TRAINING & REFERENCE

murach's Java servlets and JSP

2ND EDITION

(Chapter 1)

Thanks for downloading this chapter from ***Murach's Java Servlets and JSP (2nd Edition)***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **web site**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books for professional developers.

Thanks for your interest in our books!

MIKE MURACH & ASSOCIATES, INC.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • **www.murach.com**

Contents

1

An introduction to web programming with Java

This chapter introduces you to the concepts and terms that you need for working with servlets and JavaServer Pages (JSPs) as you create web applications. In particular, this chapter introduces you to the software that you need to be able to write, deploy, and run servlets and JSPs.

An introduction to web applications

A *web application* is a set of web pages that are generated in response to user requests. The Internet has many different types of web applications, such as search engines, online stores, auctions, news sites, discussion groups, and games.

A typical web application

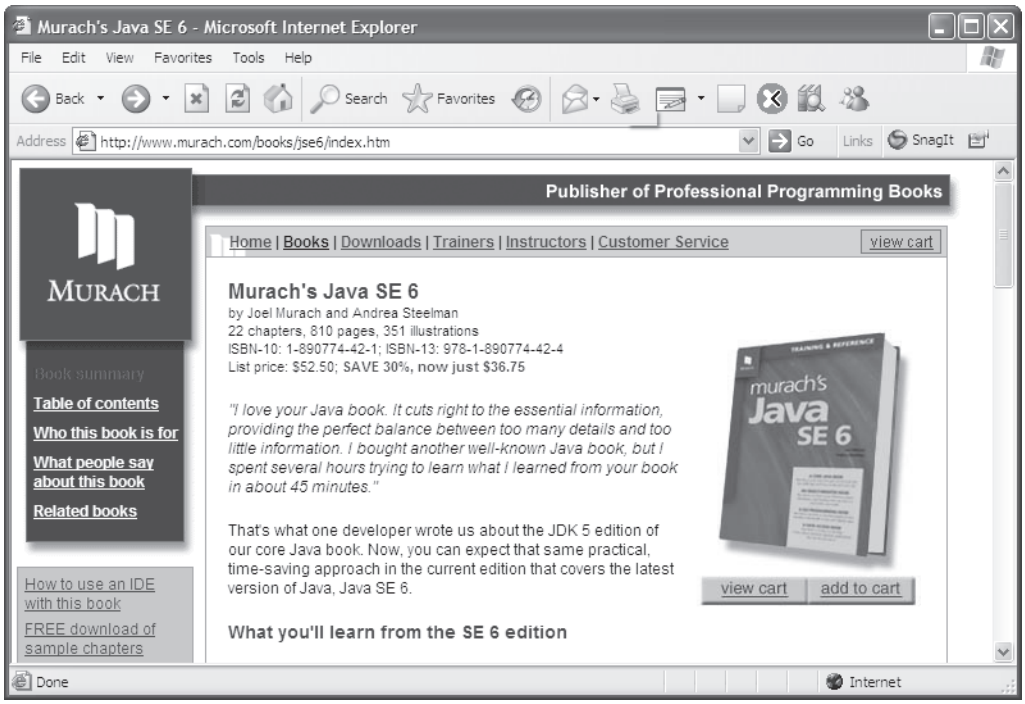
Figure 1-1 shows the first two pages of the shopping cart application that's available from www.murach.com. Here, the first page presents some information about our beginning Java book. This page contains two buttons: a View Cart button and an Add To Cart button. When you click the Add To Cart button, the web application adds the book to your cart and displays the second page in this figure, which shows all of the items in your cart.

The second page lets you change the quantity for an item or remove an item from the cart. It also lets you continue shopping or begin the checkout process. In this book, you'll learn all the skills you need to create a shopping cart application like this one.

If you take a closer look at these web pages, you can learn a little bit about how this application works. For the first page, the Address box of the browser shows an address that has an htm extension. This means that the HTML code for this page is probably stored in file with an htm extension.

In contrast, the Address box for the second page shows the address of a servlet that was mapped to the cart/displayCart URL. This means that the HTML code for this page was generated by the servlet. After the servlet address, you can see a question mark and one parameter named productCode that has a value of "jse6". This is the parameter that was passed from the first page.

The first page of a shopping cart application



The second page of a shopping cart application

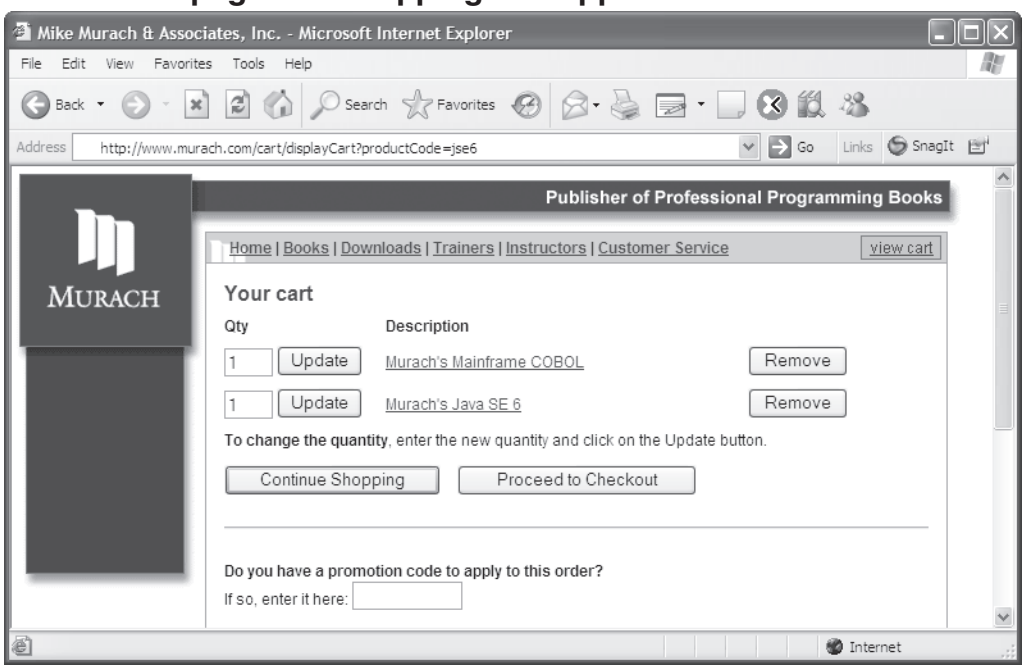


Figure 1-1 A typical web application

The components of a web application

Figure 1-2 shows the basic components that make up a web application. Because a web application is a type of *client/server application*, the components of a web application are stored on either the *client* computer or the *server* computer.

