Thanks for downloading this chapter from *Murach’s Java Servlets and JSP (3rd 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 on related topics.

Thanks for your interest in our books!
What developers have said about previous editions

“This is an excellent tutorial that will painlessly teach you how to write complex web applications. I doubt that there is another book available that makes the subject this easy to understand.”

Thomas Paul, JavaRanch.com

“Every so often you find a technical book that is so superbly written you actually enjoy reading it, because you know you’re going to be able to understand and learn more each time you pick it up. From the start of this book, through to the end, I found myself feeling so glad I found this book.”

Posted at an online bookseller

“You have done an INCREDIBLE job with this book. I browsed through several at the bookstore, and none of them laid these concepts out in such bite-sized, orderly chunks of practical knowledge as did yours. I can’t believe how competent I feel, and I am, after finishing the text.”

Mayo Olgundoye, Developer, New York

“This book is a pleasure to read, and I’ve learned a ton. Get it. You’ll like it. It’s well worth the price tag.”

Jeff Salter, TheServerSide.com

“I came across your book after being very frustrated by books from other publishers. Immediately, I began to develop high-powered servlet and JSP applications. I now have a deep understanding of the subject.”

David Brown, Computer Applications Engineer, Michigan

“I can’t over-recommend this book. I believe it’s the only book a beginning Java servlet and JSP programmer will need.”

Bruce Scott, Co-founder of Oracle
Get started right

The three chapters in this section provide the background information that you need for developing web applications with servlets and JavaServer Pages (JSPs). In chapter 1, you’ll learn what web programming is and how servlets and JSPs work. In chapter 2, you’ll learn how to use the MVC pattern to structure a web application that uses servlets and JSPs.

Then, in chapter 3, you’ll learn how to use the NetBeans IDE (Integrated Development Environment) to work with the HTML, servlet, JSP, and XML files that make up a servlet/JSP application. In addition, you’ll learn how to use NetBeans to run a Java web application on a Tomcat server that’s running on your computer.
An introduction to web programming with Java

This chapter introduces some concepts and terms that you should learn before you start web programming with servlets and JavaServer Pages (JSPs). In addition, it describes three approaches that you can use for developing Java web applications. Finally, it describes some software that you can use to develop and deploy servlet/JSP applications.
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 Java book. This page contains an Add To Cart link. When you click on this link, the web application adds the book to your cart and displays the second page, 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 a 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 ordering/cart URL. This means that the HTML code for this page was generated by a servlet. After the servlet address, you can see a question mark and one parameter named productCode that has a value of “htm5”. This is the parameter that was passed from the first page.
The first page of a shopping cart application

![Image of the first page of a shopping cart application]

The second page of a shopping cart application

![Image of 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. One widely used web browser is Google Chrome, and a 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 Apache Software Foundation’s Apache HTTP Server is one of the most popular. Many developers refer to it just as Apache.

Because most web applications work with data that’s stored in a database, most servers also run a database management system (DBMS), which is also known as a database server. Two of the most popular for Java development are MySQL and Oracle. When you use a database server, it 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. One widely used web browser is Google Chrome, but other web browsers such as Mozilla Firefox and Internet Explorer are also widely used.

- A web application runs on the server computer under the control of *web server* software. The Apache server is one of the most widely used web servers.

- For most web applications, the server computer also runs a *database management system (DBMS)*, which is also known as a *database server*. For servlet and JSP applications, Oracle and MySQL are two of the most popular database management systems.
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. Some 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 website’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.
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, such as 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.

- Most modern web applications store and retrieve data from a database that runs on a database server.

- 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.
Three approaches for Java web applications

There are many ways to develop Java web applications. Figure 1-5 describes three approaches that are commonly used today. When developing Java web applications, you typically use parts of the Java Enterprise Edition (Java EE) specification. This specification describes how web servers can interact with all Java web technologies including servlets, JavaServer Pages (JSP), JavaServer Faces (JSF), Java Persistence API (JPA), Enterprise JavaBeans (EJB), and more.

Servlet/JSP

In a well-structured servlet/JSP application, servlets store the Java code that does the server-side processing, and JavaServer Pages (JSPs) store the HTML that defines the user interface. This HTML typically contains links to CSS and JavaScript files. To run a web application that uses servlets and JSPs, you only need to work with the servlet/JSP part of the Java EE specification.

Since the servlet/JSP API is a relatively low-level API, it doesn’t do as much work for the developer as the other two APIs. However, the servlet/JSP API gives the developer a high degree of control over the HTML, CSS, and JavaScript that’s returned to the browser. In addition, the servlet/JSP API is the foundation for the other two approaches. As a result, it’s a good place to get started with Java web programming. As you progress through this book, you’ll learn how to develop servlet/JSP applications.

