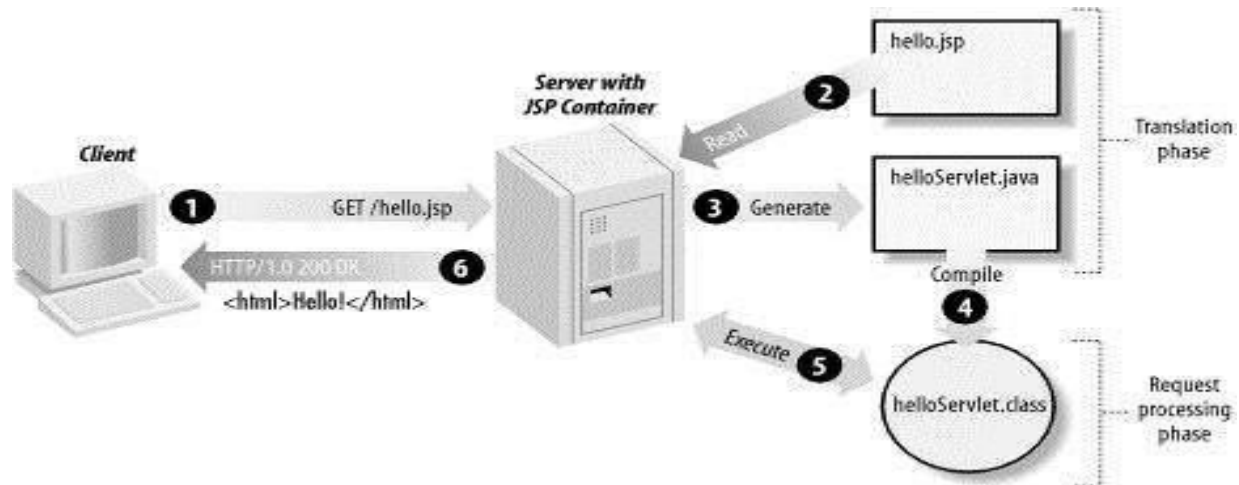


JSP Processing:

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to `println()` statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be shown below in the following diagram:



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

Scripting elements in JSP:

The scripting element provides the ability to insert java code inside JSP.

There are three types of scripting elements:

1. **Scriptlet tag**
2. **Declaration tag**
3. **Expression tag**

1. The Scriptlet tag:

- In JSP, java code can be written inside the JSP page using the scriptlet tag.
- A scriptlet tag is used to execute java source code in JSP.
- A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Syntax:

```
<% java source code %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:scriptlet>
```

```
java source code
```

```
</jsp:scriptlet>
```

NOTE: Any text and HTML tags you must write outside the scriptlet.

Example: Hello.jsp

```
<html>
<head><title>Hello World</title></head>
<body>
Hello World!<br/>

<%

out.println("Your IP address is " + request.getRemoteAddr());

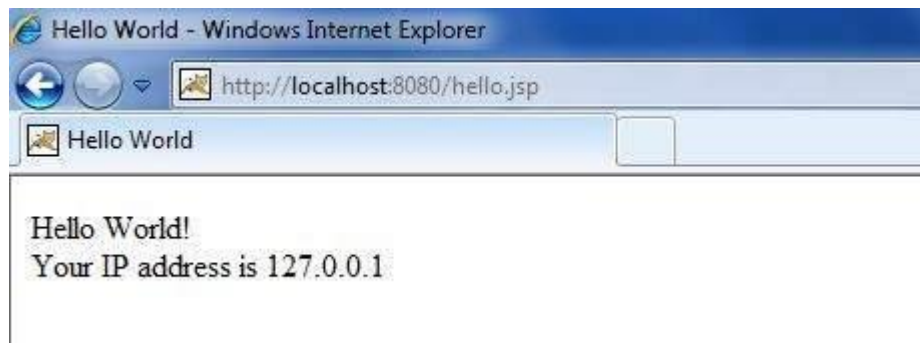
%>

</body>
</html>
```

NOTE: Assuming that Apache Tomcat is installed in C:\apache-tomcat-7.0.2 and your environment is setup.

Put **Hello.jsp** file in C:\apache-tomcat-7.0.2\webapps\ROOT directory and try to browse it by giving URL <http://localhost:8080/Hello.jsp>.

This would generate following result:



2. Declarations tag:

A declaration tag declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax of JSP Declarations:

```
<%! ... .. %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:declaration>  
    code fragment  
</jsp:declaration>
```

Following is the simple example for JSP Declarations:

```
<%! int i = 0; %>  
<%! int a, b, c; %>  
  
<%!  
int addNum(int x,int y)  
{  
    return (x+y);  
}  
%>
```

Execute:

```
<%  
out.print("sum="+addNum(5,3);  
%>
```

3. Expression tag:

Expression tag is mainly used to print the value of variable or method.