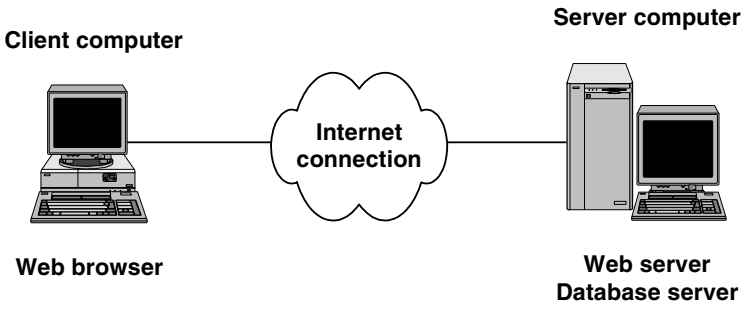
To access a web application, you use a *web browser* that runs on a client computer. The most widely used web browser is Microsoft's Internet Explorer, and the most popular alternative is Mozilla Firefox.

The web application itself is stored on the server computer. This computer runs *web server* software that enables it to send web pages to web browsers. Although there are many web servers, the most popular one for Java web applications is the Apache Software Foundation's *Apache HTTP Server*, which is usually just called *Apache*.

Because most web applications work with data that's stored in a database, most servers also run a *database management system* (*DBMS*). Two of the most popular for Java development are *Oracle* and *MySQL*. Note, however, that the DBMS doesn't have to run on the same server as the web server software. In fact, a separate database server is often used to improve an application's overall performance.

Although this figure shows the client and server computers connected via the Internet, this isn't the only way a client can connect to a server in a web application. If the client and the server are on the same *Local Area Network* (*LAN*), they function as an *intranet*. Since an intranet uses the same protocols as the Internet, a web application works the same on an intranet as it does on the Internet.

Components of a web application



Description

- Web applications are a type of *client/server application*. In a client/server application, a user at a *client* computer accesses an application at a *server* computer. For a web application, the client and server computers are connected via the Internet or an intranet.
- In a web application, the user works with a *web browser* at the client computer. The web browser provides the user interface for the application. The most widely used web browser is Microsoft's Internet Explorer, but other web browsers such as Mozilla Firefox are also widely used.
- A web application runs on the server computer under the control of *web server* software. For Java web applications, the *Apache* server is the most widely used web server.
- For most web applications, the server computer also runs a *database management system* (*DBMS*). For servlet and JSP applications, *Oracle* and *MySQL* are two of the most popular database management systems.

Figure 1-2 The components of a web application

How static web pages work

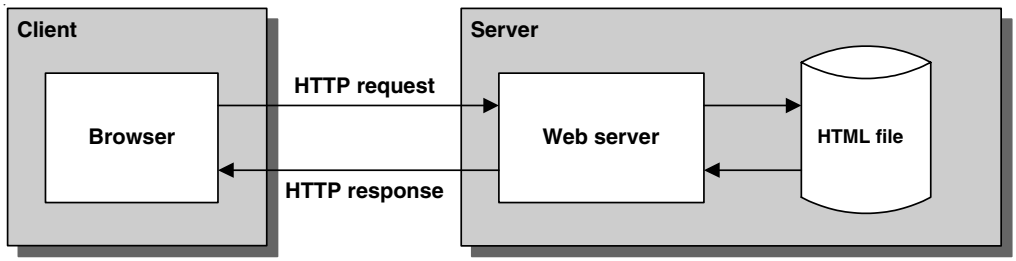
HTML (*Hypertext Markup Language*) is the language that the browser renders to the web pages that make up a web application's user interface. Many of these web pages are *static web pages*, which are the same each time they are viewed. In other words, they don't change in response to user input.

Figure 1-3 shows how a web server handles static web pages. The process begins when a user at a web browser requests a web page. This can occur when the user enters a web address into the browser's Address box or when the user clicks a link that leads to another page. In either case, the web browser uses a standard Internet protocol known as *Hypertext Transfer Protocol* (*HTTP*) to send a request known as an *HTTP request* to the web site's server.

When the web server receives an HTTP request from a browser, the server gets the requested HTML file from disk and sends the file back to the browser in the form of an *HTTP response*. The HTTP response includes the HTML document that the user requested along with any other resources specified by the HTML code such as graphics files.

When the browser receives the HTTP response, it renders the HTML document into a web page that the user can view. Then, when the user requests another page, either by clicking a link or typing another web address in the browser's Address box, the process begins again.

How a web server processes static web pages



Description

- *Hypertext Markup Language* (*HTML*) is the language that the web browser converts into the web pages of a web application.
- A *static web page* is an HTML document that's stored in a file and does not change in response to user input. Static web pages have a filename with an extension of .htm or .html.
- *Hypertext Transfer Protocol* (*HTTP*) is the protocol that web browsers and web servers use to communicate.
- A web browser requests a page from a web server by sending the server a message known as an *HTTP request*. For a static web page, the HTTP request includes the name of the HTML file that's requested.
- A web server replies to an HTTP request by sending a message known as an *HTTP response* back to the browser. For a static web page, the HTTP response includes the HTML document that's stored in the HTML file.

