COMPANY PROFILE

Pawar Infotech. provides a quality ERP services and its sole purpose is to bridge the gap between high-quality trainings and their affordability. The brand plans to accomplish this by having like-minded business partners and quality resources like training methodology, material and the instructor.

We are intended to build a good relationship with our valuable clients and provide them an end to end training, mentoring, coaching and business solutions.

During the last few years we have managed to create a close-knit community of expert trainers, associations, industry bodies, certification providers and working professionals. This itself proves our standards.

****

**HTML:**

Learn basic HTML to create your Web site. Basic HTML is not hard to learn. Learning HTML is probably the single most important thing you'll ever do if you want to create your own personal Web site. You must [learn HTML](http://personalweb.about.com/library/glossary/bldef-html.htm) to design good pages because it is the language in which Web site on the Internet are based on.

Web sites are a great way to show off something in your life. Basic HTML will let you show the world whatever it is you want to show them on your Web site. Adding colors, changing text size and including pictures on your Web site are just a few of the things you can do when you learn basic HTML.

To learn basic HTML you must keep in mind that it's simply a series of letters that are abbreviations of what they actually stand for. For example, H1 stands for a heading for a paragraph that is the first of five sizes and BR is a line break.

One important thing to remember, while you learn basic HTML, is that on a Web page the HTML tags must come in a certain order and most HTML tags must have a start and end tag for the browser to recognize the command. An end tag is just simply the same as the start tag except that it starts with the **/** symbol. A heading would look like this**<H1>Heading Here</H1>**. There is a start tag,**H1**, the heading, and an end tag, **/H1**.

Learn basic HTML the right way and remember that the tags must come in a certain order. The basic structure of a web page is as follows:

**<HTML>**

**<HEAD>** Here you can put important document information such as frames, language, and special instructions.

**<TITLE>**Title of your page.**</TITLE>**

**</HEAD>**

**<BODY>** Put your story, pictures, links, and everything else here.

**<H1>**Heading of your paragraph.**</H1>**

**<P>**This is where you enter the text of your document.**</P>**

This is how to write a link:  
**<A HREF="**http://www.nameofpage.com**">**Title or what you want to say.**</A>**

**</BODY>**

**</HTML>**

You can also enter a link into the middle of a sentence. If I wanted to say "[Microsoft](http://www.microsoft.com/en/us/default.aspx)has some wonderful demos." It would look like this:

**<A HREF="**http://www.microsoft.com/en/us/default.aspx**">**Microsoft **</A>**has some wonderful demos to help you learn basic HTML.

You not only can create links to other pages but you can also create links to another place on the same page. If I wanted you to go back to the beginning of this article I would say something like "[Go Back](http://personalweb.about.com/library/weekly/aa052498.htm#Basic)" and by clicking on the hyperlink you would go back to the beginning. This one is a little more difficult because there are two parts to it. First of all you create your link:

**<A HREF="**nameofdocument**#There">**Go Back**</A>**

The word "There" is the word I want to take you back to so now I must go to that word and create and anchor so the link I just created knows where to go:

**<A NAME="**There**">There</A>**

When you learn basic HTML you can write your Web site in the text editor that comes with Windows, programs such as [NoteTab](http://www.notetab.com/" \t "_blank) and [Arachnophilia](http://www.arachnoid.com/arachnophilia/" \t "_blank), or one that comes with your web browser. Either way if you just follow the rules from when you learned basic HTML everything will turn out just fine.

The CSS visual formatting model is a way of laying out documents on visual media (such as a computer screen or a printed page). By all accounts, it is a very simple formatting model, and the publishing world has seen much more complicated ones. Still, it is not something that is simple to explain or grasp in a few words.

Various CSS properties are used to control how a document will appear in the context of this formatting model. You have seen most of them in the examples used so far, but I haven't explained their usage because I have to explain the formatting model first.

As I said, the formatting model is a pretty complicated affair. As if that wasn't enough, though, the implementations that exist, most notably in Microsoft Internet Explorer and Netscape Navigator, are respectably problematic and horrible. Explorer does a pretty good job of sticking to the model, and will behave sensibly with a few exceptions that can be dealt with simply. Navigator, on the other hand, has its own concept of what the various properties should do.

Instead of going through the theory and observing the bugs in Explorer and Navigator along the way, as I have done in the tutorials so far, I will explain the theory, that is how the CSS specification explains formatting should be done, in this tutorial, and deal with the implementations in the next one. I hope this makes the whole process easier to understand.

## What Is a Tag?

A tag is a method of formatting HTML documents.

With tags you can create italic or bold characters, make things blink, and can control the color and size of the lettering.

Tags can be "nested". This means that you can make something bold and italic and green and blinking by simply surrounding the previous tag with the next.

Tags can be used to insert pictures and graphics.

Tags can be used to create bulleted lists like this one, or numbered lists.

Tags look something like this: <b> chosen text </b> .

All tags use the < and > (less-than and greater-than symbols) to signal the browser. These are located above the comma and period keys.

Within a tag, capitalization doesn't matter. <FONT> is the same as <font> or <Font> . I usually don't bother capitalizing when I use tags. Below, I have used capital letters when I felt that the number "1" and the letter "l" might be confused.

It is essential to always close the tags! If not, the formatting will contaminate everything that follows it. But don't panic, closing tags is very easy, and even the most experienced surfers sometimes forget

**HTML:**

Learn basic HTML to create your Web site. Basic HTML is not hard to learn. Learning HTML is probably the single most important thing you'll ever do if you want to create your own personal Web site. You must [learn HTML](http://personalweb.about.com/library/glossary/bldef-html.htm) to design good pages because it is the language in which Web site on the Internet are based on.

Web sites are a great way to show off something in your life. Basic HTML will let you show the world whatever it is you want to show them on your Web site. Adding colors, changing text size and including pictures on your Web site are just a few of the things you can do when you learn basic HTML.

To learn basic HTML you must keep in mind that it's simply a series of letters that are abbreviations of what they actually stand for. For example, H1 stands for a heading for a paragraph that is the first of five sizes and BR is a line break.

One important thing to remember, while you learn basic HTML, is that on a Web page the HTML tags must come in a certain order and most HTML tags must have a start and end tag for the browser to recognize the command. An end tag is just simply the same as the start tag except that it starts with the **/** symbol. A heading would look like this**<H1>Heading Here</H1>**. There is a start tag,**H1**, the heading, and an end tag, **/H1**.

Learn basic HTML the right way and remember that the tags must come in a certain order. The basic structure of a web page is as follows:

**<HTML>**

**<HEAD>** Here you can put important document information such as frames, language, and special instructions.

**<TITLE>**Title of your page.**</TITLE>**

**</HEAD>**

**<BODY>** Put your story, pictures, links, and everything else here.

**<H1>**Heading of your paragraph.**</H1>**

**<P>**This is where you enter the text of your document.**</P>**

This is how to write a link:  
**<A HREF="**http://www.nameofpage.com**">**Title or what you want to say.**</A>**

**</BODY>**

**</HTML>**

You can also enter a link into the middle of a sentence. If I wanted to say "[Microsoft](http://www.microsoft.com/en/us/default.aspx)has some wonderful demos." It would look like this:

**<A HREF="**http://www.microsoft.com/en/us/default.aspx**">**Microsoft **</A>**has some wonderful demos to help you learn basic HTML.

You not only can create links to other pages but you can also create links to another place on the same page. If I wanted you to go back to the beginning of this article I would say something like "[Go Back](http://personalweb.about.com/library/weekly/aa052498.htm#Basic)" and by clicking on the hyperlink you would go back to the beginning. This one is a little more difficult because there are two parts to it. First of all you create your link:

**<A HREF="**nameofdocument**#There">**Go Back**</A>**

The word "There" is the word I want to take you back to so now I must go to that word and create and anchor so the link I just created knows where to go:

**<A NAME="**There**">There</A>**

When you learn basic HTML you can write your Web site in the text editor that comes with Windows, programs such as [NoteTab](http://www.notetab.com/" \t "_blank) and [Arachnophilia](http://www.arachnoid.com/arachnophilia/" \t "_blank), or one that comes with your web browser. Either way if you just follow the rules from when you learned basic HTML everything will turn out just fine.