We can use an expression tag within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Syntax of JSP Expression:

```
<%= expression %>
```

You can write XML equivalent of the above syntax as follows:

```
<jsp:expression>  
    expression  
</jsp:expression>
```

Following is the simple example for JSP Expression:

```
<html>
<head><title>A Comment Test</title></head>
<body>

<%! int addNum(int x,int y)
{
    return (x+y);
}
%>

<%! int a=10;%>

<p>
The value of a = <%= a %>
</p>

<p>
    <%= addNume(10,20) %>
</p>

</body>
</html>
```

JSP Comments:

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

Following is the syntax of JSP comments:

<%-- This is JSP comment --%>

Following is the simple example for JSP Comments:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<h2>A Test of Comments</h2>

<%-- This comment will not be visible in the page source --%>

</body>
</html>
```

A Test of Comments

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary:

Syntax	Purpose
<%-- comment --%>	A JSP comment. Ignored by the JSP engine.
<!-- comment -->	An HTML comment. Ignored by the browser.
<\<%>	Represents static <% literal.
%\>	Represents static %> literal.
\'	A single quote in an attribute that uses single quotes.
\"	A double quote in an attribute that uses double quotes.

JSP Directives:

Directives are the message that tells the web container how to translate a JSP page into the corresponding servlet.

It usually has the following form:

<%@ directive attribute="value" %>

There are three types of directive tag:

Directive	Description
1. <%@ page ... %>	<p>Defines page-dependent attributes, such as import , contentType, scripting language, pageEncoding etc.</p> <p>Eg:</p> <div><div><%@page import="java.sql.*"%></div><div><% @page contentType="text/html" pageEncoding="UTF-8" %></div><div><%@page import="my.query.*" %></div></div>
2. <%@ include ... %>	<p>Includes directive is used to include the contents of any resource. It may be the JSP file, HTML file or text file.</p> <p>Eg:</p> <div><%@ include file="header.html" %></div>
3. <%@ taglib ... %>	<p>Declares a tag library, containing custom actions, used in the page</p>

JSP Actions tags:

There are many JSP action tags or elements that are used to perform some specific tasks.

You can dynamically insert a file, forward the user to another page.

<jsp:action_name attribute="value" />

Eg:

```
<jsp:forward page="employee.jsp"/>
```

Action elements are basically predefined functions.

There are following JSP actions available in jsp:

Syntax	Purpose
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean
jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

Eg: Home.jsp

```
<html>

<body>

<jsp:forward page="Employee.jsp" />

</body>

</html>
```

Employee.jsp

```
<html>

<body>

<%

out.println("Today is :"+ java.util.Calendar.getInstance().getTime() );

%>

</body>

</html>
```

JSP Implicit Objects:

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are available to all the JSP pages:

Objects	Description
request	This is the HttpServletRequest object associated with the request.
response	This is the HttpServletResponse object associated with the response to the client.
out	This is the PrintWriter object used to send output to the client.
session	This is the HttpSession object associated with the request.
application	This is the ServletContext object associated with application context.
config	This is the ServletConfig object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance JspWriters .
page	This is simply a synonym for this , and is used to call the methods defined by the translated servlet class.
Exception	The Exception object allows the exception data to be accessed by designated JSP.

Control-Flow Statements:

JSP provides full power of Java to be embedded in your web application. You can use all the APIs and building blocks of Java in your JSP programming including decision making statements, loops etc.

Decision-Making Statements:

The **if...else** block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

```
<%! int day = 3; %>
<html>
<head><title>IF...ELSE Example</title></head>
<body>
<% if (day == 1 | day == 7) { %>
    <p> Today is weekend</p>
<% } else { %>
    <p> Today is not weekend</p>
<% } %>
</body>
</html>
```

This would produce following result:

Today is not weekend

Now look at the following **switch...case** block which has been written a bit differently using **out.println()** and inside Scriptletas:

```
<%! int day = 3; %>
<html>
<head><title>SWITCH...CASE Example</title></head>
<body>
<%
switch(day) {
case 0:
    out.println("It's Sunday.");
    break;
case 1:
    out.println("It's Monday.");
    break;
case 2:
    out.println("It's Tuesday.");
    break;
case 3:
    out.println("It's Wednesday.");
    break;
case 4:
```

```

        out.println("It's Thursday.");
        break;
case 5:
    out.println("It's Friday.");
    break;
default:
    out.println("It's Saturday.");
}
%>
</body>
</html>

```

This would produce following result:

It's Wednesday.

Loop Statements:

You can also use three basic types of looping blocks in Java: **for**, **while**, and **do...while** blocks in your JSP programming.

Let us look at the following **for** loop example:

```

<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body>
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
    <font color="green" size="<%= fontSize %>">
        JSP Tutorial
    </font><br />
<% } %>
</body>
</html>

```

This would produce following result:

JSP Tutorial

JSP Tutorial

JSP Tutorial

Above example can be written using **while** loop as follows:

```
<%! int fontSize; %>
<html>
<head><title>WHILE LOOP Example</title></head>
<body>
<%while ( fontSize <= 3){ %>
    <font color="green" size="<%= fontSize %>">
        JSP Tutorial
    </font><br />
<%fontSize++;%>
<% } %>
</body>
</html>
```

This would also produce following result:

JSP Tutorial

JSP Tutorial

JSP Tutorial

JSP Operators:

JSP supports all the logical and arithmetic operators supported by Java. Following table give a list of all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ -- ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right

Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

JSP Literals:

The JSP expression language defines the following literals:

- **Boolean:** true and false
- **Integer:** as in Java
- **Floating point:** as in Java
- **String:** with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\.
- **Null:** null

hello1.jsp

```
<html>

<head><title>Hello World</title></head>

<body>

Hello World!<br/>

<%

out.println("Your JSP program");

%>

</body>

</html>
```

hello11.jsp

```
<html>

<head><title>Hello World</title></head>

<body>

Hello World!<br/>

<jsp:scriptlet>

out.println("Your JSP program");

</jsp:scriptlet>

</body>

</html>
```


hello2.jsp

```
<% ! int day = 3; %>

<html>

<head><title>SWITCH...CASE Example</title></head>

<body>

<%

switch(day) {

case 0:

    out.println("It's Sunday.");

    break;

case 1:

    out.println("It's Monday.");

    break;

case 2:

    out.println("It's Tuesday.");

    break;

case 3:

    out.println("It's Wednesday.");

    break;

case 4:

    out.println("It's Thursday.");
```

```
        break;

    case 5:

        out.println("It's Friday.");

        break;

    default:

        out.println("It's Saturday.");

    }

    %>
```

```
</body>
```

```
</html>
```

hello22.jsp

```
<jsp:declaration> int day = 3; </jsp:declaration>
```

```
<html>
```

```
<head><title>SWITCH...CASE Example</title></head>
```

```
<body>
```

```
<jsp:scriptlet>
```

```
switch(day) {
```

```
case 0:
```

```
    out.println("It's Sunday.");
```

```
        break;
case 1:
    out.println("It's Monday.");
    break;
case 2:
    out.println("It's Tuesday.");
    break;
case 3:
    out.println("It's Wednesday.");
    break;
case 4:
    out.println("It's Thursday.");
    break;
case 5:
    out.println("It's Friday.");
    break;
default:
    out.println("It's Saturday.");
}
</jsp:scriptlet>

</body>

</html>
```

hello3.jsp

```
<html>
```

```
<head><title>A Comment Test</title></head>
```

```
<body>
```

```
<p>
```

```
Today's date: <%= (new java.util.Date()).toString()%>
```

```
<br/>
```

```
Today's date: <% out.println((new java.util.Date()).toString());%>
```

```
</p>
```

```
</body>
```

```
</html>
```

hello33.jsp

```
<html>
```

```
<head><title>A Comment Test</title></head>
```

```
<body>
```

```
<p>
```

```
Today's date: <jsp:expression>
```

```
(new java.util.Date()).toString()
```

```
</jsp:expression>
```


Today's date: <jsp:scriptlet> out.println((new java.util.Date()).toString());</jsp:scriptlet>

</p>

</body>

</html>

hello4.jsp

<html>

<head><title>A Comment Test</title></head>

<body>

<p>

Today's date: <jsp:expression>

(new java.util.Date()).toString()

</jsp:expression>

<%--

Today's date: out.println((new java.util.Date()).toString());

this is JSP comment i.e.

ignored by JSP engine and browser(even not visible in html source)

--%>

</p>

</body>

</html>

hello44.jsp

<html>

<head><title>A Comment Test</title></head>

<body>

<p>

Today's date: <jsp:expression>

(new java.util.Date()).toString()

</jsp:expression>

<!--

Today's date: out.println((new java.util.Date()).toString());

This is HTML comment i.e.

ignored by JSP engine and browser but visible in html source

--!>

```
</p>
</body>
</html>
```

hello5.jsp

```
<% ! int i; %>

<html>
<head><title>FOR LOOP Example</title></head>
<body>

<%for ( i = 1; i <= 10; i++){

    out.println("loop: "+i);