Figure 1-3 How static web pages work

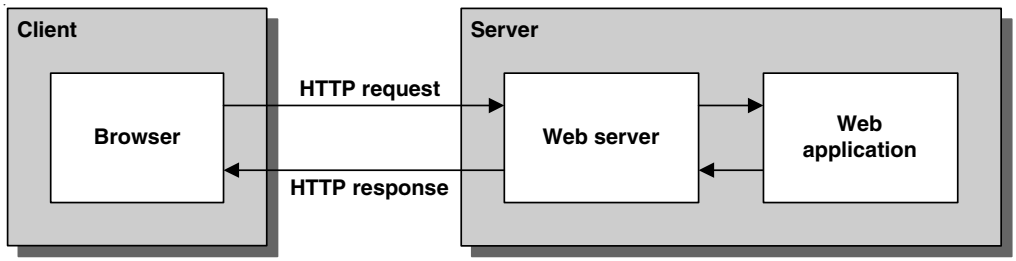
How dynamic web pages work

In contrast to a static web page, a *dynamic web page* changes based on the parameters that are sent to the web application from another page. For instance, when the Add To Cart button in the first page in figure 1-1 is clicked, the static web page calls the web application and sends one parameter to it. Then, the web application generates the dynamic web page and sends the HTML for it back to the browser.

Figure 1-4 shows how this works. When a user enters data into a web page and clicks the appropriate button, the browser sends an HTTP request to the server. This request contains the address of the next web page along with any data entered by the user. Then, when the web server receives this request and determines that it is a request for a dynamic web page, it passes the request back to the web application.

When the web application receives the request, it processes the data that the user entered and generates an HTML document. Next, it sends that document to the web server, which sends the document back to the browser in the form of an HTTP response. Then, the browser displays the HTML document that's included in the response so the process can start over again.

How a web server processes dynamic web pages



Description

- A *dynamic web page* is an HTML document that's generated by a web application. Often, the web page changes according to parameters that are sent to the web application by the web browser.
- When a web server receives a request for a dynamic web page, the server passes the request to the web application. Then, the application generates a response, which is usually an HTML document, and returns it to the web server. The web server, in turn, wraps the generated HTML document in an HTTP response and sends it back to the browser.
- The browser doesn't know or care whether the HTML was retrieved from a static HTML file or was dynamically generated by the web application. Either way, the browser displays the HTML document that is returned.

Figure 1-4 How dynamic web pages work

An introduction to Java web programming

In the early days of Java, Java received much attention for its ability to create *applets.* These are Java applications that can be downloaded from a web site and run within a web browser. However, once Microsoft's Internet Explorer stopped supporting new versions of Java, applets lost much of their appeal. As a result, many developers switched their attention to *servlets* and *JavaServer Pages* (*JSPs*). These technologies allow developers to write Java web applications that run on the server.

The components of a Java web application

Figure 1-5 shows the primary software components for a Java web application. By now, you should understand why the server must run web server software. To run a Java application, though, the server must also run a software product known as a *servlet/JSP engine*, or *servlet/JSP container*. This software allows a web server to run servlets and JSPs.

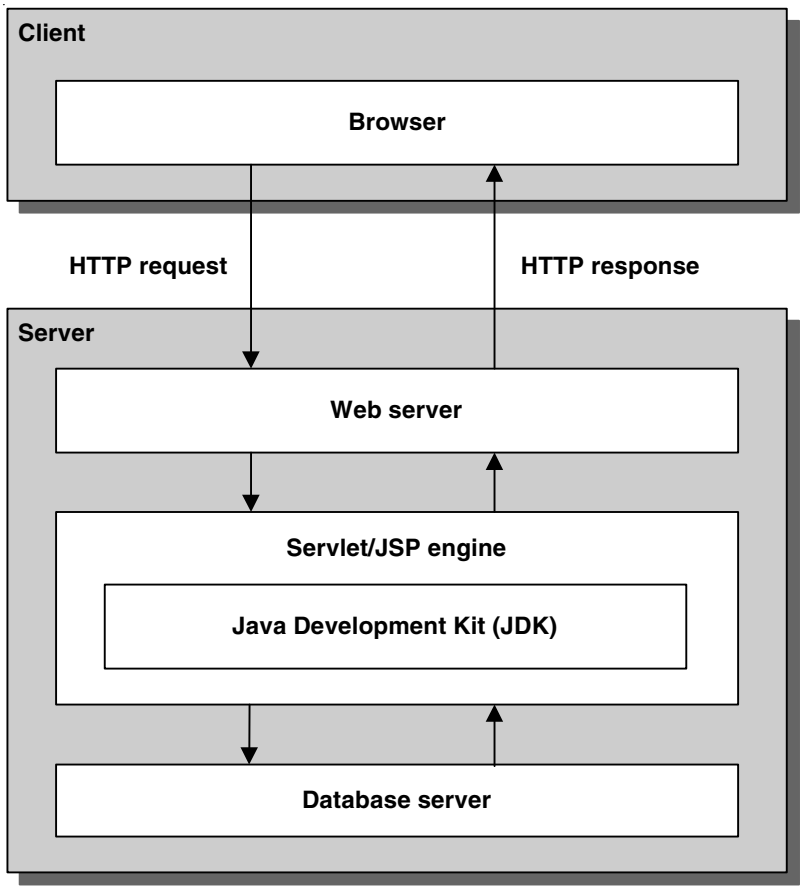
Sun's *Java Enterprise Edition* (*Java EE*) specification describes how a servlet/JSP engine should interact with a web server. Since all servlet/JSP engines must implement this specification, all servlet/JSP engines should work similarly. In theory, this makes servlet/JSP code portable between servlet/JSP engines and web servers. In practice, though, there are minor differences between each servlet/JSP engine and web server. As a result, you may need to make some modifications to your code when switching servlet/JSP engines or web servers.

Tomcat is a free, open-source servlet/JSP engine that was developed by the Jakarta project at the Apache Software Foundation. This engine is the official reference implementation of the servlet/JSP specification set forth by Sun, and it's one of the most popular servlet/JSP engines. In the next chapter, you'll learn how to install and use Tomcat on your own computer.

For a servlet/JSP engine to work properly, the engine must be able to access the *Java Development Kit (JDK)* that comes as part of the *Java Standard Edition* (*Java SE*). The JDK contains the Java compiler and the core classes for working with Java. It also contains the *Java Runtime Environment* (*JRE*) that's necessary for running compiled Java classes. Since this book assumes that you already have some Java experience, you should already be familiar with the JDK and the JRE.