JSF

JavaServer Faces (JSF) is a newer technology that’s designed to replace both servlets and JSPs. It provides a higher-level API that does more work for the programmer. When you use JSF, you typically use more Java EE features than you do with the servlet/JSP approach.

When you use JSF, you can also use Enterprise JavaBeans (EJBs) to define server-side components. Although there are some benefits to using EJBs, they’re overkill for most websites. As a result, this book doesn’t show how to use them.

Spring Framework

Like JSF, the Spring Framework is a higher-level API that does more work for the programmer than the servlet/JSP API. However, due to the way it’s structured, the Spring Framework still gives the developer a high degree of control over the HTML/CSS/JavaScript that’s returned to the browser. As a result, if control over HTML/CSS/JavaScript is a priority for your website, the Spring Framework might be the right approach for you.
Three approaches for developing Java web apps

**Servlet/JSP**
- Is a lower-level API that does less work for the programmer.
- Provides a high degree of control over the HTML/CSS/JavaScript that’s returned to the browser.

**JSF**
- Is a higher-level API that does more work for the programmer.
- Makes it more difficult to control the HTML/CSS/JavaScript that’s returned to the browser.

**Spring Framework**
- Is a higher-level API that does more work for the programmer.
- Provides a high degree of control over the HTML/CSS/JavaScript that’s returned to the browser.

**Description**

- The **Java Enterprise Edition (Java EE)** specification describes how web servers can interact with all Java web technologies including servlets, JavaServer Pages (JSP), JavaServer Faces (JSF), Java Persistence API (JPA), and Enterprise JavaBeans (EJB).

- In a well-structured servlet/JSP application, servlets store the Java code that does the server-side processing, and JavaServer Pages (JSPs) store the HTML that defines the user interface. This typically includes links to the CSS and JavaScript for the user interface.

- **JavaServer Faces (JSF)** is a newer technology than servlet/JSP that provides a higher-level API that replaces both servlets and JSPs. For more on JSF, see chapter 21.

- **Java Persistence API (JPA)** is an API for working with databases. It can be used with servlet/JSP, JSF, or Spring. For more on JPA, see chapter 13.

- **Enterprise JavaBeans (EJB)** can be used to define server-side components. Since these components are overkill for most websites, they aren’t described in this book.
An introduction to servlet/JSP web development

This topic introduces you to servlet/JSP development. In particular, it presents the software components, application architecture, and standard directories that you typically use when you develop Java web applications.

The software components

Figure 1-6 shows the primary software components for a servlet/JSP web application. By now, you should understand why the server must run web server software. In addition, to work with servlets and JSPs, the server must also run a *servlet/JSP engine*, which is also known as a *servlet/JSP container*. In this book, you’ll learn how to use the Tomcat server. This server is one of the most popular servers for Java web applications, and it includes both a web server and a servlet/JSP engine.

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.

Since all servlet/JSP engines must implement the servlet/JSP part of the Java EE specification, all servlet/JSP engines should work similarly. In theory, this makes servlet/JSP code portable between servlet/JSP engines and application 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.

Since most servlet/JSP web applications store their data in a database, the server typically runs a database server too. In this book, you’ll learn how to use MySQL as the database server. This software is open-source and commonly used with servlet/JSP applications.
The components of a servlet/JSP application

Description

- A servlet/JSP application must have a web server and a servlet/JSP engine, also known as a servlet/JSP container, to process the HTTP request and return an HTTP response, which is typically an HTML page. Most servlet/JSP applications use Tomcat as both the web server and the servlet/JSP engine.

- Most servlet/JSP applications use a database to store the data that’s used by the application. Many servlet/JSP applications use MySQL as the database, though there are many other databases to use.

- 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).
The architecture

Figure 1-7 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 works 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 the data of the application on the server’s disk. Typically, this data is stored in a relational database such as MySQL. However, this data can also be stored in text files, binary files, and XML files. Or, it can come from web services running on the other servers.
The architecture for a typical servlet/JSP application

**Presentation layer**
- HTML files
- JSP files

**Business rules layer**
- Servlets
- JavaBeans
- Other Java classes

**Data access layer**
- Data access classes

**Description**
- The *presentation layer* for a typical servlet/JSP web application consists of HTML pages and JSPs.
- The *business rules layer* for a typical servlet/JSP web application consists of servlets. These servlets may call other Java classes including a special type of Java class known as a *JavaBean*. As you progress through this book, 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 most web applications, the data is stored in a relational database such as MySQL. However, it may also be stored in binary files, text files, or XML files.
The standard directories and files

Figure 1-8 shows a simplified version of the directory structure for the Music Store application that’s presented at the end of this book. Some of these directories are part of the servlet and JSP specifications. As a result, they are standard for all web applications. For example, all web applications that use servlets must have the WEB-INF and WEB-INF\classes directories. In addition, you can optionally include other standard directories such as the WEB-INF\lib directory or the META-INF directory.