}%>

</body>
</html>
```

hello55.jsp

```
<% ! int i; %>

<html>
<head><title>FOR LOOP Example</title></head>
```

```
<body>
```

```
<%for ( i = 1; i <= 10; i++){
```

```
    out.println("loop: "+i);
```

```
%>
```

```
<br/>
```

```
<% } %>
```

```
</body>
```

```
</html>
```

hello6.jsp

```
<% ! int fontSize; %>
```

```
<html>
```

```
<head><title>FOR LOOP Example</title></head>
```

```
<body>
```

```
<%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
```

```
<font color="green" size="5">
```

JSP Tutorial


```
</font> <br/>
```

```
<% } %>
```

```
</body>
```

```
</html>
```

hello66.jsp

```
<% ! int fontSize; %>
```

```
<html>
```

```
<head><title>FOR LOOP Example</title></head>
```

```
<body>
```

```
<% for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
```

```
<font color="green" size="<%= fontSize %>">
```

JSP Tutorial

```
</font> <br/>
```

```
<% } %>
```

```
</body>
```

```
</html>
```

hello7.jsp

```
<html>

<body>

<form name="frm" method="post" action="hello77.jsp">

<!-- using post method --!>

<table width="100%" border="0" cellspacing="0" cellpadding="0">

  <tr>

    <td width="22%">&nbsp;</td>

    <td width="78%">&nbsp;</td>

  </tr>

  <tr>

    <td>Name Student </td>

    <td><input type="text" name="studentName"></td>

  </tr>

  <tr>

    <td>&nbsp;</td>

    <td><input type="submit" name="b1" value="Enter"></td>

  </tr>

  <tr>

    <td>&nbsp;</td>

    <td>&nbsp;</td>

  </tr>

</table>

<%
```

```
String studentNameInJSP=request.getParameter("studentName");  
%>
```

Value of input text box in JSP : <%=studentNameInJSP%>

```
</form>
```

```
</body>
```

```
</html>
```

hello77.jsp

```
<html>
```

```
<body>
```

```
<form name="frm" method="post" action="hello77.jsp">
```

```
<!-- using post method --!>
```

```
<table width="100%" border="0" cellspacing="0" cellpadding="0">
```

```
<tr>
```

```
<td width="22%">&nbsp;</td>
```

```
<td width="78%">&nbsp;</td>
```

```
</tr>
```

```
<tr>
```

```
<td>Name Student </td>
```

```
<td><input type="text" name="studentName"></td>
```

```
</tr>
```

```
<tr>
```

```
<td>&nbsp;</td>
```

```
<td><input type="submit" name="b1" value="Enter"></td>
```

```

        </tr>

        <tr>

            <td>&nbsp;</td>

            <td>&nbsp;</td>

        </tr>

    </table>

    <%
String studentNameInJSP=request.getParameter("studentName");
%>

Value of input text box in JSP : <%=studentNameInJSP%>

```

```

</form>

</body>

</html>

```

hello8.jsp

```

<html>

<body>

<form name="frm" method="post" action="hello8.jsp">

<table width="100%" border="0" cellspacing="0" cellpadding="0">

    <tr>

        <td width="22%">&nbsp;</td>

        <td width="78%">&nbsp;</td>

    </tr>


```

```

<tr>

    <td>First value </td>

    <td><input type="text" name="v1"></td>

</tr>

<tr>

    <td>Second value </td>

    <td><input type="text" name="v2"></td>

</tr>

<tr>

    <td>&nbsp;</td>

    <td><input type="submit" name="b1" value="ADD"></td>

</tr>

<tr>

    <td>&nbsp;</td>

    <td>&nbsp;</td>

</tr>
</table>

<%
try{
int x=Integer.parseInt(request.getParameter("v1"));
int y=Integer.parseInt(request.getParameter("v2"));
int z=x+y;
out.println("Addition of numbers is "+ z);

```

```

    }
    catch(Exception e)
    {
        out.println("You entered invalid number");
    }
    %>

```

```

</form>

```

```

</body>

```

```

</html>

```

hello88.jsp

```

<html>

```

```

<body>

```

```

<form name="frm" method="post" action="hello88.jsp">

```

```

<table width="100%" border="0" cellspacing="0" cellpadding="0">

```

```

    <tr>

```

```

        <td width="22%">&nbsp;</td>

```

```

        <td width="78%">&nbsp;</td>

```

```

    </tr>

```

```

    <tr>

```

```

        <td>First value </td>

```

```

        <td><input type="text" name="v1"></td>

```

```

    </tr>

```

```

</tr>

```

```

        <td>Second value </td>

        <td><input type="text" name="v2"></td>

    </tr>

    <tr>

        <td>&nbsp;</td>

        <td><input type="submit" name="b1" value="ADD"></td>

    </tr>

    <tr>

        <td>&nbsp;</td>

        <td>&nbsp;</td>

    </tr>

</table>

<%
try{
int x=Integer.parseInt(request.getParameter("v1"));
int y=Integer.parseInt(request.getParameter("v2"));
int z=x+y;
out.println("Addition of numbers is "+ z);
}%>