Many large websites also use a Java technology known as *Enterprise JavaBeans* (*EJBs*). To use EJBs, the server must run an additional piece of software known as an *EJB server*, or *EJB container*. Although there are some benefits to using EJBs, they're more difficult to use when you're first learning how to code Java web applications, and they can make web applications unnecessarily complex. That's why this book shows how to develop web applications without using EJBs.

The components of a Java web application



Description

- Java web applications consist of JavaServer Pages and servlets. You'll learn more about them in the next two figures.
- A *servlet/JSP engine*, or *servlet/JSP container*, is the software that allows the web server to work with servlets and JSPs.
- The *Java Enterprise Edition* (*Java EE*) specification describes how web servers can interact with servlet/JSP engines. *Tomcat* is one of the most popular servlet/JSP engines. It was developed by the Jakarta project at the Apache Software Foundation.
- For a servlet/JSP engine to work, it must have access to Java's *Java Development Kit* (*JDK*), which comes as part of the *Java Standard Edition* (*Java SE*). Among other things, the JDK contains the core Java class libraries, the Java compiler, and the *Java Runtime Environment* (*JRE*).
- Java web applications that use *Enterprise JavaBeans* (*EJBs*) require an additional server component known as an *EJB server*, or *EJB container*. As a result, they won't run on the Tomcat server.

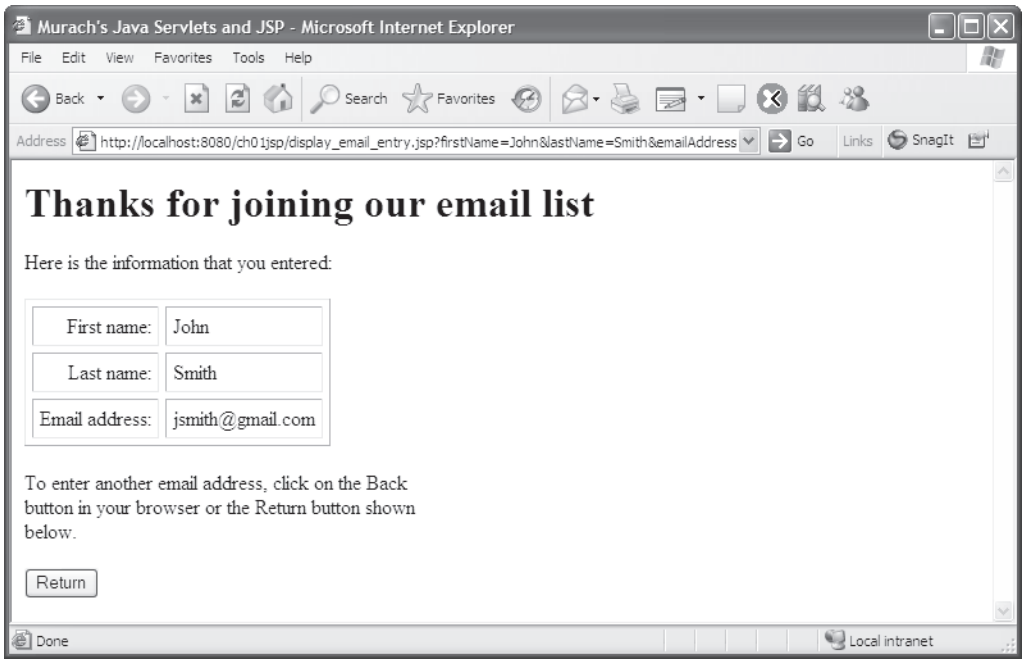
Figure 1-5 The components of a Java web application

An introduction to JavaServer Pages

To give you a better idea how *JavaServer Pages* (*JSPs*) work, part 1 of figure 1-6 shows a simple JSP displayed in a browser. Then, part 2 of this figure shows the code for the JSP.

In the Address box of the browser, the address of the JSP ends with a jsp extension. After that, the address includes a question mark followed by the parameters that are passed to the JSP. Finally, the body of the web page displays the values of these parameters in a table. For example, the value of the firstName parameter is John, and this value is displayed in the first row of the table.

A JSP that displays three parameters entered by the user



Description

- A *JavaServer Page*, or *JSP*, consists of Java code that is embedded within HTML code. This makes it easy to write the HTML portion of a JSP, but harder to write the Java code.
- When a JSP is first requested, the JSP engine translates it into a servlet and compiles it. Then, the servlet is run by the servlet engine.

Figure 1-6 An introduction to JavaServer Pages (part 1 of 2)

In part 2 of this figure, you can see the code for this JSP. If you're already familiar with HTML, you can see that most of this code consists of HTML code. In fact, the only Java code in this JSP is shaded. That makes JSPs easy to write if you know HTML and if you are able to keep the Java code to a minimum.

If a JSP requires extensive Java programming, though, it's easier to write the Java code with a servlet. In practice, web designers often write the HTML portions of the JSPs, while web programmers write the Java portions.

In case you're interested, the first three lines of Java code in this JSP get three parameters from the request object that has been passed to it. To do that, the code uses the getParameter method of the built-in request object, and it stores the values of these parameters in three String variables. Then, the three Java expressions that are used later in the JSP refer to the String variables that store the values of the parameters.

When a JSP is requested for the first time, the *JSP engine* (which is part of the servlet/JSP engine) converts the JSP code into a servlet and compiles the servlet. Then, the JSP engine loads that servlet into the servlet engine, which runs it. For subsequent requests, the JSP engine runs the servlet that corresponds to the JSP.

In chapter 4, you'll get a crash course in HTML that will teach you all the HTML you need to know for writing JSPs. Then, in chapter 5, you'll learn how to combine HTML code with Java code as you write JSPs. When you're done with those chapters, you'll know how to write significant JSPs of your own.