The CSS visual formatting model is a way of laying out documents on visual media (such as a computer screen or a printed page). By all accounts, it is a very simple formatting model, and the publishing world has seen much more complicated ones. Still, it is not something that is simple to explain or grasp in a few words.

Various CSS properties are used to control how a document will appear in the context of this formatting model. You have seen most of them in the examples used so far, but I haven't explained their usage because I have to explain the formatting model first.

As I said, the formatting model is a pretty complicated affair. As if that wasn't enough, though, the implementations that exist, most notably in Microsoft Internet Explorer and Netscape Navigator, are respectably problematic and horrible. Explorer does a pretty good job of sticking to the model, and will behave sensibly with a few exceptions that can be dealt with simply. Navigator, on the other hand, has its own concept of what the various properties should do.

Instead of going through the theory and observing the bugs in Explorer and Navigator along the way, as I have done in the tutorials so far, I will explain the theory, that is how the CSS specification explains formatting should be done, in this tutorial, and deal with the implementations in the next one. I hope this makes the whole process easier to understand.

## What Is a Tag?

A tag is a method of formatting HTML documents.

With tags you can create italic or bold characters, make things blink, and can control the color and size of the lettering.

Tags can be "nested". This means that you can make something bold and italic and green and blinking by simply surrounding the previous tag with the next.

Tags can be used to insert pictures and graphics.

Tags can be used to create bulleted lists like this one, or numbered lists.

Tags look something like this: <b> chosen text </b> .

All tags use the < and > (less-than and greater-than symbols) to signal the browser. These are located above the comma and period keys.

Within a tag, capitalization doesn't matter. <FONT> is the same as <font> or <Font> . I usually don't bother capitalizing when I use tags. Below, I have used capital letters when I felt that the number "1" and the letter "l" might be confused.

It is essential to always close the tags! If not, the formatting will contaminate everything that follows it. But don't panic, closing tags is very easy, and even the most experienced surfers sometimes forget.

JSP

JavaServer Pages (JSP) is another Java technology for developing web applications. JSP was released during the time servlet technology had gained popularity as one of the best web technologies available. JSP is not meant to replace servlets, however. In fact, JSP is an extension of the servlet technology, and it is common practice to use both servlets and JSP pages in the same web applications.

Authoring JSP pages is so easy that you can write JSP applications without much knowledge of the underlying API. If you want to be a really good Java web programmer, however, you need to know both JSP and servlets. Even if you use only JSP pages in your Java web applications, understanding servlets is still very important. As you will see in this chapter and the chapters to come, JSP uses the same techniques as those found in servlet programming. For example, in JSP you work with HTTP requests and HTTP responses, request parameters, request attributes, session management, cookies, URL-rewriting, and so on. This chapter explains the relation between JSP and servlets, introduces the JSP technology, and presents many examples that you can run easily.

Note

If you are not familiar with servlet technology, read this chapter only after reading Chapters [1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm) to [7](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%207.htm), which focus specifically on creating and working with servlets.

### What's Wrong with Servlets?

The history of web server-side programming in Java started with servlets. Sun introduced servlets in 1996 as small Java-based applications for adding dynamic content to web applications. Not much later, with the increasing popularity of Java, servlets took off to become one of the most popular technologies for Internet development today.

Servlet programmers know how cumbersome it is to program with servlets, however, especially when you have to send a long HTML page that includes little code. Take the snippet in [Listing 8.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list01) as an example. The code is a fragment from a servlet-based application that displays all parameter names and values in an HTTP request.

##### Listing 8.1 Displays All Parameter/Value Pairs in a Request Using a Servlet

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

import java.util.\*;

public class MyDearServlet extends HttpServlet {

//Process the HTTP GET request

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

doPost(request, response);

}

//Process the HTTP POST request

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<HTML>");

out.println("<HEAD><TITLE>Using Servlets</TITLE></HEAD>");

out.println("<BODY BGCOLOR=#123123>");

//Get parameter names

Enumeration parameters = request.getParameterNames();

String param = null;

while (parameters.hasMoreElements()) {

param = (String) parameters.nextElement();

out.println(param + ":" + request.getParameter(param) +

"<BR>");

}

out.println("</BODY>");

out.println("</HTML>");

out.close();

} //End of doPost method

/\* other parts of the class goes here

.

.

.

\*/

} //End of class

Nearly half of the content sent from the doPost method is static HTML. However, each HTML tag must be embedded in a String and sent using the println method of the PrintWriter object. It is a tedious chore. Worse still, the HTML page may be much longer.

Another disadvantage of using servlets is that every single change will require the intervention of the servlet programmer. Even a slight graphical modification, such as changing the value of the <BODY> tag's BGCOLOR attribute from #DADADA to #FFFFFF, will need to be done by the programmer (who in this case will work under the supervision of the more graphic-savvy web designer).

Sun understood this problem and soon developed a solution. The result was JSP technology. According to Sun's web site, "JSP technology is an extension of the servlet technology created to support authoring of HTML and XML pages." Combining fixed or static template data with dynamic content is easier with JSP. Even if you're comfortable writing servlets, you will find in this chapter several compelling reasons to investigate JSP technology as a complement to your existing work.

What needs to be highlighted is that "JSP technology is an extension of the servlet technology." This means that JSP did not replace servlets as the technology for writing server-side Internet/intranet applications. In fact, JSP was built on the servlet foundation and needs the servlet technology to work.

JSP solves drawbacks in the servlet technology by allowing the programmer to intersperse code with static content, for example. If the programmer has to work with an HTML page template written by a web designer, the programmer can simply add code into the HTML page and save it as a .jsp file. If at a later stage the web designer needs to change the HTML body background color, he or she can do it without wasting the charging-by-the-hour programmer's time. He or she can just open the .jsp file and edit it accordingly.

The code in [Listing 8.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list01) can be rewritten in JSP as shown [Listing 8.2](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list02).

##### Listing 8.2 Displays All Parameter/Value Pairs in a Request Using JSP

<%@ page import="java.util.Enumeration" %>

<HTML>

<HEAD><TITLE>Using JSP</TITLE></HEAD>

<BODY BGCOLOR=#DADADA>

<%

//Get parameter names

Enumeration parameters = request.getParameterNames();

String param = null;

while (parameters.hasMoreElements()) {

param = (String) parameters.nextElement();

out.println(param + ":" + request.getParameter(param) +

"<BR>");

}

out.close();

%>

</BODY>

</HTML>

You can see that <HTML> tags stay as they are. When you need to add dynamic content, all you need to do is enclose your code in <% … %> tags.

Again, JSP is not a replacement for servlets. Rather, JSP technology and servlets together provide an attractive solution to web scripting/programming by offering platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise, and most importantly, ease of use.

### Running Your First JSP

This section invites you to write a simple JSP page and run it. The emphasis here is not on the architecture or syntax and semantics of a JSP page; instead the section demonstrates how to configure minimally the servlet/JSP container to run JSP. Tomcat 4 is used to run JSP applications. If you have installed and configured Tomcat 4 for your servlet applications, there is no more to do. If you haven't, see [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm), "Tomcat Installation and Configuration."

Note

In the JSP context, Tomcat is often referred to as a "JSP container." Because Tomcat also is used to run servlets, however, it is more common to call it a servlet/JSP container.

After reading this section, you will understand how much JSP simplifies things for servlets. To make your JSP page run, all you need to do is configure your JSP container (Tomcat) and write a JSP page. Configuration is only done once, at the beginning. No compilation is necessary.

#### Configuring Tomcat to Run a JSP Application

The first thing you need to do before you can run your JSP application is configure Tomcat so that it recognizes your JSP application. To configure Tomcat to run a particular JSP application, follow these steps:

1. Create a directory under %CATALINA\_HOME%/webapps called myJSPApp. The directory structure is shown in [Figure 8.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08fig01).

##### Figure 8.1. The JSP application directory structure for the myJSPApp application.

1. Add a subdirectory named WEB-INF under the myJSPApp directory.
2. Edit server.xml, the server configuration file, so Tomcat knows about this new JSP application. The server.xml file is located in the conf directory under %CATALINA\_HOME%. Open the file with your text editor and look for code similar to the following:
3. <Context path="/examples" docBase="examples" debug="0"
4. reloadable="true">
5. .
6. .
7. .

</Context>

Right after the closing tag </Context>, add the following code:

<Context path="/myJSPApp" docBase="myJSPApp" debug="0" reloadable="true">

</Context>

4. Restart Tomcat.

Now you can write your JSP file and store it under the myJSPApp file. Alternatively, to make it more organized, you can create a subdirectory called jsp under myJSPApp and store your JSP files there. If you do this, you don't need to change the setting in the server.xml file.

#### Writing a JSP File

A JSP page consists of interwoven HTML tags and Java code. The HTML tags represent the presentation part and the code produces the contents. In its most basic form, a JSP page can include only the HTML part, like the code shown in [Listing 8.3](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list03).

##### Listing 8.3 The Simplest JSP Page

<HTML>

<HEAD>

</HEAD>

<BODY>

JSP is easy.

</BODY>

</HTML>

Save this file as SimplePage.jsp in the myJSPApp directory. Your directory structure should resemble [Figure 8.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08fig01).

Now, start your web browser, and type the following URL:

http://localhost:8080/myJSPApp/SimplePage.jsp

The browser is shown in [Figure 8.2](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08fig02).

##### Figure 8.2. Your first JSP page.

#### Other Examples

Of course, the code in [Listing 8.3](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list03) is not a really useful page, but it illustrates the point that a JSP page does not need to have code at all. If your page is purely static, like the one in [Listing 8.3](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list03), you shouldn't put it in a JSP file because JSP files are slower to process than HTML files. You might want to use a JSP file for pure HTML tags, however, if you think the code might include Java code in the future. This saves you the trouble of changing all the links to this page at the later stage.

To write Java code in your JSP file, you embed the code in <% … %> tags. For example, the code in [Listing 8.4](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list04) is an example of intertwining Java code and HTML in a JSP file.

##### Listing 8.4 Interweaving HTML and Code

<HTML>

<HEAD>

</HEAD>

<BODY>

<%

out.println("JSP is easy");

%>

</BODY>

</HTML>

The code in [Listing 8.4](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list04) produces the same output as the one in [Listing 8.3](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list03). Notice, however, the use of the Java code to send the text. If you don't understand what out.println does, bear with me for a moment—it is discussed in detail in the next section. For now, knowing that out.println is used to send a String to the web browser is sufficient.

Notice also that the output of a JSP page is plain text consisting of HTML tags. No code section of the page will be sent to the browser.

Another example is given in [Listing 8.5](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list05). This snippet displays the string "Welcome. The server time is now" followed by the server time.

##### Listing 8.5 Displaying the Server Time

<HTML>

<HEAD>

<TITLE>Displaying the server time</TITLE>

</HEAD>

<BODY>

Welcome. The server time is now

<%

java.util.Calendar now = java.util.Calendar.getInstance();

int hour = now.get(java.util.Calendar.HOUR\_OF\_DAY);

int minute = now.get(java.util.Calendar.MINUTE);

if (hour<10)

out.println("0" + hour);

else

out.println(hour);

out.println(":");

if (minute<10)

out.println("0" + minute);

else

out.println(minute);

%>

</BODY>

</HTML>

The code in [Listing 8.5](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08list05) displays the time in the hh:mm format. Therefore, if the hour is less than 10, a "0" precedes, which means that nine will be displayed as 09 instead of 9.

### How JSP Works

Inside the JSP container is a special servlet called the page compiler. The servlet container is configured to forward to this page compiler all HTTP requests with URLs that match the .jsp file extension. This page compiler turns a servlet container into a JSP container. When a .jsp page is first called, the page compiler parses and compiles the .jsp page into a servlet class. If the compilation is successful, the jsp servlet class is loaded into memory. On subsequent calls, the servlet class for that .jsp page is already in memory; however, it could have been updated. Therefore, the page compiler servlet will always compare the timestamp of the jsp servlet with the jsp page. If the .jsp page is more current, recompilation is necessary. With this process, once deployed, JSP pages only go through the time-consuming compilation process once.

You may be thinking that after the deployment, the first user requests for a .jsp page will experience unusually slow response due to the time spent for compiling the .jsp file into a jsp servlet. To avoid this unpleasant situation, a mechanism in JSP allows the .jsp pages to be pre-compiled before any user request for them is received. Alternatively, you deploy your JSP application as a web archive file in the form of a compiled servlet. This technique is discussed in [Chapter 16](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%2016.htm), "Application Deployment."

### The JSP Servlet Generated Code

When the JSP is invoked, Tomcat creates two files in the C:\%CATALINA\_HOME%\work\localhost\examples\jsp directory. Those two files are SimplePage\_ jsp.java and SimplePage\_ jsp.class. When you open the SimplePage\_ jsp.java file, you will see the following:

package org.apache.jsp;

import javax.servlet.\*;

import javax.servlet.http.\*;

import javax.servlet.jsp.\*;

import javax.servlet.jsp.tagext.\*;

import org.apache.jasper.runtime.\*;

public class SimplePage\_jsp extends HttpJspBase {

static {

}

public SimplePage\_jsp( ) {

}

private static boolean \_jspx\_inited = false;

public final void \_jspx\_init() throws org.apache.jasper.JasperException {

}

public void \_jspService(HttpServletRequest request,

HttpServletResponse response)

throws java.io.IOException, ServletException {

JspFactory \_jspxFactory = null;

PageContext pageContext = null;

HttpSession session = null;

ServletContext application = null;

ServletConfig config = null;

JspWriter out = null;

Object page = this;

String \_value = null;

try {

if (\_jspx\_inited == false) {

synchronized (this) {

if (\_jspx\_inited == false) {

\_jspx\_init();

\_jspx\_inited = true;

}

}

}

\_jspxFactory = JspFactory.getDefaultFactory();

response.setContentType("text/html;charset=ISO-8859-1");

pageContext = jspxFactory.getPageContext(this,

request, response, "", true, 8192, true);

application = pageContext.getServletContext();

config = pageContext.getServletConfig();

session = pageContext.getSession();

out = pageContext.getOut();

// begin

[file="C:\\tomcat4\\bin\\..\\webapps\\examples\\jsp\\SimplePage.jsp";from=(0,

2);to=(2,0)]

out.println("JSP is easy");

// end

// HTML // begin

[file="C:\\tomcat4\\bin\\..\\webapps\\examples\\jsp\\SimplePage.jsp";from=(2,

2);to=(3,0)]

out.write("\r\n");

// end

}

catch (Throwable t) {

if (out != null && out.getBufferSize() != 0)

out.clearBuffer();

if (pageContext != null)

pageContext.handlePageException(t);

}

finally {

if (\_jspxFactory != null)

\_jspxFactory.releasePageContext(pageContext);

}

}

}

For now, I will defer a full explanation of the preceding code until you learn more about how the interfaces and classes are used to run a JSP page. You can read the section, "[The Generated Servlet Revisited](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08lev1sec6)," later in this chapter, to learn more about the code listed here.

### The JSP API

The JSP technology is based on the JSP API that consists of two packages: javax.servlet.jsp and javax.servlet.jsp.tagext. Both packages are given in detail in [Appendix D](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20D.htm), "The javax.servlet.jsp Package Reference," and [Appendix E](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20E.htm), "The javax.servlet.jsp.tagext Package Reference." This chapter will discuss the classes and interfaces of the javax.servlet.jsp package and javax.servlet.jsp.tagext will be discussed in [Chapter 11](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%2011.htm), "Using JSP Custom Tags."

In addition to these two packages, JSP also needs the two servlet packages—javax.servlet and javax.servlet.http. When you study the javax.servlet.jsp package, you will know why we say that JSP is an extension of servlet technology and understand why it is important that a JSP application programmer understands the servlet technology well.

The javax.servlet.jsp package has two interfaces and four classes. The interfaces are as follows:

* JspPage
* HttpJspPage

The four classes are as follows:

* JspEngineInfo
* JspFactory
* JspWriter
* PageContext

In addition, there are also two exception classes: JspException and JspError.

#### The JspPage Interface

The JspPage is the interface that must be implemented by all JSP servlet classes. This may remind you of the javax.servlet.Servlet interface in [Chapter 1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm), "The Servlet Technology," of course. And, not surprisingly, the JspPage interface does extend the javax.servlet.Servlet interface.

The JSPPage interface has two methods, JspInit and JspDestroy, whose signatures are as follows:

public void jspInit()

public void jspDestroy()

jspInit, which is similar to the init method in the javax.servlet.Servlet interface, is called when the JspPage object is created and can be used to run some initialization. This method is called only once during the life cycle of the JSP page: the first time the JSP page is invoked.

The jspDestroy method is analogous with the destroy method of the javax.servlet.Servlet interface. This method is called before the JSP servlet object is destroyed. You can use this method to do some clean-up, if you want.

Most of the time, however, JSP authors rarely make full use of these two methods. The following example illustrates how you can implement these two methods in your JSP page:

<%!

public void jspInit() {

System.out.println("Init");

}

public void jspDestroy() {

System.out.println("Destroy");

}

%>

<%

out.println("JSP is easy");

%>

Notice that the first line of the code starts with <%!. You will find the explanation of this construct in the [Chapter 9](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%209.htm), "JSP Syntax."

#### The HttpJspPage Interface

This interface directly extends the JspPage interface. There is only one method: \_ jspService. This method is called by the JSP container to generate the content of the JSP page. The \_ jspService has the following signature:

public void \_jspService(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException.

You can't include this method in a JSP page, such as in the following code:

<%!

public void jspInit() {

System.out.println("Init");

}

public void jspDestroy() {

System.out.println("Destroy");

}

public void \_jspService(HttpServletRequest request,

HttpServletResponse response) throws ServletException, IOException {

System.out.println("Service");

}

%>

This is because the page content itself represents this method. See the section "[The Generated Servlet Revisited](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm#ch08lev1sec6)."

#### The JspFactory Class

The JspFactory class is an abstract class that provides methods for obtaining other objects needed for the JSP page processing. The class has the static method getDefaultFactory that returns a JspFactory object. From the JspFactory object, a PageContext and a JspEngineInfo object can be obtained that are useful for the JSP page processing. These objects are obtained using the JspFactory class's getEngineInfo method and the getPageContext method, whose signatures are given here:

public abstract JspEngineInfo getEngineInfo()

public abstract PageContext getPageContext (

Servlet requestingServlet, ServletRequest request,

ServletResponse response, String errorPageURL,

boolean needsSession, int buffer, boolean autoFlush)

The following code is part of the \_ jspService method that is generated by the JSP container:

JspFactory \_jspxFactory = null;

PageContext pageContext = null;

jspxFactory = JspFactory.getDefaultFactory();

.

.

.

pageContext = \_jspxFactory.getPageContext(this, request,

response, "", true, 8192, true);

#### The JspEngineInfo Class

The JspEngineInfo class is an abstract class that provides information on the JSP container. Only one method, getSpecificationVersion, returns the JSP container's version number. Because this is the only method currently available, this class does not have much use.

You can obtain a JspEngineInfo object using the getEngineInfo method of the JspFactory class.

#### The PageContext Class

PageContext represents a class that provides methods that are implementation-dependent. The PageContext class itself is abstract, so in the \_ jspService method of a JSP servlet class, a PageContext object is obtained by calling the getPageContext method of the JspFactory class.

The PageContext class provides methods that are used to create other objects. For example, its getOut method returns a JspWriter object that is used to send strings to the web browser. Other methods that return servlet-related objects include the following:

* getRequest, returns a ServletRequest object
* getResponse, returns a ServletResponse object
* getServletConfig, returns a ServletConfig object
* getServletContext, returns a ServletContext object
* getSession, returns an HttpSession object

#### The JspWriter Class

The JspWriter class is derived from the java.io.Writer class and represents a Writer that you can use to write to the client browser. Of its many methods, the most important are the print and println methods. Both provide enough overloads that ensure you can write any type of data. The difference between print and println is that println always adds the new line character to the printed data.

Additional methods allow you to manipulate the buffer. For instance, the clear method clears the buffer. It throws an exception if some of the buffer's content has already been flushed. Similar to clear is the clearBuffer method, which clears the buffer but never throws any exception if any of the buffer's contents have been flushed.

SERVLETS

The servlet technology is the foundation of web application development using the Java programming language. It is one of the most important Java technologies, and it is the underlying technology for another popular Java technology for web application development: JavaServer Pages (JSP). Therefore, understanding the servlet technology and its architecture is important if you want to be a servlet developer. Even if you plan to develop your Java web application using JSP pages alone, understanding the servlet technology helps you build a more efficient and effective JSP application.

The aim of this chapter is to introduce the servlet technology and make you comfortable with it by presenting step-by-step instructions that enable you to build and run a servlet application.

In particular, this chapter discusses the following topics:

* The benefits of servlets
* Servlet application architecture
* How a servlet works
* How to write and run your first servlet application

Throughout this book, Tomcat 4.0 is used as both the servlet container and JSP container. In this chapter, you learn how to configure Tomcat quickly so that you can run your first servlet application.

Note

For a complete reference on how to configure your application, see [Chapter 16](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%2016.htm), "Application Deployment." You can find more detail on Tomcat installation in [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm), "Tomcat Installation and Configuration."

### The Benefits of Servlets

When it first emerged, this great thing we call the Internet consisted of only static contents written using Hypertext Markup Language (HTML). At that time, anyone who could author HTML pages was considered an Internet expert.

This did not last long, however.

Soon dynamic web contents were made possible through the Common Gateway Interface (CGI) technology. CGI enables the web server to call an external program and pass HTTP request information to that external program to process the request. The response from the external program is then passed back to the web server, which forwards it to the client browser. CGI programs can be written in any language that can be called by the web server. Over the course of time, Perl became the most popular language to write CGI programs.

As the Internet became more and more popular, however, the number of users visiting a popular web site increased exponentially, and it became apparent that CGI had failed to deliver scalable Internet applications. The flaw in CGI is that each client request makes the web server spawn a new process of the requested CGI program. As we all know, process creation is an expensive operation that consumes a lot of CPU cycles and computer memory.

Gradually, new and better technologies will replace CGI as the main technology for web application development. The world has witnessed the following technologies trying to dominate web development:

* ColdFusion. Allaire's ColdFusion provides HTML-like custom tags that can be used to perform a number of operations, especially querying a database. This technology had its glamorous time in the history of the World Wide Web as the main technology for web application programming. Its glorious time has since gone with the invention of other technologies.
* Server-side JavaScript (SSJS). SSJS is an extension of the JavaScript language, the scripting language that still rules client-side web programming. SSJS can access Java classes deployed at the server side using the LiveWire technology from Netscape.
* PHP. PHP is an exciting open-source technology that has matured in recent years. The technology provides easy web application development with its session management and includes some built-in functionality, such as file upload. The number of programmers embracing PHP as their technology of choice has risen sharply in recent years.
* Servlet. The servlet technology was introduced by Sun Microsystems in 1996. This technology is the main focus of this book and will be explained in more detail in this and coming chapters.
* JavaServer Pages (JSP). JSP is an extension of the servlet technology. This, too, is the center of attention in this book.
* Active Server Pages (ASP). Microsoft's ASP employs scripting technologies that work in Windows platforms, even though there have been efforts to port this technology to other operating systems. Windows ASP works with the Internet Information Server web server. This technology will soon be replaced by Active Server Pages.NET.
* Active Server Pages.NET (ASP.NET). This technology is part of Microsoft's .NET initiative. Interestingly, the .NET Framework employs a runtime called the Common Language Runtime that is very similar to Java Virtual Machine and provides a vast class library available to all .NET languages and from ASP.NET pages. ASP.NET is an exciting technology. It introduced several new technologies including state management that does not depend on cookies or URL rewriting.

In the past, ASP and servlet/JSP have been the main technologies used in web application development. With the release of ASP.NET, it is not hard to predict that this technology will become the servlet/JSP's main competitor. ASP (and ASP.NET) and servlet/JSP each have their own fans, and it is not easy to predict which one will come out the winner. The most likely outcome is that neither will be an absolute winner that corners the market; instead the technologies will probably run head-to-head in the coming years.

Servlet (and JSP) offers the following benefits that are not necessarily available in other technologies:

* Performance. The performance of servlets is superior to CGI because there is no process creation for each client request. Instead, each request is handled by the servlet container process. After a servlet is finished processing a request, it stays resident in memory, waiting for another request.
* Portability. Similar to other Java technologies, servlet applications are portable. You can move them to other operating systems without serious hassles.
* Rapid development cycle. As a Java technology, servlets have access to the rich Java library, which helps speed up the development process.
* Robustness. Servlets are managed by the Java Virtual Machine. As such, you don't need to worry about memory leak or garbage collection, which helps you write robust applications.
* Widespread acceptance. Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies. One of the advantages of this widespread acceptance is that you can easily find and purchase components that suit your needs, which saves precious development time.

### Servlet Application Architecture

A servlet is a Java class that can be loaded dynamically into and run by a special web server. This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology.

Servlets interact with clients via a request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses. However, a servlet container also can support similar protocols, such as HTTPS (HTTP over SSL) for secure transactions.

[Figure 1.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig01) provides the architecture of a servlet application.

##### Figure 1.1. The servlet application architecture.

In a JSP application, the servlet container is replaced by a JSP container. Both the servlet container and the JSP container often are referred to as the web container or servlet/JSP container, especially if a web application consists of both servlets and JSP pages.

Note

You will learn more about servlet and JSP containers in [Chapter 8](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%208.htm), "JSP Basics.

As you can see in the [Figure 1.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig01), a servlet application also can include static content, such as HTML pages and image files. Allowing the servlet container to serve static content is not preferable because the content is faster if served by a more robust HTTP server, such as the Apache web server or Microsoft Internet Information Server. As such, it is common practice to put a web server at the front to handle all client requests. The web server serves static content and passes to the servlet containers all client requests for servlets. [Figure 1.2](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig02) shows a more common architecture for a servlet application.

##### Figure 1.2. The servlet application architecture employing an HTTP server.

Caution

A Java web application architecture employing a J2EE server is different from the diagrams in Figures [1.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig01) and [1.2](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig02). This is discussed in [Chapter 28](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%2028.htm), "Enterprise JavaBeans."

### How a Servlet Works

A servlet is loaded by the servlet container the first time the servlet is requested. The servlet then is forwarded the user request, processes it, and returns the response to the servlet container, which in turn sends the response back to the user. After that, the servlet stays in memory waiting for other requests—it will not be unloaded from the memory unless the servlet container sees a shortage of memory. Each time the servlet is requested, however, the servlet container compares the timestamp of the loaded servlet with the servlet class file. If the class file timestamp is more recent, the servlet is reloaded into memory. This way, you don't need to restart the servlet container every time you update your servlet.

The way in which a servlet works inside the servlet container is depicted in the diagram in [Figure 1.3](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig03).

##### Figure 1.3. How a servlet works.

### The Tomcat Servlet Container

A number of servlet containers are available today. The most popular one—and the one recognized as the official servlet/JSP container—is Tomcat. Originally designed by Sun Microsystems, Tomcat source code was handed over to the Apache Software Foundation in October 1999. In this new home, Tomcat was included as part of the Jakarta Project, one of the projects of the Apache Software Foundation. Working through the Apache process, Apache, Sun, and other companies—with the help of volunteer programmers worldwide—turned Tomcat into a world-class servlet reference implementation. Two months after the handover, Tomcat version 3.0 was released. Tomcat went through several 3.x releases until version 3.3 was introduced.

The successor of version 3.3 is the current version, version 4.0. The 4.0 servlet container (Catalina) is based on a completely new architecture and has been developed from the ground up for flexibility and performance. Version 4.0 implements the Servlet 2.3 and JSP 1.2 specifications, and it is this version you will be using in this book.

Another popular servlet container is JRun from Allaire Corporation. JRun is available in three editions: Developer, Professional, and Enterprise. The Developer edition is free but not licensed for deployment. The Professional and Enterprise editions grant you the license for deployment with a fee. You can download JRun from <http://commerce.allaire.com/download>.

Tomcat by itself is a web server. This means that you can use Tomcat to service HTTP requests for servlets, as well as static files (HTML, image files, and so on). In practice, however, since it is faster for non-servlet, non-JSP requests, Tomcat normally is used as a module with another more robust web server, such as Apache web server or Microsoft Internet Information Server. Only requests for servlets or JSP pages are passed to Tomcat.

To write a servlet, you need at least version 1.2 of the Java Development Kit. If you have not already downloaded one, you can download JDK 1.2 from <http://java.sun.com/j2se>. The reference implementation for both servlets and JSP are not included in J2SE, but they are included in Tomcat. Tomcat is written purely in Java.

If you haven't yet installed and configured Tomcat, now's the time to do it. If you need help with these tasks, refer to [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm) for specific steps.

### Six Steps to Running Your First Servlet

After you have installed and configured Tomcat, you can put it into service. Basically, you need to follow six steps to go from writing your servlet to running it. These steps are summarized as follows:

1. Create a directory structure under Tomcat for your application.
2. Write the servlet source code. You need to import the javax.servlet package and the javax.servlet.http package in your source file.
3. Compile your source code.
4. Create a deployment descriptor.
5. Run Tomcat.
6. Call your servlet from a web browser.

The sections that follow walk you through each of these steps.

#### Step 1: Create a Directory Structure Under Tomcat

Note

The directory where Tomcat is installed is often referred to as %CATALINA\_HOME%. In previous versions of Tomcat, this directory was called %TOMCAT\_HOME%.

When you install Tomcat, several subdirectories are automatically created under the Tomcat home directory (%CATALINA\_HOME%). One of the subdirectories is webapps. The webapps directory is where you store your web applications. A web application is a collection of servlets and other content installed under a specific subset of the server's URL namespace. A separate directory is dedicated for each servlet application. Therefore, the first thing to do when you build a servlet application is create an application directory. To create a directory structure for an application called myApp, follow these steps:

1. Create a directory called myApp under the webapps directory. The directory name is important because this also appears in the URL to your servlet.
2. Create the and WEB-INF directories under myApp, and create a directory named classes under WEB-INF. The directory structure is shown in [Figure 1.4](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig04). The directory classes under WEB-INF is for your Java classes. If you have HTML files, put them directly under the myApp directory. You may also want to create a directory called images under myApp for all your image files.

##### Figure 1.4. Tomcat application directory structure.

Note that the examples, manager, ROOT, tomcat-doc, and webdav directories are for applications that are created automatically when you install Tomcat.

#### Step 2: Write the Servlet Source Code

In this step, you prepare your source code. You can write the source code yourself using your favorite text editor or copy it from the accompanying CD.

Tip

The source code for all examples in this book are also available on the book's web site. Check out [www.newriders.com](http://www.newriders.com) to download the files you need.

The code in [Listing 1.1](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01list01) shows a simple servlet called TestingServlet. The file is named TestingServlet.java. The servlet sends a few HTML tags and some text to the browser. For now, don't worry if you haven't got a clue about how it works.

##### Listing 1.1 TestingServlet.java

import javax.servlet.\*;

import javax.servlet.http.\*;

import java.io.\*;

import java.util.\*;

public class TestingServlet extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

PrintWriter out = response.getWriter();

out.println("<HTML>");

out.println("<HEAD>");

out.println("<TITLE>Servlet Testing</TITLE>");

out.println("</HEAD>");

out.println("<BODY>");

out.println("Welcome to the Servlet Testing Center");

out.println("</BODY>");

out.println("</HTML>");

}

}

Now, save your TestingServlet.java file to the WEB-INF/classes directory under myApp. Placing your source code here will make it inaccessible from a web browser. Static files, such as HTML files and image files, should be placed directly under the myApp directory or a directory under it.

Warning

Placing your source code files outside the WEB-INF directory will make them viewable from a browser.

#### Step 3: Compile Your Source Code

For your servlet source code to compile, you need to include the path to the servlet.jar file in your CLASSPATH environment variable. The servlet.jar is located in the common\lib\ subdirectory under %CATALINA\_HOME%.

For example, if you installed Tomcat under the C:\drive on Windows and you named the install directory tomcat, type the following command from the directory where TestingServlet.java resides.

javac classpath C:\tomcat\common\lib\servlet.jar TestingServlet.java

Alternatively, to save you typing the class path every time you compile your source code, you can add the complete path to the servlet.jar file to your CLASSPATH environment variable. Again, if you have installed Tomcat under C:\and named the install directory tomcat, you must add C:\tomcat\ common\lib\servlet.jar to the CLASSPATH environment variable. Afterward, you can compile your source by simply typing the following.

javac TestingServlet.java

Note

If you have forgotten how to edit the CLASSPATH environment variable, refer to [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm).

If you are using Windows, remember that the new environment variable takes effect only for new console windows. In other words, after changing a new environment variable, open a new console window for typing in your command lines.

#### Step 4: Create the Deployment Descriptor

A deployment descriptor is an optional component in a servlet application. The descriptor takes the form of an XML document called web.xml and must be located in the WEB-INF directory of the servlet application. When present, the deployment descriptor contains configuration settings specific to that application. Deployment descriptors are discussed in detail in [Chapter 16](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%2016.htm).

To create the deployment descriptor, you now need to create a web.xml file and place it under the WEB-INF directory under myApp.

The web.xml for this example application must have the following content.

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>Testing</servlet-name>

<servlet-class>TestingServlet</servlet-class>

</servlet>

</web-app>

The web.xml file has one element—web-app. You should write all your servlets under <web-app>. For each servlet, you have a <servlet> element and you need the <servlet-name> and <servlet-class> elements. The <servlet-name> is the name for your servlet, by which it is known Tomcat. The <servlet-class> is the compiled file of your servlet without the .class extension.

Having more than one servlet in an application is very common. For every servlet, you need a <servlet> element in the web.xml file. For example, the following shows you how web.xml looks if you add another servlet called Login:

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app>

<servlet>

<servlet-name>Testing</servlet-name>

<servlet-class>TestingServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>Login</servlet-name>

<servlet-class>LoginServlet</servlet-class>

</servlet>

</web-app>

#### Step 5: Run Tomcat

If Tomcat is not already running, you need to start it. See [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm) for information on how to start or run Tomcat.

#### Step 6: Call Your Servlet from a Web Browser

Now, you can call your servlet from a web browser. By default, Tomcat runs on port 8080 in the myApp virtual directory under the servlet subdirectory. The servlet that you wrote in the preceding steps is named Testing. The URL for that servlet has the following format:

http://domain-name/virtual-directory/servlet/servlet-name

Any static file can be accessed using the following URL:

http://domain-name/virtual-directory/staticFile.html

For example, a Logo.gif file under the myApp/images/ directory can be accessed using the following URL.

http://domain-name/virtual-directory/images/Logo.gif

If you run the web browser from the same computer as Tomcat, you can replace the domain-name part with "localhost". In that case, the URL for your servlet is

http://localhost:8080/myApp/servlet/Testing

In the deployment descriptor you wrote in Step 4, you actually mapped the servlet class file called TestingServlet with the name "Testing," so that your servlet can be called by specifying its class file (TestingServlet) or its name (Testing). Without a deployment descriptor, you must call the servlet by specifying its class name; that is, TestingServlet. This means that if you did not write a deployment descriptor in Step 4, you need to use the following URL to call your servlet:

http://localhost:8080/myApp/servlet/TestingServlet

Typing the URL in the Address or Location box of your web browser will give you the string "Welcome to the Servlet Testing Center," as shown in [Figure 1.5](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/_chapter%201.htm#ch01fig05).

##### Figure 1.5. The Testing servlet.

Congratulations. You have just written your first servlet.

If you don't want to type the port number each time, you can change the default port of Tomcat so that it runs on port 80, the default port for a web server. (Details on how to change the port number can be found in [Appendix A](mk:@MSITStore:D:\D%20Drive\e-books1\j2ee%20tutuors\j2ee\J2EE%20-%20Java%20for%20the%20Web%20with%20Servlets,%20JSP,%20and%20EJB,%20A%20Devel.chm::/Appendix%20A.htm).) However, the rest of the book will use Tomcat's default port 8080.

Note

You will find code for various servlets in this chapter and the next. To run each individual servlet, you need to repeat these six steps. To avoid repetition, I do not mention these steps for every servlet presented in this book. You don't need to worry about these steps if you are using a Java development tool, such as Borland's JBuilder or IBM's VisualAge, because those steps are taken care of by the RAD program.

MYSQL

Entering Queries

Make sure you are connected to the server, as discussed in the previous section. Doing so

will not in itself select any database to work with, but that's okay. At this point, it's more

important to \_nd out a little about how to issue queries than to jump right in creating

tables, loading data into them, and retrieving data from them. This section describes the

basic principles of entering commands, using several queries you can try out to familiarise

yourself with how mysql works.

Here's a simple command that asks the server to tell you its version number and the current

date. Type it in as shown below following the mysql> prompt and press Enter:

mysql> SELECT VERSION(), CURRENT\_DATE;

+--------------+--------------+

| version() | CURRENT\_DATE |

+--------------+--------------+

| 3.22.20a-log | 1999-03-19 |

+--------------+--------------+

1 row in set (0.01 sec)

mysql>

This query illustrates several things about mysql:

\_ A command normally consists of a SQL statement followed by a semicolon. (There are

some exceptions where a semicolon is not needed. QUIT, mentioned earlier, is one of

them. We'll get to others later.)

\_ When you issue a command, mysql sends it to the server for execution and displays the

results, then prints another mysql> to indicate that it is ready for another command.

\_ mysql displays query output as a table (rows and columns). The \_rst row contains

labels for the columns. The rows following are the query results. Normally, column

labels are the names of the columns you fetch from database tables. If you're retrieving

the value of an expression rather than a table column (as in the example just shown),

mysql labels the column using the expression itself.

\_ mysql shows how many rows were returned and how long the query took to execute,

which gives you a rough idea of server performance. These values are imprecise because

130 MySQL Technical Reference for Version 4.0.1-alpha

they represent wall clock time (not CPU or machine time), and because they are a\_ected

by factors such as server load and network latency. (For brevity, the \rows in set" line

is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

mysql> SELECT VERSION(), CURRENT\_DATE;

mysql> select version(), current\_date;

mysql> SeLeCt vErSiOn(), current\_DATE;

Here's another query. It demonstrates that you can use mysql as a simple calculator:

mysql> SELECT SIN(PI()/4), (4+1)\*5;

+-------------+---------+

| SIN(PI()/4) | (4+1)\*5 |

+-------------+---------+

| 0.707107 | 25 |

+-------------+---------+

The commands shown thus far have been relatively short, single-line statements. You can

even enter multiple statements on a single line. Just end each one with a semicolon:

mysql> SELECT VERSION(); SELECT NOW();

+--------------+

| version() |

+--------------+

| 3.22.20a-log |

+--------------+

+---------------------+

| NOW() |

+---------------------+

| 1999-03-19 00:15:33 |

+---------------------+

A command need not be given all on a single line, so lengthy commands that require several

lines are not a problem. mysql determines where your statement ends by looking for the

terminating semicolon, not by looking for the end of the input line. (In other words, mysql

accepts free-format input: it collects input lines but does not execute them until it sees the

semicolon.)

Here's a simple multiple-line statement:

mysql> SELECT

-> USER()

-> ,

-> CURRENT\_DATE;

+--------------------+--------------+

| USER() | CURRENT\_DATE |

+--------------------+--------------+

| joesmith@localhost | 1999-03-18 |

+--------------------+--------------+

In this example, notice how the prompt changes from mysql> to -> after you enter the \_rst

line of a multiple-line query. This is how mysql indicates that it hasn't seen a complete

Chapter 3: Introduction to MySQL: A MySQL Tutorial 131

statement and is waiting for the rest. The prompt is your friend, because it provides valuable

feedback. If you use that feedback, you will always be aware of what mysql is waiting for.

If you decide you don't want to execute a command that you are in the process of entering,

cancel it by typing \c:

mysql> SELECT

-> USER()

-> \c

mysql>

Here, too, notice the prompt. It switches back to mysql> after you type \c, providing

feedback to indicate that mysql is ready for a new command.

The following table shows each of the prompts you may see and summarises what they

mean about the state that mysql is in:

Prompt Meaning

mysql> Ready for new command.

-> Waiting for next line of multiple-line command.

'> Waiting for next line, collecting a string that begins with a single quote (`'').

"> Waiting for next line, collecting a string that begins with a double quote (`"').

Multiple-line statements commonly occur by accident when you intend to issue a command

on a single line, but forget the terminating semicolon. In this case, mysql waits for more

input:

mysql> SELECT USER()

->

If this happens to you (you think you've entered a statement but the only response is a

-> prompt), most likely mysql is waiting for the semicolon. If you don't notice what the

prompt is telling you, you might sit there for a while before realising what you need to do.

Enter a semicolon to complete the statement, and mysql will execute it:

mysql> SELECT USER()

-> ;

+--------------------+

| USER() |

+--------------------+

| joesmith@localhost |

+--------------------+

The '> and "> prompts occur during string collection. In MySQL, you can write strings

surrounded by either `'' or `"' characters (for example, 'hello' or "goodbye"), and mysql

lets you enter strings that span multiple lines. When you see a '> or "> prompt, it means

that you've entered a line containing a string that begins with a `'' or `"' quote character,

but have not yet entered the matching quote that terminates the string. That's \_ne if you

really are entering a multiple-line string, but how likely is that? Not very. More often,

the '> and "> prompts indicate that you've inadvertantly left out a quote character. For

example:

mysql> SELECT \* FROM my\_table WHERE name = "Smith AND age < 30;

">

If you enter this SELECT statement, then press Enter and wait for the result, nothing will

happen. Instead of wondering why this query takes so long, notice the clue provided by the

132 MySQL Technical Reference for Version 4.0.1-alpha

"> prompt. It tells you that mysql expects to see the rest of an unterminated string. (Do

you see the error in the statement? The string "Smith is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However,

you cannot just type \c in this case, because mysql interprets it as part of the string that

it is collecting! Instead, enter the closing quote character (so mysql knows you've \_nished

the string), then type \c:

mysql> SELECT \* FROM my\_table WHERE name = "Smith AND age < 30;

"> "\c

mysql>

The prompt changes back to mysql>, indicating that mysql is ready for a new command.

It's important to know what the '> and "> prompts signify, because if you mistakenly enter

an unterminated string, any further lines you type will appear to be ignored by mysql |

including a line containing QUIT! This can be quite confusing, especially if you don't know

that you need to supply the terminating quote before you can cancel the current command.

**Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[[1]](http://en.wikipedia.org/wiki/Software_testing#cite_note-0) Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding [software bugs](http://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

1. meets the business and technical requirements that guided its design and development;
2. works as expected; and
3. can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as [Agile](http://en.wikipedia.org/wiki/Agile_software_development), often employ [test driven development](http://en.wikipedia.org/wiki/Test_driven_development) and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

**White box testing** is when the tester has access to the internal data structures and algorithms including the code that implement these.

**Types of white box testing**

The following types of white box testing exist:

* [API](http://en.wikipedia.org/wiki/Application_programming_interface) testing (application programming interface) - testing of the application using public and private APIs
* [Code coverage](http://en.wikipedia.org/wiki/Code_coverage) - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
* [Fault injection](http://en.wikipedia.org/wiki/Fault_injection) methods - improving the coverage of a test by introducing faults to test code paths
* [Mutation testing](http://en.wikipedia.org/wiki/Mutation_testing) methods
* [Static testing](http://en.wikipedia.org/wiki/Static_testing) - White box testing includes all static testing

**Test coverage**

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important [function points](http://en.wikipedia.org/wiki/Function_points) have been tested.[[21]](http://en.wikipedia.org/wiki/Software_testing#cite_note-20)

Two common forms of code coverage are:

* Function coverage, which reports on functions executed
* Statement coverage, which reports on the number of lines executed to complete the test

They both return a [code coverage](http://en.wikipedia.org/wiki/Code_coverage) [metric](http://en.wikipedia.org/wiki/Software_metric), measured as a [percentage](http://en.wikipedia.org/wiki/Percentage).

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=17)]**Black box testing**

Main article: [Black box testing](http://en.wikipedia.org/wiki/Black_box_testing)

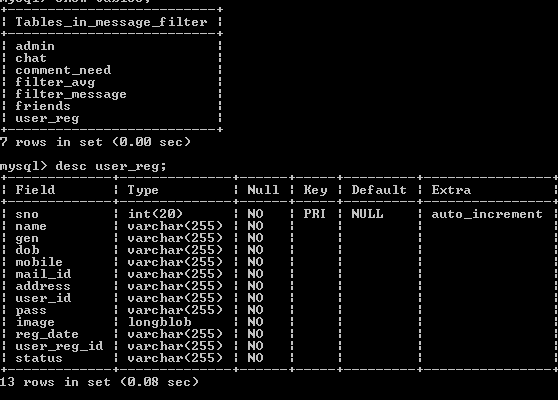
Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: [equivalence partitioning](http://en.wikipedia.org/wiki/Equivalence_partitioning), [boundary value analysis](http://en.wikipedia.org/wiki/Boundary_value_analysis), [all-pairs testing](http://en.wikipedia.org/wiki/All-pairs_testing), [fuzz testing](http://en.wikipedia.org/wiki/Fuzz_testing), [model-based testing](http://en.wikipedia.org/wiki/Model-based_testing), [exploratory testing](http://en.wikipedia.org/wiki/Exploratory_testing) and specification-based testing.

**Specification-based testing**: Specification-based testing aims to test the functionality of software according to the applicable requirements.[[22]](http://en.wikipedia.org/wiki/Software_testing#cite_note-21) Thus, the tester inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.[[23]](http://en.wikipedia.org/wiki/Software_testing#cite_note-22)

**Advantages and disadvantages**: The black box tester has no "bonds" with the code, and a tester's perception is very simple: a codemust have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. On the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed. As a result, there are situations when (1) a tester writes many test cases to check something that could have been tested by only one test case, and/or (2) some parts of the back-end are not tested at all.

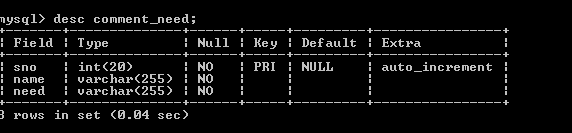
**Database design**



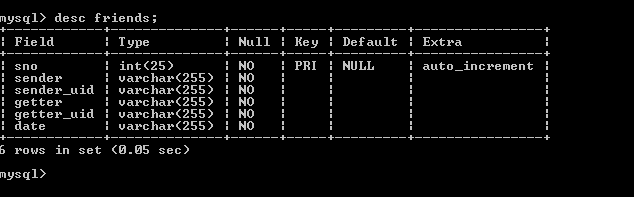
Chat:



Comment:



Friends:



Source code:

Index.jsp

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />

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

<title>A-System-to-Filter-Unwanted-Messages-from-OSN-User-Walls</title>

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

**function** val()

{

**var** a=document.f1.uname.value;

**if**(a=="")

{

alert("Enter Your Username Here");

document.f1.uname.focus();

**return** **false**;

}

**var** b=document.f1.passw.value;

**if**(b=="")

{

alert("Enter Your Password Here");

document.f1.passw.focus();

**return** **false**;

}

}

</script>

</head>

<body>

<div id=*"container"*>

<div id=*"mainpic"*>

<h1>A-System-to-Filter-Unwanted-Messages-from-OSN-User-Walls</h1>

</div>

<div id=*"menu"*>

<!--<ul>

<li class="menuitem"><a href="#">Home</a></li>

<li class="menuitem"><a href="#">About</a></li>

<li class="menuitem"><a href="#">Products</a></li>

<li class="menuitem"><a href="#">Services</a></li>

<li class="menuitem"><a href="#">Design</a></li>

<li class="menuitem"><a href="#">Contact</a></li>

</ul>-->

</div>

<div id=*"content"*>

<h2>&nbsp; </h2>

<h3>&nbsp;</h3>

<table width=*"800"*>

<tr>

<td>

<fieldset>

<table width=*"500"* height=*"300"*>

<tr>

<td><img src=*"images/cover\_image.jpg"* alt=*"cover\_main"* width=*"400"* height=*"300"* /></td>

</tr>

</table>

</fieldset>

<td>&nbsp;</td>

</td>

<td>

<fieldset>

<table width=*"200"* height=*"300"*>

<tr>

<td align=*"justify"*>

<form name=*"f1"* action=*"user\_login.jsp"* method=*"post"* onsubmit="return val();">

<p align=*"center"*> <font face=*"Courier New, Courier, monospace"* color=*"#990033"* size=*"4"*><b>&nbsp;&nbsp;&nbsp;&nbsp;UserLogin</b></font></p>

<p align=*"center"*> <img src=*"images/login.png"* alt=*"uname"* />&nbsp; <input type=*"text"* name=*"uname"*/></b></font></p>

<p align=*"center"*> <img src=*"images/pass.jpg"* alt=*"pass"* />&nbsp;<input type=*"password"* name=*"passw"*/></b></font></p>

<br />

<p align=*"center"*>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type=*"submit"* name=*"sub"* value=*"Submit"* /> &nbsp;&nbsp;<input type=*"reset"* name=*"clr"* value=*"Reset"* /></p>

</form>

<br /><br />

<p align=*"center"*>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Newuser <a href=*"registration\_user.jsp"*>Signup</a>&nbsp;Here</p>

<br /><br />

<p align=*"center"*>&nbsp;&nbsp;&nbsp;&nbsp;<font color=*"#0000FF"* size=*"3"* face=*"Verdana, Arial, Helvetica, sans-serif"*><b>Admin <a href=*"admin\_log.jsp"*>Login</a>&nbsp;Here</b></font></p>

</td>

</tr>

</table>

</fieldset>

</td>

</tr>

</table>

<br />

<br />

<div id=*"footer"*>

<p align=*"center"*> <a target=*"\_blank"* href=*"http://"*> </a></p>

</div>

</div>

</div>

</body>

</html>

Register user.jsp

<%@ page import=*"java.util.date.\*,java.util.text.DateFormat.\*,java.text.ParseException.\*"*%>

<%@page import=*"com.oreilly.servlet.\*,java.sql.\*,java.lang.\*,databaseconnection.\*,java.text.SimpleDateFormat,java.util.\*,java.io.\*,javax.servlet.\*, javax.servlet.http.\*"* %>

<%@ page import = *"java.util.Date,java.text.SimpleDateFormat,java.text.ParseException"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"* />

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

<title>A-System-to-Filter-Unwanted-Messages-from-OSN-User-Walls</title>

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

**function** valid()

{

**var** aa=document.s.un.value;

**if**(aa=="")

{

alert("Enter Name");

document.s.un.focus();

**return** **false**;

}

**if**((document.s.gen[0].checked==**false**)&&(document.s.gen[1].checked==**false**))

{

alert("Select Gender");

document.s.un.focus();

**return** **false**;

}

**var** db=document.s.dob.value;

**if**(db=="")

{

alert("Enter The Date Of Birth");

document.s.dob.focus();

**return** **false**;

}

**var** k = document.s.mob.value;

**if**(k=="")

{

alert("Enter mobile number");

document.s.mob.focus();

**return** **false**;

}

**if**(isNaN(k))

{

alert("Enter mobile number in numbers");

document.s.mob.focus();

**return** **false**;

}

**if**(k.charAt(0)!=9)

{

alert("Enter the correct mobile no");

document.s.mob.focus();

**return** **false**;

}

**if**(k.length!=10)

{

alert("Enter 10 digits");

document.s.mob.focus();

**return** **false**;

}

**var** emailfilter=/^\w+[\+\.\w-]\*@([\w-]+\.)\*\w+[\w-]\*\.([a-z]{2,4}|\d+)$/i;

**var** m=emailfilter.test(document.s.eid.value);

**if**(m==**false**)

{

alert("Please enter a valid Email Id");

document.s.eid.focus();

**return** **false**;

}

**var** ad = document.s.ad.value;

**if**(ad=="")

{

alert("Enter Address");

document.s.ad.focus();

**return** **false**;

}

**var** a=document.s.uid.value;

**if**(a=="")

{

alert("Enter User ID");

document.s.uid.focus();

**return** **false**;

}

**var** b=document.s.pass.value;

**if**(b=="")

{

alert("Enter Password");

document.s.pass.focus();

**return** **false**;

}

}

</script>

</head>

<body>

<%

java.util.Date now = **new** java.util.Date();

String DATE\_FORMAT1 = "dd/MM/yyyy";

SimpleDateFormat sdf1 = **new** SimpleDateFormat(DATE\_FORMAT1);

String strDateNew1 = sdf1.format(now);

String alphabet="MOBVCXZPNWIEURYTKSLAHFJDGQ";

**int** character=(**int**)(Math.random()\*26);

String s=alphabet.substring(character, character+4);

//System.out.println(s);

**int** y=100+(**int**)(Math.random()\*1000);

String z=s+y;

session.setAttribute("z1",z);

%>

<div id=*"container"*>

<div id=*"mainpic"*>

<h1>A-System-to-Filter-Unwanted-Messages-from-OSN-User-Walls</h1>

</div>

<div id=*"menu"*>

<ul>

<li class=*"menuitem"*><a href=*"index.html"*>Back</a></li>

<!-- <li class="menuitem"><a href="#">About</a></li>

<li class="menuitem"><a href="#">Products</a></li>

<li class="menuitem"><a href="#">Services</a></li>

<li class="menuitem"><a href="#">Design</a></li>

<li class="menuitem"><a href="#">Contact</a></li>-->

</ul>

</div>

<div id=*"content"*>

<h2>&nbsp; </h2>

<h3>&nbsp;</h3>

<table width=*"800"*>

<tr>

<td>

<fieldset>

<table width=*"500"* height=*"300"*>

<tr>

<td>

<form name=*"s"* action=*"register1\_user.jsp"* method=*"post"* onsubmit="return valid()" enctype=*"multipart/form-data"*>

<table align=*"center"* cellpadding=*"5"* cellspacing=*"5"* width=*"420"* border=*"0"*>

<tr>

<tr>

<td colspan=*"2"* align=*"center"*><strong><font color=*"#FF3300"* size=*"+1"* face=*"Courier New"*>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;Newuser Registration</font></strong></td>

</tr>

<td colspan=*"2"* align=*"center"*>&nbsp;</td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Name</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"un"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Gender</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"radio"* name=*"gen"* value=*"male"* class=*"b"*>Male&nbsp;&nbsp;&nbsp;&nbsp;

<input type=*"radio"* name=*"gen"* value=*"female"* class=*"b"*>Female</td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>DOB</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"dob"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Mobile</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"mob"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Email ID</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"eid"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Address</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <textarea name=*"ad"*></textarea></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>User ID</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"uid"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Password</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"password"* name=*"pass"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Profile Image</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"file"* name=*"fil"* class=*"b"*></td>

</tr>

<tr>

<td><font face=*"Courier New"* size=*"+1"* color=*"#000000"*><strong>Date</strong></font></td>

<td>&nbsp;&nbsp;&nbsp;&nbsp; <input type=*"text"* name=*"date"* class=*"b"* value=*"*<%=strDateNew1%>*"*></td>

</tr>

<tr>

<tr>

<td></td>

<td><input type=*"submit"* name=*"s"* value=*"submit"* class=*"b1"* >

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<input type=*"reset"* name=*"r"* value=*"clear"* class=*"b1"*></td>

</tr>

</table>

</form>

</td>

</tr>

</table>

</fieldset>

<td>&nbsp;</td>

</td>

<td>

<fieldset>

<table width=*"200"* height=*"300"* border=*"0"*>

<tr>

<td><font size=*"2"*><b>

<%

String message=request.getParameter("message");

**if**(message!=**null** && message.equalsIgnoreCase("success"))

{

out.println("<font color='red'><blink>Registered Successfully !</blink></font>");

}

%>

</b></font></td>

</tr>

<tr>

<td align=*"justify"*><p align=*"center"*><img src=*"images/reg.jpg"* />

</td>

</tr>

</table>

</fieldset>

</td>

</tr>

</table>

<br />

<br />

<div id=*"footer"*>

<p align=*"center"*> <a target=*"\_blank"* href=*"http://"*> </a></p>

</div>

</div>

</div>

</body>

</html>

Block diagrams