Of course, to organize and structure an application, a programmer can create other directories within these directories. In this figure, for example, the admin, cart, and download directories organize the Administration, Cart, and Download sections of the Music Store application.

To start, each web application must have a root directory. This directory can be referred to as the document root directory, or just document root. In this figure, the document root directory is named musicStore, and it is subordinate to Tomcat’s webapps directory. Then, all of the other directories and files for the application must be subordinate to this document root directory.

The WEB-INF directory that’s subordinate to the document root directory typically contains a web.xml file for the application. You’ll learn more about this file in the next chapter. In addition, you can use this directory or any of its subdirectories to store files that you don’t want to be directly accessible from the web. For example, some applications in this book use the WEB-INF directory to store data files. This prevents users from directly accessing these files.

The WEB-INF directory also has a few standard directories that are subordinate to it. In particular, the WEB-INF\classes directory is the root directory for all Java classes for the application that aren’t stored in JAR files, including servlets. Typically, these are the classes that you write. In contrast, the WEB-INF\lib directory contains the JAR files that contain any Java class libraries for the application. Typically, you get these JAR files from a third party. Remember, though, that Tomcat’s lib directory stores the JAR files for the class libraries that are available to all web applications running on the web server. As a result, you only use the WEB-INF\lib directory if you want to make the JAR file available to just that one application.

To organize the classes that you create for the application, you can store them in packages. In that case, you need to create one subdirectory for each package. For example, this figure shows five packages. Three of these packages (music.admin, music.cart, and music.download) contain servlets that work with the Administration, Cart, and Download sections of the application while the other two contain the Java classes that provide the business objects (music.business) and data access objects (music.data) for the application.

As you progress through this book, you’ll learn how to use some other standard directories and files to deploy web applications. For example, if you want to use the database connection pool that’s available from Tomcat, you can modify the context.xml file that’s stored in the META-INF directory as described in chapter 12. For now, though, this figure should give you a general idea of what it takes to deploy a web application.
The directory structure for a web application named musicStore

![directory structure diagram]

A summary of the directories and files for a web application

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>This directory and its subdirectories typically contain the HTML and JSP files for the application.</td>
</tr>
<tr>
<td>\WEB-INF</td>
<td>This directory typically contains a file named web.xml. You can use this file to configure the servlets and other components that make up the application. In addition, this directory is not directly accessible from the web.</td>
</tr>
<tr>
<td>\WEB-INF\classes</td>
<td>This directory and its subdirectories contain the servlets and other Java classes for your application. Each subdirectory must correspond with the package for the Java class.</td>
</tr>
<tr>
<td>\WEB-INF\lib</td>
<td>This directory contains any JAR files that contain Java class libraries that are used by the web application.</td>
</tr>
<tr>
<td>\META-INF</td>
<td>This directory contains the context.xml file. You can use this file to configure the web application context.</td>
</tr>
</tbody>
</table>

Description

- The top-level directory for a web application is known as its *root directory*.
- A Java web application is a hierarchy of directories and files in a standard layout defined by the Java EE specification. All Java web applications must use the first three directories that are summarized above.
- To make classes within a JAR file available to more than one web application, you can put the JAR file in Tomcat’s lib directory.
Software for developing Java web applications

This topic presents some of the software options that you have for developing Java web applications.

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 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 1-9, for example, you can see the NetBeans IDE with the project for chapter 2 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 your IDE to perform the tasks presented in chapter 3.

Since the JDK, NetBeans, Tomcat, and MySQL can be run by most operating systems, Java web developers aren’t tied to a specific operating system. In fact, developers often use Windows or Mac OS X during development. However, when the applications are ready for release, they are often deployed to a Linux or Unix server.
The NetBeans IDE

![Image of NetBeans IDE]

Popular IDEs for Java web development
- NetBeans
- Eclipse
- 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.
- Chapter 3 shows how to use NetBeans for developing Java web applications. We recommend using this IDE with this book.
Web servers for Java web applications

Figure 1-10 describes two popular web servers for Java web applications. This book shows how to use the Tomcat server, which is the most popular server for Java web applications. Tomcat includes a web server named Coyote and a servlet/JSP engine named Catalina. As a result, this server can work with the servlet/JSP part of the Java EE specification.

In contrast, the GlassFish server implements the complete Java EE specification. Since it’s more than just a servlet/JSP engine, it’s known as an application server. The advantage of using an application server like GlassFish is that it provides more features for working with other parts of the Java EE specification such as JSF and JPA. One disadvantage is that it requires more system resources than Tomcat. As a result, it may not run as quickly on your computer.