<input type="text" name="v3" value="<%=z%>">

```

```

<% }
catch(Exception e)
{
out.println("You entered invalid number");
}
%>

```

```

</form>

```

```

</body>

```

```

</html>

```

hello9.jsp

```

<html>

```

```

<body>

```

```

<form name="frm" method="get" action="hello99.jsp">

```

```

<table width="100%" border="0" cellspacing="0" cellpadding="0">

```

```

<tr>

```

```

    <td width="22%">&nbsp;  </td>

```

```

    <td width="78%">&nbsp;  </td>

```

```

</tr>

```

```

<tr>

```

```

    <td>Active <input type="radio" name="active" value="Active"></td>

```

```

    <td>DeActive <input type="radio" name="active" value="DeActive"></td>

```

```

</tr>

```

```

<tr>

```

```

    <td>&nbsp;  </td>

```



```

        <td><input type="submit" name="submit" value="Submit"></td>

    </tr>

    <tr>

        <td>&nbsp;</td>

        <td>&nbsp;</td>

    </tr>

</table>

</form>

</body>

</html>

```

hello99.jsp

```

<%
String radioInJSP=request.getParameter("active");
%>

<html>

<body>

Value of radio button box in JSP : <%=radioInJSP%>

</body>

</html>

```

What are 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.

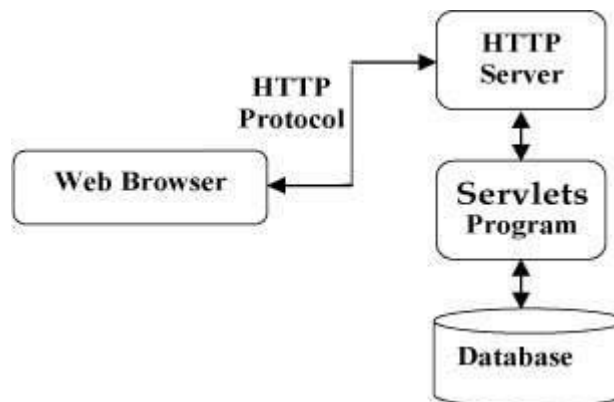
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet.

Servlets Architecture:

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



Note:

Like any other Java program, you need to compile a servlet by using the Java compiler **javac** and after compilation the servlet application, it would be deployed in a configured environment to test and run.

Set classpath variable with value as follows

C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar;

Servlets - Examples

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend **javax.servlet.http.HttpServlet**, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code for helloServlet example:

Following is the sample source code structure of a servlet example to write Hello World:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class helloServlet extends HttpServlet {

    private String message;

    public void init() throws ServletException
    {
        // Do required initialization
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }
}
```

```

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

```

Compiling a Servlet:

Assuming your environment is setup properly, go in directory that contain helloServlet.java file and compile helloServlet.java as follows:

```
> javac helloServlet.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. I have included only servlet-api.jar JAR file because I'm not using any other library in this program.

i.e.

classpath

C:\Program Files\Apache Software Foundation\Tomcat 7.0\lib\servlet-api.jar;

Servlet Deployment:

copy helloServlet.class into <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes/helloServlet.class

and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/web.xml

```

<?xml version="1.0"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>HelloWorld</servlet-name>
        <servlet-class>helloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloWorld</servlet-name>
        <url-pattern>/HelloWorld</url-pattern>
    </servlet-mapping>
</web-app>

```

Above entries to be created inside `<web-app>...</web-app>` tags available in `web.xml` file. There could be various entries in this table already available, but never mind.

Finally type **`http://localhost:8080/HelloWorld`** in browser's address box. If everything goes fine, you would get following result:



Servlets - Life Cycle

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

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

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
public void init() throws ServletException {  
    // Initialization code...  
}
```

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
public void service(ServletRequest request,  
    ServletResponse response)
```

```
    throws ServletException, IOException {  
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