The code for the JSP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
    <title>Murach's Java Servlets and JSP</title>
</head>
<body>
    <%
        // get parameters from the request
        String firstName = request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress = request.getParameter("emailAddress");
    %>

    <h1>Thanks for joining our email list</h1>

    <p>Here is the information that you entered:</p>

    <table cellspacing="5" cellpadding="5" border="1">
        <tr>
            <td align="right">First name:</td>
            <td><%= firstName %></td>
        </tr>
        <tr>
            <td align="right">Last name:</td>
            <td><%= lastName %></td>
        </tr>
        <tr>
            <td align="right">Email address:</td>
            <td><%= emailAddress %></td>
        </tr>
    </table>

    <p>To enter another email address, click on the Back <br>
    button in your browser or the Return button shown <br>
    below.</p>

    <form action="join_email_list.html" method="get">
        <input type="submit" value="Return">
    </form>

</body>
</html>
```

Figure 1-6 An introduction to JavaServer Pages (part 2 of 2)

An introduction to servlets

To give you a better idea of how *servlets* work, figure 1-7 shows a servlet that generates the same web page as the JSP in figure 1-6. In short, a servlet is a Java class that runs on a server and does the processing for the dynamic web pages of a web application. That's why servlets for a web application are written by web programmers, not web designers. After the processing is done, a servlet can return HTML code to the browser by using the println method of an out object. Note, however, that this makes it more difficult to code the HTML.

If you study the code in this figure, you can see that each servlet is a Java class that extends (or inherits) the HttpServlet class. Then, each servlet can override the doGet method of the inherited class, which receives both a request and a response object from the web server, and the servlet can get the parameters that have been passed to it by using the getParameter method of the request object. After that, the servlet can do whatever processing is required by using normal Java code.

In chapter 6, you'll learn the details for coding servlets. When you complete that chapter, you'll be able to write significant servlets of our own.

How to combine servlets and JSPs in a web application

When you're developing Java web applications, you will usually want to use a combination of servlets and JSPs so you get the benefits of both. As you have seen, servlets are actually Java classes. As a result, it makes sense to use them for the processing requirements of a web application. Similarly, JSPs are primarily HTML code so it makes sense to use them for the design of the web pages in an application. But how can you do that in an efficient way?

The solution is for the servlets to do the processing for the application and then forward the request and response objects to a JSP. That way, the servlet does the processing, and the JSP provides the HTML for the user interface. With this approach, the JSP requires a minimum of embedded Java code. And that means that the web designer can write the JSPs with minimal interaction with the Java programmer, and the Java programmer can write the servlets without worrying about the HTML.

In chapter 7, you'll learn how to use this approach for developing web applications. You'll also learn how to use the Model-Controller-View (MVC) pattern to structure your applications so they're easy to manage and maintain. When you finish that chapter, you'll know how to develop Java web applications in a thoroughly professional manner.

The code for a servlet that works the same as the JSP in figure 1-6

```
package email;

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

public class DisplayEmailListServlet extends HttpServlet
{
    protected void doGet(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    {
        // get parameters from the request
        String firstName = request.getParameter("firstName");
        String lastName = request.getParameter("lastName");
        String emailAddress = request.getParameter("emailAddress");

        // return response to browser
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println(
          "<!doctype html public \"-//W3C//DTD HTML 4.0 Transitional//EN\">\n"
        + "<html>\n"
        + "<head>\n"
        + "  <title>Murach's Java Servlets and JSP</title>\n"
        + "</head>\n"
        + "<body>\n"
        + "<h1>Thanks for joining our email list</h1>\n"
        + "<p>Here is the information that you entered:</p>\n"
        + "  <table cellspacing=\"5\" cellpadding=\"5\" border=\"1\">\n"
        + "  <tr><td align=\"right\">First name:</td>\n"
        + "      <td>" + firstName + "</td>\n"
        + "  </tr>\n"
        + "  <tr><td align=\"right\">Last name:</td>\n"
        + "      <td>" + lastName + "</td>\n"
        + "  </tr>\n"
        + "  <tr><td align=\"right\">Email address:</td>\n"
        + "      <td>" + emailAddress + "</td>\n"
        + "  </tr>\n"
        + "  </table>\n"
        + "<p>To enter another email address, click on the Back <br>\n"
        + "button in your browser or the Return button shown <br>\n"
        + "below.</p>\n"
        + "<form action=\"join_email_list.html\" >\n"
        + "  <input type=\"submit\" value=\"Return\">\n"
        + "</form>\n"
        + "</body>\n"
        + "</html>\n");
        out.close();
    }
}
```

Figure 1-7 An introduction to servlets

An introduction to Java web development

This topic introduces you to servlet/JSP development. In particular, it presents some of the hardware and software options that you have as you develop Java web applications.

Three environments for servlet and JSP development

Figure 1-8 shows the three possible environments that you can use to develop servlets and JSPs. First, you can use a single computer. Second, you can use a Local Area Network (or LAN). Third, you can use the Internet.

When you use a single computer, you need to install all of the required software on that computer. That includes the JDK, the web server software, the servlet/JSP engine, and the database management system. To make this easy, you can use Tomcat as both the web server and the servlet/JSP engine. Then, you can use MySQL as the database server. In the next chapter, you'll learn how to install Tomcat, and you can learn how to install the other components in appendix A.

When you work over a LAN, it functions as an intranet. In this development environment, you can use the same software components as you do on your own computer, but you divide them between client and server. To compile and run servlets on the server, the server requires the JDK, a web server and servlet/JSP engine like Tomcat, and a DBMS like MySQL. Then, the client just needs the JDK and the JAR files for any classes that aren't available from the JDK. For example, to compile servlets on a client, the client requires the servlet.jar file, which contains all of the classes required for servlet development. These JAR files come with Tomcat, and you'll learn more about them in the next chapter.

When you work over the Internet, you use the same general components as you do when you work over an intranet. To improve performance, though, you can use a dedicated web server like Apache together with a dedicated servlet/JSP engine like Tomcat. If necessary, you can also improve the performance of an intranet application by using Apache as the web server.

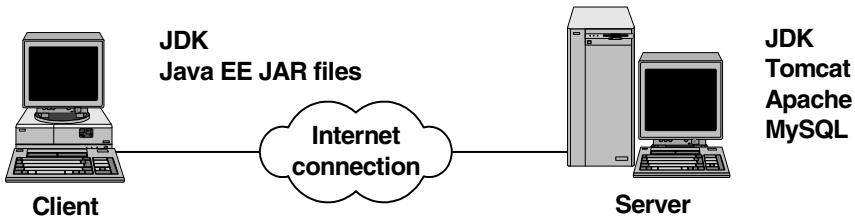
Since the JDK, Apache, Tomcat, and MySQL can be run by most operating systems, Java web developers aren't tied to a specific operating system. In fact, the Windows operating system is commonly used for the client computers during development. But when the applications are ready for use, they are often deployed to a Unix or Solaris server.

Three environments for servlet and JSP development

Stand-alone development

Local Area Network development

Internet development



Description

- When you develop web applications, you can set up your development environment in several different ways.
- If you want to develop web applications on your own computer, you need to install the JDK, a web server, a servlet/JSP engine, and a DBMS. In this case, it's common to use Tomcat as both the web server and the servlet/JSP engine, and MySQL as the DBMS.
- If you're working in a group over an intranet, the server can run Tomcat as the web server and the servlet/JSP engine, and it can run MySQL as the DBMS. Then, the client just needs the JDK and the JAR files for any classes that aren't available from the JDK. At the least, the client will need the servlet-api.jar, jsp-api.jar, and el-api.jar files that contain standard Java EE classes for working with servlets and JSPs.
- If you're working in a group over the Internet, you may want to use a web server such as Apache and a dedicated servlet/JSP engine like Tomcat. Otherwise, this works the same as when you're working over a LAN.

Figure 1-8 Three environments for servlet and JSP development

The architecture for a Java web application

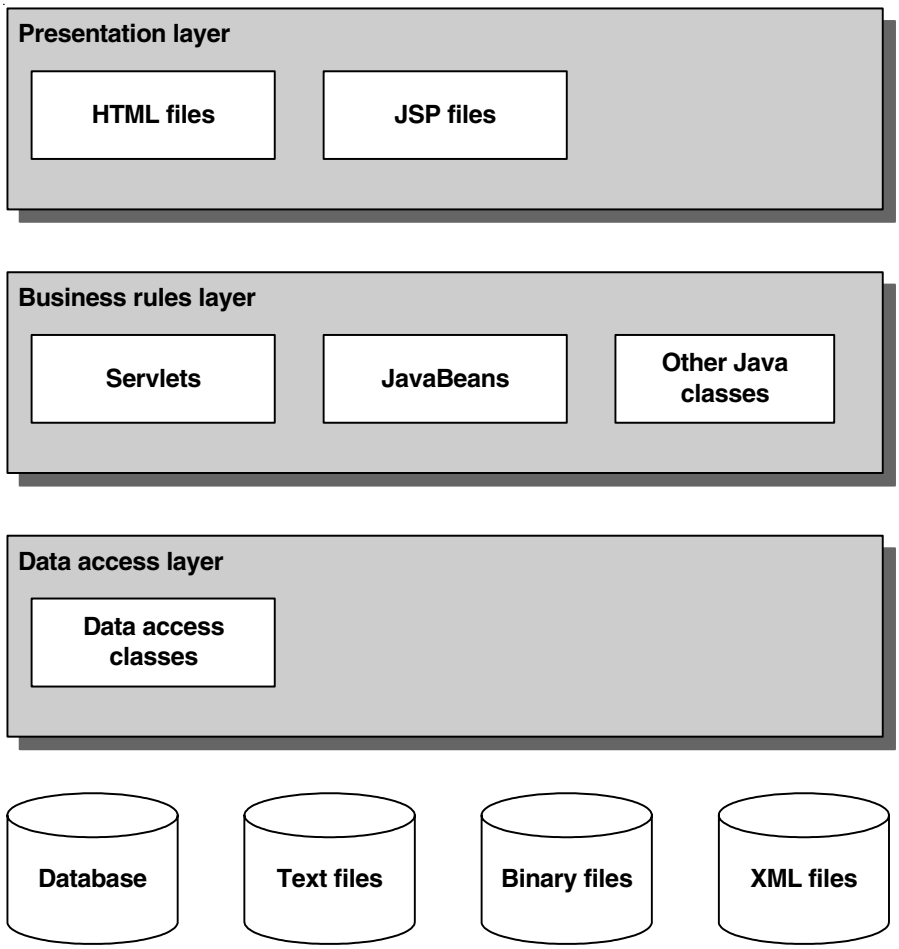
Figure 1-9 shows the architecture for a typical web application that uses servlets and JSPs. This architecture uses three layers: (1) the *presentation layer*, or *user interface layer*, (2) the *business rules layer*, and (3) the *data access layer*. In theory, the programmer tries to keep these layers as separate and independent as possible. In practice, though, these layers are often interrelated, and that's especially true for the business and data access layers.

The presentation layer consists of HTML pages and JSPs. Typically, a web designer will work on the HTML stored in these pages to create the look and feel of the user interface. Later, a Java programmer may need to edit these pages so they work properly with the servlets of the application.

The business rules layer uses servlets to control the flow of the application. These servlets may call other Java classes to store or retrieve data from a database, and they may forward the results to a JSP or to another servlet. Within the business layer, Java programmers often use a special type of Java class known as a *JavaBean* to temporarily store and process data. A JavaBean is typically used to define a business object such as a User or Invoice object.

The data layer works with data that's stored on the server's disk. For a serious web application, this data is usually stored in a relational database. However, this data can also be stored in text files and binary files. In addition, the data for an application can be stored in an *Extensible Markup Language* (*XML*) file.

The architecture for a typical Java web application



Description

- The *presentation layer* for a typical Java web application consists of HTML pages and JSPs.
- The *business rules layer* for a typical Java web application consists of servlets. These servlets may call other Java classes including a special type of Java class known as a *JavaBean*. In chapters 9 and 10, you'll learn how to use several special types of tags within a JSP to work with JavaBeans.
- The *data access layer* for a typical Java web application consists of classes that read and write data that's stored on the server's disk drive.
- For a serious web application, the data is usually stored in a relational database. However, it may also be stored in binary files, in text files, or in *Extensible Markup Language* (or *XML*) files.

Figure 1-9 The architecture for a Java web application

IDEs for developing Java web applications

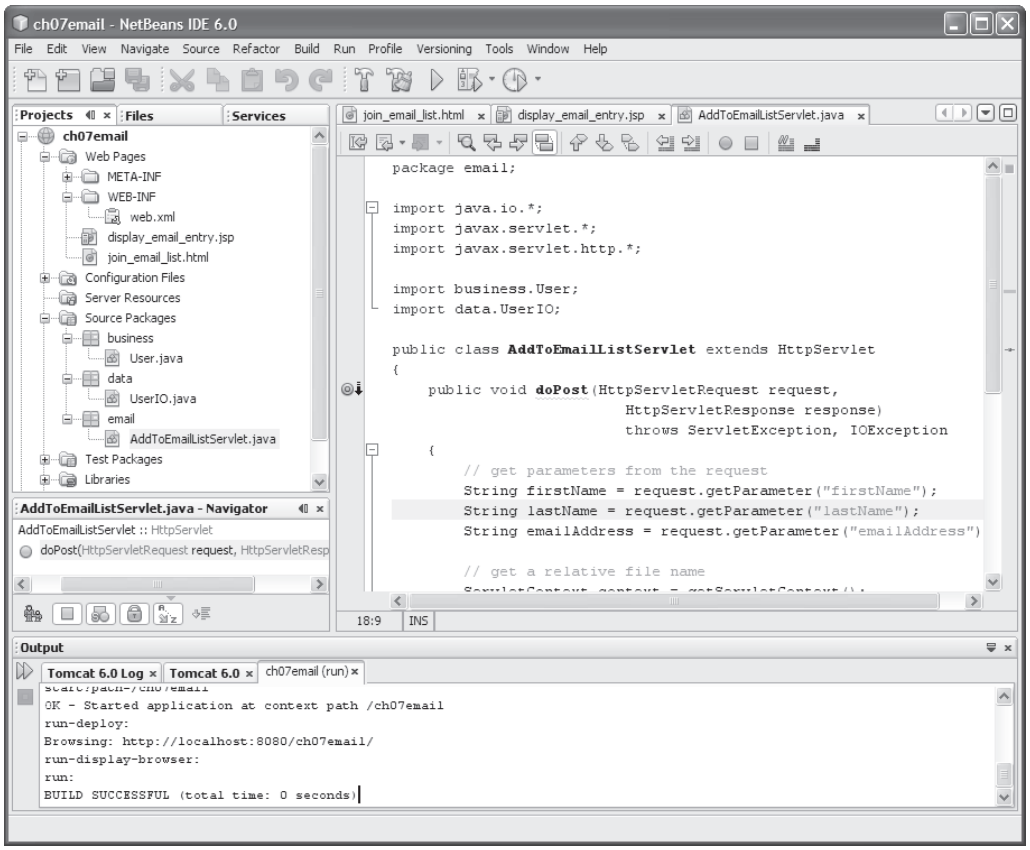
In the early days of Java web programming, programmers commonly used text editors to enter, edit, compile, and test the HTML, JSP, Java, servlet, and XML files that make up a web application. Today, however, many *Integrated Development Environments* (*IDEs*) are available that make Java web programming far more efficient.

Two of the most popular IDEs for developing Java web applications are *NetBeans* and *Eclipse*. Both are open-source, and both are available for free. Of the two, we think that NetBeans is easier to use, especially when you're getting started with web programming. That's why we recommend that you use NetBeans with this book.

In figure 1-10, for example, you can see the NetBeans IDE with the project for chapter 7 in the Projects window, the code for a servlet class in the editor window, and runtime messages in the Output window. This is similar to what you'll find in most IDEs. As a result, once you're done with this book, you can easily apply the skills that you learn with NetBeans to another IDE.

Although we recommend using NetBeans with this book, you should be able to use another IDE with this book if you prefer. To do that, though, you will need to figure out how to import the source code for this book into your IDE so you can compile and run the sample applications and complete the exercises. In addition, you will need to use the documentation that's available for your IDE to learn how to perform the tasks presented in chapter 3.

The NetBeans IDE



Popular IDEs for Java web development

NetBeans

Eclipse

JBuilder

IntelliJ IDEA

Description

- An *Integrated Development Environment* (*IDE*) is a tool that provides all of the functionality that you need for developing web applications.
- *NetBeans* and *Eclipse* are popular IDEs for Java web development that are open-source and free.
- In chapter 3, you will learn how to use NetBeans for developing Java web applications. This is the IDE that we recommend for use with this book.

Figure 1-10 IDEs for developing Java web applications

Tools for deploying Java web applications

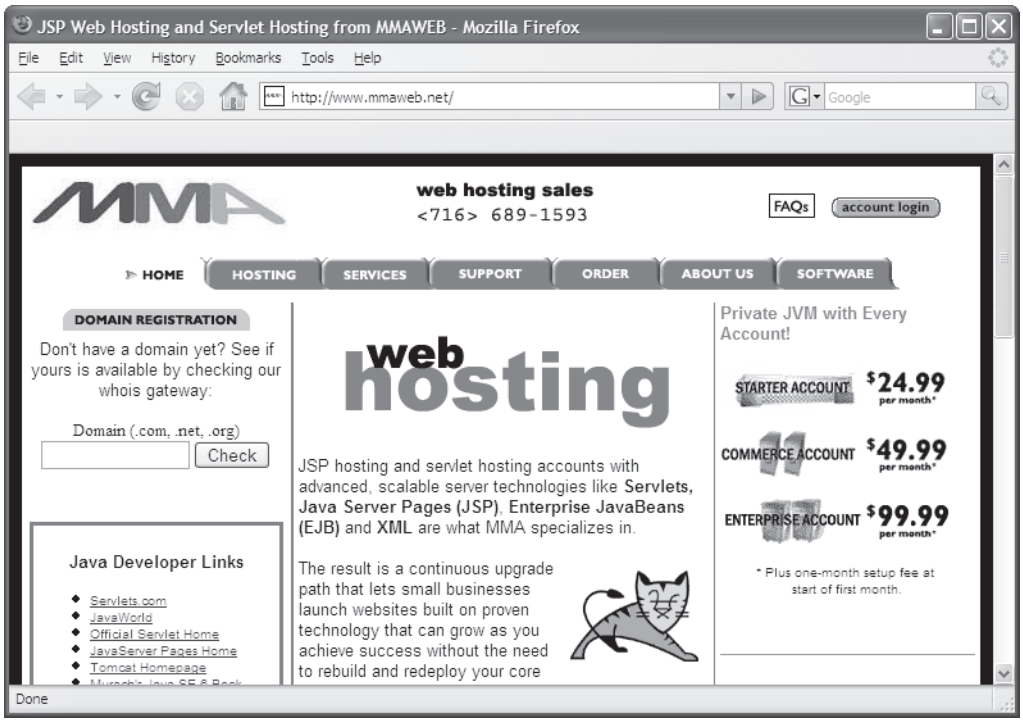
Once you've tested your servlets and JSPs on your own computer or an intranet, you may want to deploy your web application on the Internet. To do that, you need to get a *web host*. One way to do that is to find an *Internet service provider* (*ISP*) that provides web hosting that supports servlets and JSPs. If you read the text for the ISP on the web page shown in figure 1-11, for example, you can see that this ISP supports servlets and JSPs.

If you search the web, you'll be able to find many other ISPs and web hosts. Just make sure that the one you choose not only supports servlet and JSP development, but also the database management system that your application requires.

When you select a web host, you get an *IP address* like 64.71.179.86 that uniquely identifies your web site (IP stands for Internet Protocol). Then, you can get a *domain name* like www.murach.com. To do that, you can use any number of companies that you can find on the Internet. Until you get your domain name, you can use the IP address to access your site.

After you get a web host, you need to transfer your files to the web server. To do that, you can use *File Transfer Protocol* (*FTP*). The easiest way to use FTP is to use an FTP client such as the FileZilla client shown in this figure. An FTP client like this one lets you upload files from your computer to your web server and download files from your web server to your computer.

An ISP that provides web hosting that supports servlets and JSPs



The FileZilla program

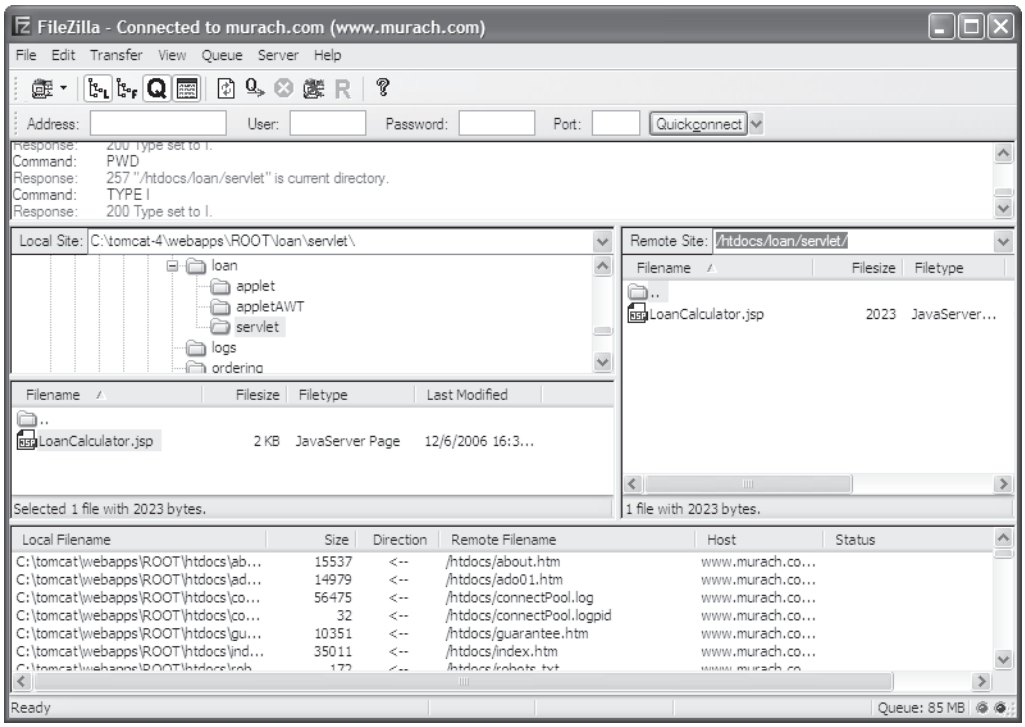


Figure 1-11 Tools for deploying Java web applications

Perspective

The goal of this chapter has been to provide the background that you need for developing servlets and JSPs. Now, if this chapter has succeeded, you should be ready to install Tomcat on your own computer as shown in the next chapter. Then, you'll be ready to install the NetBeans IDE on your computer as shown in chapter 3.

Summary

- A *web application* is a set of web pages that are generated in response to user requests.
- To run a web application, the client requires a web browser and the server requires *web server* software. The server may also require a *database management system* (*DBMS*).
- *Hypertext Markup Language* (*HTML*) is the language that the browser converts into the user interface, while *Hypertext Transfer Protocol* (*HTTP*) is the protocol that web browsers and web servers use to communicate.
- A web browser requests a page from a web server by sending an *HTTP request*. A web server replies by sending an *HTTP response* back to the browser.
- A *static web page* is generated from an HTML document that doesn't change, while a *dynamic web page* is generated by a web application based on the parameters that are included in the HTTP request.
- To run Java web applications, the server requires the *Java Development Kit* (*JDK*) and a *servlet/JSP engine* like Tomcat.
- A *JavaServer Page* (*JSP*) consists of HTML with embedded Java code. When it is requested, the JSP engine generates a servlet from the JSP and compiles that servlet. Then, the servlet engine runs that servlet.
- A *servlet* is a Java class that runs on a server. For web applications, a servlet extends the HttpServlet class. To pass HTML back to the browser, a servlet can use the println method of the out object.
- When you develop a Java web application, you can use servlets to do the processing that's required and JSPs to present the user interface.
- You can develop servlets and JSPs on your own computer, on a *Local Area Network* (*LAN*) that functions as an *intranet*, and on the Internet. When you use the Internet, it's common to use a web server that's separate from the servlet/JSP engine.
- As you develop a Java web application, you try to divide its classes into three layers: *presentation*, *business rules*, and *data access*. This makes it easier to manage and maintain the application.