This figure also lists four other popular web servers for Java applications. These web servers include WildFly, which was formerly known as JBoss, Jetty, Oracle WebLogic, and IBM WebSphere. Each server implements some or all of the Java EE specification, and each server has its advantages and disadvantages.

Database servers for Java web applications

This figure also describes the most popular database server for Java web applications, the MySQL database server. This is the database server that you’ll learn to use in this book. Like Tomcat and GlassFish, MySQL is open-source and runs on all modern operating systems. In addition, MySQL implements most of the SQL standard. As a result, database code developed for the MySQL server should be mostly portable between database servers.

Although many Java web applications use a MySQL database, some developers prefer other open-source databases, such as PostgreSQL. On the other hand, some large enterprises prefer using a proprietary database like Oracle to store mission-critical data.
Two popular web servers

**Tomcat**
- Is a servlet/JSP engine that includes a web server.
- Is free, open-source, and runs on all modern operating systems.
- Is a popular web server for Java web applications.

**GlassFish**
- Is a complete Java EE application server.
- Is free, open-source, and runs on all modern operating systems.
- Provides more features than Tomcat.
- Requires more system resources than Tomcat.

Other popular web servers

- WildFly (formerly JBoss)
- Jetty
- Oracle WebLogic
- IBM WebSphere

A popular database server

**MySQL**
- Is a relational database server that implements most of the SQL standard.
- Is free, open-source, and runs on all modern operating systems.
- Is the most popular database server for Java web applications.

Other popular database servers

- PostgreSQL
- Oracle

Description

- A servlet/JSP engine is the software that allows a web server to work with the servlet/JSP part of the Java EE specification.
- Tomcat is one of the most popular servlet/JSP engines. It includes a web server named Coyote and a servlet/JSP engine named Catalina.
- A Java EE application server is the software that allows the web server to work with the entire Java EE specification including servlets, JSP (JavaServer Pages), JSF (JavaServer Faces), JPA (Java Persistence API), EJB (Enterprise JavaBeans), and so on.
- GlassFish is one of the most popular application servers. It includes a web server, a servlet/JSP engine that’s based on Tomcat’s servlet/JSP engine, and more.
- MySQL is one of the most popular database servers.
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 website (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 learn how to use the MVC pattern to structure a web application.

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 web pages of a web application, 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.
- The Java Enterprise Edition (Java EE) specification describes how web servers can interact with all Java web technologies.
- A servlet/JSP engine is the software that allows a web server to work with the servlet/JSP part of the Java EE specification.
- Tomcat is one of the most popular servlet/JSP engines, and most servlet/JSP applications use Tomcat as both the web server and the servlet/JSP engine.
- An application server is the software that allows the web server to work with the entire Java EE specification.
- To run Java web applications, the server requires the Java Development Kit (JDK), a web server, and a servlet/JSP engine. This allows the server to process the HTTP request and return an HTTP response.
- Most servlet/JSP applications use a database such as MySQL to store the data that’s used by the application.
- 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 maintain the application.
- The top-level directory for a web application is known as its root directory.
• An Integrated Development Environment (IDE) is a tool that provides most of the functionality that you need for developing web applications.

• NetBeans and Eclipse are popular IDEs for Java web development. Both are open-source, free, and work on all modern operating systems.

• To deploy your web application on the Internet, you can get an Internet service provider (ISP) that provides web hosting that supports servlets and JSPs.

• When a web application runs on the Internet, it has an IP (Internet Protocol) address like 64.71.179.86 and a domain name like www.murach.com. These addresses provide two ways to uniquely identify the website.

• To transfer your web application files to a web server, you can use a File Transfer Protocol (FTP) client such as FileZilla.
How to build your Java web programming skills

The easiest way is to let *Murach’s Java Servlets and JSP (3rd Edition)* be your guide! So if you’ve enjoyed this chapter, I hope you’ll get your own copy of the book today. You can use it to:

- Teach yourself how to use servlets and JSPs to develop professional, secure web applications that conform to the MVC pattern, manage state, take advantage of EL and JSTL, do database processing with JPA or JDBC...and more
- Pick up new skills whenever you want to or need to by focusing on material that’s new to you
- Look up coding details or refresh your memory on forgotten details when you’re in the middle of developing a web application
- Loan to your colleagues who are always asking you questions about servlet/JSP programming

To get your copy, you can order online at [www.murach.com](http://www.murach.com) or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

```
100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you’ve ever tried, or send it back within 60 days for a full refund...no questions asked.
```

Thanks for your interest in Murach books!