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

The doGet() Method

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

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

The doPost() Method

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

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

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

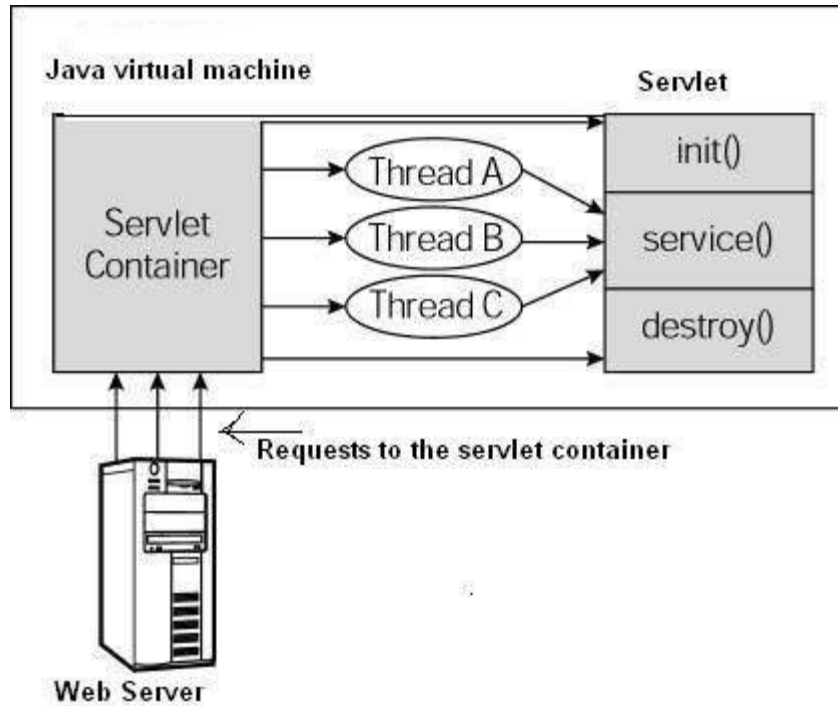
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {  
    // Finalization code...  
}
```

Architecture Diagram:

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Difference between Servlet and JSP

| JSP | Servlets |
|--|---|
| JSP is a <u>webpage scripting language</u> that can generate dynamic content. | Servlets are <u>Java programs</u> that are already compiled which also creates dynamic web content. |
| JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets. | Servlets run faster compared to JSP. |
| It's easier to code in JSP than in Java Servlets. | Its little much code to write here. |
| In MVC, jsp act as a view. | In MVC, servlet act as a controller. |
| JSP are generally preferred when there is not much processing of data required. | servlets are best for use when there is more processing and manipulation involved. |
| The advantage of JSP programming over servlets is that we can build <u>custom tags</u> which can directly call <u>Java beans</u> . | There is no such custom tag facility in servlets. |
| We can achieve functionality of JSP at client side by running <u>JavaScript</u> at client side. | There are no such methods for servlets. |

Servlets - Form Data

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

GET method:

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

`http://www.test.com/hello?key1=value1&key2=value2`

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password, date of birth or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string.

This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

POST method:

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

Reading Form Data using Servlet:

Servlets handles form data parsing automatically using the following methods depending on the situation:

- **getParameter():** You call `request.getParameter()` method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

GET Method Example Using URL:

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name=ZARA&last_name=ALI

Below is **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information:

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String title = "Using GET Method to Read Form Data";
        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" +
            "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
            "</body></html>");
    }
}
```

Assuming your environment is setup properly, compile HelloForm.java as follows:

```
> javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class** file. Next you would have to copy this class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```

<servlet>
  <servlet-name>HelloForm</servlet-name>
  <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloForm</servlet-name>
  <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>

```

Now type *http://localhost:8080/HelloForm?first_name=Ankit &last_name=Poddar* in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result:

GET Method Example Using Form:

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```

<html>
<body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>

```

Keep this HTML in a file Hello.htm and put it in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.

First Name: Last Name:

Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

POST Method Example Using Form:

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is **HelloForm.java** servlet program to handle input given by web browser using GET or POST methods.

```

// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String title = "Using GET Method to Read Form Data";
        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" +
            "<ul>\n" +
            "  <li><b>First Name</b>: "
            + request.getParameter("first_name") + "\n" +
            "  <li><b>Last Name</b>: "
            + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
            "</body></html>");
    }

    // Method to handle POST method request.
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}

```

Now compile, deploy the above Servlet and test it using Hello.htm with the POST method as follows:

```

<html>
<body>
<form action="HelloForm" method="POST">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>

```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

First Name: Last Name:

Based on the input provided, it would generate similar result as mentioned in the above examples.

Passing Checkbox Data to Servlet Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
<body>
<form action="CheckBox" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" />
    Chemistry
<input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

The result of this code is the following form

☒ Maths ☐ Physics ☒ Chemistry

Below is CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String title = "Reading Checkbox Data";
        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" +
        "<ul>\n" +
        "  <li><b>Maths Flag : </b>: "
        + request.getParameter("maths") + "\n" +
        "  <li><b>Physics Flag: </b>: "
        + request.getParameter("physics") + "\n" +
        "  <li><b>Chemistry Flag: </b>: "
        + request.getParameter("chemistry") + "\n" +
        "</ul>\n" +
        "</body></html>");
    }
    // Method to handle POST method request.
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}

```

For the above example, it would display following result:

Reading Checkbox Data

- **Maths Flag : : on**
- **Physics Flag: : null**
- **Chemistry Flag: : on**

Reading All Form Parameters:

Following is the generic example which uses **getParameterNames()** method of **HttpServletRequest** to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order.

Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

    // Method to handle GET method request.
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set response content type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();
        String title = "Reading All Form Parameters";
        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" +
            "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
            "<tr bgcolor=\"#949494\">\n" +
            "<th>Param Name</th><th>Param Value(s)</th>\n" +
            "</tr>\n");

        Enumeration paramNames = request.getParameterNames();

        while(paramNames.hasMoreElements()) {
            String paramName = (String)paramNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n<td>");
            String[] paramValues =
                request.getParameterValues(paramName);
            // Read single valued data
            if (paramValues.length == 1) {
                String paramValue = paramValues[0];
                if (paramValue.length() == 0)
                    out.println("<i>No Value</i>");
                else
                    out.println(paramValue);
            } else {
                // Read multiple valued data
                out.println("<ul>");
                for(int i=0; i < paramValues.length; i++) {
                    out.println("<li>" + paramValues[i]);
```

```

        }
        out.println("</ul>");
    }
}
out.println("</tr>\n</table>\n</body></html>");
}
// Method to handle POST method request.
public void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
}
}
}

```

Now, try above servlet with the following form:

```

<html>
<body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form>
</body>
</html>

```

Now calling servlet using above form would generate following result:

Reading All Form Parameters

| Param Name | Param Value(s) |
|------------|----------------|
| maths | on |
| chemistry | on |

You can try above servlet to read any other form's data which is having other objects like text box, radio button or drop down box etc.

States of website

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server:

Cookies:

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way because many time browser does not support a cookie, so I would not recommend to use this procedure to maintain the sessions.

Hidden Form Fields:

A web server can send a hidden HTML form field along with a unique session ID as follows:

```
<input type="hidden" name="sessionid" value="12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting:

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.

For example, with `http://tutorialspoint.com/file.htm;sessionid=12345`, the session identifier is attached as `sessionid=12345` which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies but here drawback is that you would have generate every URL dynamically to assign a session ID though page is simple static HTML page.

The session Object:

Apart from the above mentioned three ways, JSP makes use of servlet provided HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows:

```
<%@ page session="false" %>
```

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or getSession().

Final Example:

Session.jsp

```
<%@ page import="java.io.*,java.util.*" %>
```

```
<%
```

```
String title = "Welcome Back to my website";
```

```
Integer visitCount = new Integer(0);
```

```
String visitCountKey = new String("visitCount");
```

```
String userID = new String("ABCD");
```

```
String userIDKey = new String("userID");
```

```
// Check if this is new comer on your web page.
```

```
if (session.isNew()){
```

```

        title = "Welcome to my website";

        session.setAttribute(userIDKey, userID);

        session.setAttribute(visitCountKey, visitCount);
    }

    visitCount = (Integer)session.getAttribute(visitCountKey);

    visitCount = visitCount + 1;

    userID = (String)session.getAttribute(userIDKey);

    session.setAttribute(visitCountKey, visitCount);
%>

<html>

<head>

<title>Session Tracking</title>

</head>

<body>

<center>

<h1>Session Tracking</h1>

</center>

<table border="1" align="center">

<tr bgcolor="#949494">

    <th>Session info</th>

    <th>Value</th>

</tr>

```

```

<tr>

    <td>id</td>

    <td><% out.print(session.getId()); %></td>

</tr>

<tr>

    <td>User ID</td>

    <td><% out.print(userID); %></td>

</tr>

<tr>

    <td>Number of visits</td>

    <td><% out.print(visitCount); %></td>

</tr>

</table>

</body>

</html>

```

Session Tracking Example:

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```

<% @ page import="java.io.*,java.util.*" %>
<%
    // Get session creation time.
    Date createTime = new Date(session.getCreationTime());
    // Get last access time of this web page.
    Date lastAccessTime = new Date(session.getLastAccessedTime());

    String title = "Welcome Back to my website";
    Integer visitCount = new Integer(0);

```

```

String visitCountKey = new String("visitCount");
String userIDKey = new String("userID");
String userID = new String("ABCD");

// Check if this is new comer on your web page.
if (session.isNew()){
    title = "Welcome to my website";
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey, visitCount);
}
visitCount = (Integer)session.getAttribute(visitCountKey);
visitCount = visitCount + 1;
userID = (String)session.getAttribute(userIDKey);
session.setAttribute(visitCountKey, visitCount);
%>
<html>
<head>
<title>Session Tracking</title>
</head>
<body>
<center>
<h1>Session Tracking</h1>
</center>
<table border="1" align="center">
<tr bgcolor="#949494">
    <th>Session info</th>
    <th>Value</th>
</tr>
<tr>
    <td>id</td>
    <td><% out.print( session.getId()); %></td>
</tr>
<tr>
    <td>Creation Time</td>
    <td><% out.print(createTime); %></td>
</tr>
<tr>
    <td>Time of Last Access</td>
    <td><% out.print(lastAccessTime); %></td>
</tr>
<tr>
    <td>User ID</td>
    <td><% out.print(userID); %></td>
</tr>
<tr>
    <td>Number of visits</td>
    <td><% out.print(visitCount); %></td>
</tr>
</table>
</body>
</html>

```

Now put above code in main.jsp and try to access <http://localhost:8080/main.jsp>.

Cookies Handling Example

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. JSP transparently supports HTTP cookies using underlying servlet technology.

There are three steps involved in identifying returning users:

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

Setting Cookies with JSP:

Setting cookies with JSP involves three steps:

(1) Creating a Cookie object: You call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

```
Cookie cookie = new Cookie("key","value");
```

Keep in mind, neither the name nor the value should contain white space or any of the following characters:

```
[ ] ( ) = , " / ? @ : ;
```

(2) Setting the maximum age: You use `setMaxAge` to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60*60*24);
```

(3) Sending the Cookie into the HTTP response headers: You use `response.addCookie` to add cookies in the HTTP response header as follows:

```
response.addCookie(cookie);
```

Final example:

Cookies.jsp

```
<%  
    // Create cookies for first and last names.  
    Cookie firstName = new Cookie("first_name",  
                                   request.getParameter("first_name"));  
    Cookie lastName = new Cookie("last_name",
```



```

        request.getParameter("last_name"));

    // Set expiry date after 24 Hrs for both the cookies.
    firstName.setMaxAge(60*60*24);
    // Set expiry date after 2 seconds for both the cookies.
    lastName.setMaxAge(2);

    // Add both the cookies in the response header.
    response.addCookie(firstName);
    response.addCookie(lastName);
%>
<html>
<body>
<form action="cookies.jsp" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>

<center>
<h1>Setting Cookies</h1>
</center>
<ul>
<li><p><b>First Name:</b>
    <%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last Name:</b>
    <%= request.getParameter("last_name")%>
</p></li>
</ul>

</body>
</html>

```

Example:

Let us modify our [Form Example](#) to set the cookies for first and last name.

```

<%
    // Create cookies for first and last names.
    Cookie firstName = new Cookie("first_name",
        request.getParameter("first_name"));
    Cookie lastName = new Cookie("last_name",
        request.getParameter("last_name"));

    // Set expiry date after 24 Hrs for both the cookies.
    firstName.setMaxAge(60*60*24);
    lastName.setMaxAge(60*60*24);

    // Add both the cookies in the response header.
    response.addCookie( firstName );

```

```

    response.addCookie( lastName );
%>
<html>
<head>
<title>Setting Cookies</title>
</head>
<body>
<center>
<h1>Setting Cookies</h1>
</center>
<ul>
<li><p><b>First Name:</b>
    <%= request.getParameter("first_name")%>
</p></li>
<li><p><b>Last Name:</b>
    <%= request.getParameter("last_name")%>
</p></li>
</ul>
</body>
</html>

```

Let us put above code in main.jsp file and use it in the following HTML page:

```

<html>
<body>
<form action="main.jsp" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>

```

Keep above HTML content in a file hello.jsp and put hello.jsp and main.jsp in <Tomcat-installation-directory>/webapps/ROOT directory. When you would access <http://localhost:8080/hello.jsp>, here is the actual output of the above form.

First Name:

Last Name:

Try to enter First Name and Last Name and then click submit button. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

Next section would explain you how you would access these cookies back in your web application.

Reading Cookies with JSP:

To read cookies, you need to create an array of *javax.servlet.http.Cookie* objects by calling the **getCookies()** method of *HttpServletRequest*. Then cycle through the array, and use **getName()** and **getValue()** methods to access each cookie and associated value.

Example:

Let us read cookies which we have set in previous example:

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
    Cookie cookie = null;
    Cookie[] cookies = null;
    // Get an array of Cookies associated with this domain
    cookies = request.getCookies();
    if( cookies != null ){
        out.println("<h2> Found Cookies Name and Value</h2>");
        for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];
            out.print("Name : " + cookie.getName() + ", ");
            out.print("Value: " + cookie.getValue()+" <br/>");
        }
    }else{
        out.println("<h2>No cookies founds</h2>");
    }
%>
</body>
</html>
```

Now let us put above code in main.jsp file and try to access it. If you would have set first_name cookie as "John" and last_name cookie as "Player" then running *http://localhost:8080/main.jsp* would display the following result:

Found Cookies Name and Value

Name : first_name, Value: John

Name : last_name, Value: Player

Delete Cookies with JSP:

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps:

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using **setMaxAge()** method to delete an existing cookie.
- Add this cookie back into response header.

Example:

Following example would delete an existing cookie named "first_name" and when you would run main.jsp JSP next time it would return null value for first_name.

```
<html>
<head>
<title>Reading Cookies</title>
</head>
<body>
<center>
<h1>Reading Cookies</h1>
</center>
<%
    Cookie cookie = null;
    Cookie[] cookies = null;
    // Get an array of Cookies associated with this domain
    cookies = request.getCookies();
    if( cookies != null ){
        out.println("<h2> Found Cookies Name and Value</h2>");
        for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];
            if((cookie.getName( )).compareTo("first_name") == 0 ){
                cookie.setMaxAge(0);
                response.addCookie(cookie);
                out.print("Deleted cookie: " +
                    cookie.getName( ) + "<br/>");
            }
            out.print("Name : " + cookie.getName( ) + ", ");
            out.print("Value: " + cookie.getValue( )+" <br/>");
        }
    }else{
        out.println(
            "<h2>No cookies found</h2>");
    }
%>
</body>
</html>
```

Now let us put above code in main.jsp file and try to access it. It would display the following result:

Cookies Name and Value

Deleted cookie : first_name

Name : first_name, Value: John

Name : last_name, Value: Player

Now try to run *http://localhost:8080/main.jsp* once again and it should display only one cookie as follows:

Found Cookies Name and Value

Name : last_name, Value: Player

You can delete your cookies in Internet Explorer manually. Start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies.