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](http://www.murach.com). 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 Olagundoye, 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
How to structure a web application with the MVC pattern

This chapter begins by presenting two patterns that you can use for servlet/JSP web development. Then, it shows how to use the second pattern, which is known as the MVC pattern, to structure a web application. There are several advantages to this approach, and it’s generally considered a best practice.

Two patterns for servlet/JSP applications .................30
The Model 1 pattern ..................................................................................................30
The Model 2 (MVC) pattern ..................................................................................32
A servlet/JSP application that uses the MVC pattern ......34
The user interface ......................................................................................................34
The HTML for the first page .................................................................................36
The CSS for both web pages .................................................................................38
The servlet for the back-end processing ......................................................... 40
The web.xml file ......................................................................................................42
The User class ........................................................................................................44
The JSP for the second page ............................................................................... 46
Perspective ...............................................................................................................48
Two patterns for servlet/JSP applications

A pattern is a standard approach that programmers use to solve common programming problems. This topic describes two patterns that you can use when you develop Java web applications. However, for a serious web application, most developers consider the second pattern to be a best practice and recommend avoiding the first pattern.

The Model 1 pattern

The Model 1 pattern that’s shown in figure 2-1 uses a JSP to handle both the request and response of the application. In addition, the JSP does all of the processing for the application.

To do that, the JSP interacts with Java classes and objects that represent the data of the business objects in the application and provide the methods that do the business processing. In this figure, for example, the application stores the data for a user in a User object.

To save the data of the business classes, the application maps the data to a database or files that can be called the data store for the application. This is also known as persistent data storage because it exists after the application ends. Usually, data access classes like the UserDB class shown in this figure store the data of the business objects in a database. Later in this book, you’ll learn two ways to create data access classes like this one.

Although the Model 1 pattern is sometimes adequate for applications that have limited processing requirements, this pattern is not recommended for most applications. As a result, it isn’t presented in this book.
The Model 1 pattern

The Model 1 pattern uses JSPs to handle all of the processing and presentation for the application. This pattern is sometimes adequate for web applications with limited processing requirements, but it is generally considered a bad practice.

- In the Model 1 pattern, the JSPs can use regular Java classes to store the data of the application and to do the business processing of the application. In addition, they can use data classes to work with the data store.
- The data store is typically a database, but it can include disk files. This is often referred to as persistent data storage because it exists after the application ends.

Description

- The Model 1 pattern uses JSPs to handle all of the processing and presentation for the application. This pattern is sometimes adequate for web applications with limited processing requirements, but it is generally considered a bad practice.

Figure 2-1  The Model 1 pattern
The Model 2 (MVC) pattern

Figure 2-2 shows the Model 2 pattern. This pattern separates the code for the application into three layers: the model, the view, and the controller. As a result, this pattern is also known as the Model-View-Controller pattern (MVC pattern). This pattern works better than the Model 1 pattern whenever the processing requirements are substantial.

In the MVC pattern, the model defines the business layer of the application. This layer is usually implemented by JavaBeans, which you’ll learn more about in a moment. This type of class defines the data for the business objects and provides the methods that do the business processing.

The view defines the presentation layer of the application. Since it’s cumbersome to use a servlet to send HTML to a browser, an MVC application uses HTML documents or JSPs to present the view to the browser.

The controller manages the flow of the application, and this work is done by one or more servlets. To start, a servlet usually reads any parameters that are available from the request. Then, if necessary, the servlet updates the model and saves it to the data store. Finally, the servlet forwards the updated model to one of several possible JSPs for presentation.

Here again, most applications need to map the data in the model to a data store. But the JavaBeans usually don’t provide the methods for storing their own data. Instead, data access classes like the UserDB class provide those methods. That separates the business logic from the data access operations.

When you use the MVC pattern, you should try to keep the model, view, and controller as independent of each other as possible. That makes it easier to modify an application later on. If, for example, you decide to modify an application so it presents the view in a different way, you should be able to modify the view layer without making any changes to the controller or model layers. In practice, it’s difficult to separate these layers completely, but complete independence is the goal.
The Model 2 (MVC) pattern

Description

- The Model 2 pattern separates the code into a model, a view, and a controller. As a result, it’s also known as the Model-View-Controller (MVC) pattern.
- The MVC pattern is commonly used to structure web applications that have significant processing requirements. That makes them easier to code and maintain.
- In the MVC pattern, the model consists of business objects like the User object, the view consists of HTML pages and JSPs, and the controller consists of servlets.
- Usually, the methods of data access classes like the UserDB class are used to read and write business objects like the User object to and from the data store.
- When you use the MVC pattern, you try to construct each layer so it’s as independent as possible. Then, if you need to make changes to one layer, any changes to the other layers are minimized.
A servlet/JSP application that uses the MVC pattern

Now that you understand how the MVC pattern works, you’re ready to see the code for a servlet/JSP application that uses this pattern. To start, you can view the user interface for this application. Then, you can see the code for the most important files of this application.

The user interface

Figure 2-3 shows the user interface for a web application that allows a user to join an email list. Here, the first page is a static HTML page that allows the user to enter his or her email address, first name, and last name. When the user enters his or her data and clicks on the Join Now button, the web browser sends an HTTP request to the server that contains this data.

On the server, a servlet gets the data that the user entered from the HTTP request, stores that data in a User object, uses the UserDB class to save the data, and returns an HTTP response to the browser. This HTTP response is a dynamic HTML page that includes the data that the user entered on the first page.

To display the first page of this app on a development computer, the browser calls this URL:

```
localhost:8080/ch02email/
```

The first part of this URL indicates that the server is running on the same server as the browser on port 8080, which is the default port for the version of Tomcat that’s included with NetBeans. The second part of this URL indicates that the name of the app is ch02email. Since the URL doesn’t specify a file name, Tomcat displays the default file for the directory, which is typically named `index.html`, `index.htm`, or `index.jsp`.

To display the second page of this app, the user can click on the Join Now button. This displays this URL:

```
localhost:8080/ch02email/emailList
```

The first two parts of this URL are the same as the first page. However, the third part of the URL is mapped to a servlet that processes the data that’s entered on the first page. You’ll see how this works in the next few figures.
The HTML page that gets data from the user

![Image of HTML page](image1.png)

Join our email list

To join our email list, enter your name and email address below.

Email: joel@murach.com
First Name: Joel
Last Name: Murach

Join Now

The JSP that displays the data

![Image of JSP](image2.png)

Thanks for joining our email list

Here is the information that you entered:

Email: joel@murach.com
First Name: Joel
Last Name: Murach

To enter another email address, click on the Back button in your browser or the Return button shown below.

Return

Description

- The first page of this application is an HTML page that gets data from the user and sends it to the server as an HTTP request.
- The second page of this application is a JSP that displays the data that the user has entered.

Figure 2-3  The user interface
The HTML for the first page

Figure 2-4 shows the HTML for the first page of the application. If you’re already familiar with HTML, you shouldn’t have any trouble understanding this code. If your HTML skills are a little rusty, chapter 4 presents a crash course in HTML that should get you up to speed for this book. For now, focus on the general concepts, and don’t worry if you don’t understand all of the details of this code.

Since there isn’t any Java code in a static HTML page, these pages are often written by web designers, not web programmers. However, web programmers may need to modify the attributes of the form or input tags to get them to work correctly with the servlet that processes the data that’s submitted by a static HTML page.

The HTML in this figure contains the tags that define the first page. In short, the h1 and p tags identify the text that’s displayed above the form. Then, the form tags mark the beginning and end of the form. Within the form, the first input tag defines a hidden field that submits hard-coded data to the server. The next three input tags define the three text boxes that get data from the user and a button that submits the data to the server.

These input tags use HTML5 to create three text boxes that have required values. As a result, if the user doesn’t enter a value for one of these text boxes, the browser displays an appropriate message that asks the user to enter a value.

All three text boxes use the name attribute to specify a name that the servlet can use to access the value that’s stored in the text box. The first text box has a name of “email”, the second text box has a name of “firstName”, and the third text box has a name of “lastName”.

The fifth input tag is for a button that submits an HTTP request to the server. Since the form tag specifies an action attribute of “emailList”, this form submits this data to the servlet URL described in the previous figure. Similarly, since the form tag specifies a method attribute of “post”, this form uses the POST method for its HTTP request.

When you’re performing a request that modifies data on the server, you typically use the POST method. However, if you’re not modifying data, you typically use a method attribute of “get” to use the GET method for the HTTP request. You’ll learn more about how this works in chapter 5.
The index.html file

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Murach's Java Servlets and JSP</title>
  <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>
  <h1>Join our email list</h1>
  <p>To join our email list, enter your name and email address below.</p>
  <form action="emailList" method="post">
   <input type="hidden" name="action" value="add">

   <label>Email:</label>
   <input type="email" name="email" required><br>

   <label>First Name:</label>
   <input type="text" name="firstName" required><br>

   <label>Last Name:</label>
   <input type="text" name="lastName" required><br>

   <input type="submit" value="Join Now" id="submit">
  </form>
</body>
</html>
```

Description

- An HTML file contains the tags that define the content of the web page.
- The action and method attributes of the form tag set up a request for the URL that’s executed when the user clicks on the submit button.
- The three text boxes represent parameters that are passed to the servlet when the user clicks the submit button.
- The parameter names are firstName, lastName, and emailAddress, and the parameter values are the strings that the user enters into the text boxes.
The CSS for both web pages

Figure 2-5 shows the CSS for both web pages of the application. If you’re already familiar with CSS, you shouldn’t have any trouble understanding this code. If your CSS skills are a little rusty, chapter 4 presents a crash course in CSS that should get you up to speed for this book. For now, focus on the general concepts, and don’t worry if you don’t understand all of the details of this code.

The CSS (Cascading Style Sheet) contains the formatting for the web pages. In this figure, the web page uses the CSS that’s stored in a file named main.css that is in the styles subdirectory of the current directory. Here, the CSS specifies the font and margins for most of the HTML elements. In addition, it specifies that the label tags should be floated to the left of the other tags until a br (break) tag is encountered.

CSS provides a way to separate the formatting of the pages from the content of the pages, which is generally considered a best practice. Since CSS controls the appearance and format of the page, it’s typically created by a web designer, not a web programmer. Of course, it’s always helpful for web programmers to understand the CSS too. In fact, the programmer and the designer often end up being the same person.
The main.css file in the styles subdirectory

```css
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 11pt;
    margin-left: 2em;
    margin-right: 2em;
}

h1 {
    color: teal;
}

label {
    float: left;
    width: 6em;
    margin-bottom: 0.5em;
}

input[type="text"], input[type="email"] {
    width: 15em;
    margin-left: 0.5em;
    margin-bottom: 0.5em;
}

br {
    clear: both;
}

#submit {
    margin-left: 0.5em;
}
```

Description

- A CSS (Cascading Style Sheet) file contains the formatting for the web pages.
The servlet for the back-end processing

To give you a better idea of how servlets work, figure 2-6 shows the code for a servlet. In short, a servlet is a special type of Java class that runs on a server and does the processing for the dynamic web pages of a web application. That’s why the servlets for a web application are written by web programmers, not web designers.

Each servlet is a Java class that extends the HttpServlet class. In this figure, for example, the servlet is a Java class named EmailListServlet that extends the HttpServlet class. This servlet begins by overriding the doPost method of the HttpServlet class. This method is executed when its URL receives an HTTP request that uses the POST method.

The doPost method accepts both a request and a response object from the web server. Within the body of this method, the first statement defines a string for a URL. Then, the second statement uses the getParameter method of the request object to get the parameter named “action” from the HTTP request. After that, this code checks whether the action is null. If so, this code sets the URL string to a value of “join”.

After setting up the URL and action variables, this code uses an if/else statement to check the action variable to determine what action to perform. If the action variable is equal to a value of “join”, this code sets the URL string so it will display the first page of the application, the index.html page.

However, if the action variable is equal to a value of “add”, this code processes the data that has been sent to the server. To do that, the first three statements within the else clause use the getParameter method of the request object to get the parameters from the HTTP request. Then, the servlet performs the required processing by using normal Java code. In this figure, for example, it stores the data in a User object, which is described in detail in the next figure, and uses the insert method of the UserDB class to save the User object in the database. After that, this code sets the User object as an attribute of the request object so it’s available to a JSP, and it sets the URL string to the JSP for the second page of the application, which is stored in a file named thanks.jsp.

After the if/else statement finishes, the servlet can return HTML to the browser by forwarding the request and response objects to the specified URL. This URL is typically a JSP, but can also be an HTML page. In this figure, for example, the code can forward these objects to the HTML file for the first page of the application or the JSP for the second page of the application. Since this code controls the flow of the web application, this servlet is known as a controller in the MVC pattern.
The EmailListServlet class

```java
package murach.email;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import murach.business.User;
import murach.data.UserDB;

public class EmailListServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
        String url = "/index.html";
        // get current action
        String action = request.getParameter("action");
        if (action == null) {
            action = "join";   // default action
        }

        // perform action and set URL to appropriate page
        if (action.equals("join")) {
            url = "/index.html"; // the "join" page
        } else if (action.equals("add")) {
            // get parameters from the request
            String firstName = request.getParameter("firstName");
            String lastName = request.getParameter("lastName");
            String email = request.getParameter("email");

            // store data in User object and save User object in database
            User user = new User(firstName, lastName, email);
            UserDB.insert(user);

            // set User object in request object and set URL
            request.setAttribute("user", user);
            url = "/thanks.jsp"; // the "thanks" page
        }

        // forward request and response objects to specified URL
        getServletContext().getRequestDispatcher(url)
                         .forward(request, response);
    }
}
```

Figure 2-6 The servlet and web.xml file for the back-end processing (part 1 of 2)
The servlet in this figure finishes by overriding the doGet method of the HttpServlet class. This method allows this servlet to handle HTTP requests that use the GET method.

Like the doPost method, the doGet method accepts both a request and a response object from the web server. The body of this method contains a single statement that calls the doPost method defined earlier in this servlet and passes it the request and response objects. As a result, the doGet method can be used to do the same processing as the doPost method. This is typical for many servlets. However, it’s also a potential security risk. To learn how to control the types of processing that the doGet and doPost methods do, please see chapter 22.

In chapter 5, you’ll learn the details for coding servlets. When you complete that chapter, you’ll be able to write significant servlets of your own.

**The web.xml file**

Figure 2-6 also shows the web.xml file for this application. Since the web.xml file describes how the web application should be configured when it is deployed on a server, this file is known as the *deployment descriptor (DD)*. If this file exists, it is always stored in the WEB-INF directory of the application.

At the minimum, this file must contain the shaded code. This code defines the XML version and the Java EE standards for the web.xml file. Typically, though, a web.xml file contains some additional code that’s used to configure the web application.

In this figure, for example, the web.xml file uses the servlet element to provide a name for the servlet presented in this figure. Then, it uses the servlet-mapping element to map that name to a URL pattern that you can use to call the servlet.

In addition, the web.xml file defines two welcome files: index.html and index.jsp. Then, if a browser specifies a URL that only includes a directory, the web server attempts to display one of the welcome pages for the application. In this figure, for example, the web server starts by attempting to display the index.html page. If this page doesn’t exist in the directory, it attempts to display the index.jsp page. These settings are typical for a Java web application.

As you progress through this book, you’ll learn more about working with a web.xml file. In chapter 5, for example, you’ll learn how to use the web.xml file to work with initialization parameters and error handling.

Before you go on, you should know that there’s a new and easier way of mapping servlets that was introduced with the servlet 3.0 specification (Tomcat 7). You’ll learn how to use it in chapter 5. Unfortunately, this technique doesn’t work with earlier versions of the servlet specification. As a result, if you’re using an older version of Tomcat, you’ll still need to use the web.xml file to map your servlets to URLs.
The EmailListServlet class

```java
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doPost(request, response);
}
}
```

The web.xml file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

    <servlet>
        <servlet-name>EmailListServlet</servlet-name>
        <servlet-class>murach.email.EmailListServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>EmailListServlet</servlet-name>
        <url-pattern>/emailList</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

Description

- Servlets contain Java code for a web application. When a servlet controls the flow of the application, it's known as a controller.
- With the servlet 3.0 specification and later, you can use the `@WebServlet` annotation to map the servlet to a URL pattern. Prior to the servlet 3.0 specification, you must use the web.xml file to map a servlet to a URL pattern.
- A servlet provides a request object that you can use to get data from the HTTP request.
- A servlet provides a response object that you can use to return an HTTP response to the browser. Typically, you return an HTTP response for an HTML page by forwarding the request and response objects to a JSP page.
- Since the web.xml file describes how the web application should be configured when it is deployed on a server, this file is known as the deployment descriptor (DD). If this file exists, it is always stored in the WEB-INF directory of the application.
The User class

Figure 2-7 shows the code for the User class. This class is a JavaBean, or bean, because it follows the three rules that all JavaBeans must follow.

First, a JavaBean must contain a zero-argument constructor, which is a constructor that doesn’t accept any arguments. In this figure, the zero-argument constructor uses three statements to set all three instance variables equal to empty strings. As a result, a newly created User object stores empty strings for its instance variables instead of null values, which is usually what you want.

Second, a JavaBean must contain get and set methods for all of the properties that need to be accessed by JSPs. In this figure, for example, the methods provide access to all of the instance variables of the User class, so this class qualifies as a bean. Of course, you can also code get and set methods that provide access to other properties in a bean.

To provide access to a Boolean value, you code is and set methods instead of get and set methods. For example, you could code methods named isEmailUpdated and setEmailUpdated to provide access to a Boolean property named emailUpdated.

When you code the get, set, and is methods, you must follow the capitalization conventions used in this figure. In other words, each method name must start with a lowercase letter, followed by a property name that starts with an uppercase letter as in firstName.

Third, a JavaBean must implement the Serializable or Externalizable interface. The Serializable interface is a tagging interface in the java.io package that indicates that a class contains get, set, and is methods that another class can use to read and write an object’s instance variables to and from a persistent data source. In this figure, for example, the User class implements the Serializable interface and contains all the necessary get and set methods. As a result, some servlet engines can save and restore this object if that’s necessary. For example, the Tomcat container can save the User object’s state before it shuts down, and it can restore the User object’s state when it starts up the next time.

When coding a web application, it’s common to use JavaBeans to define the business objects of an application. These beans can be called invisible JavaBeans because they don’t define visible components. The focus of this book is on this type of JavaBean.

You should realize, though, that JavaBeans are capable of doing much more than defining business objects. For instance, JavaBeans can be used to define buttons and other user interface controls.

You should also realize that there’s another type of JavaBean called an Enterprise JavaBean (EJB). Although EJBs are similar in some ways to JavaBeans, EJBs are more complex and difficult to code than JavaBeans. To learn more about them, you can get a book that covers the advanced features of the Java EE specification.
The User class

```java
package murach.business;

import java.io.Serializable;

public class User implements Serializable {
    private String firstName;
    private String lastName;
    private String email;

    public User() {
        firstName = "";
        lastName = "";
        email = "";
    }

    public User(String firstName, String lastName, String email) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
    }

    public String getFirstName() { return firstName; }
    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getLastName() { return lastName; }
    public void setLastName(String lastName) { this.lastName = lastName; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

Description

- A JavaBean, or bean, is a Java class that (1) provides a zero-argument constructor, (2) provides get and set methods for all of its instance variables, and (3) implements the Serializable or Externalizable interface.
- Since JavaBeans are just Java classes, they are a type of plain old Java object (POJO).
The JSP for the second page

The main benefit that you get from coding your business classes so they qualify as JavaBeans is that you can then use special JSP tags for working with the beans. This is illustrated by the JSP in figure 2-8.

Most of this code is HTML code. In fact, the only other code in this page is the three special JSP tags that are shaded. This makes JSPs easy to write if you know HTML.

The three JSP tags shown in this figure are Expression Language (EL) tags, and they’re easy to identify because they begin with a dollar sign ($) followed by an opening brace ({}), and they end with a closing brace (}). These tags work because the servlet in figure 2-6 stored the User object as an attribute of the request object with a name of user. In addition, this works because the User object has getEmail, getFirstName, and getLastName methods that provide access to the data that’s stored in the User object.

Although it’s possible to embed Java code in a JSP, that’s generally considered a bad practice. Instead, it’s considered a good practice to restructure the app to move the Java code into a servlet that forwards the request and response objects to the JSP. That way, the servlet does the processing, and the JSP provides the HTML for the user interface. With this approach, the JSP doesn’t require any embedded Java code, only special JSP tags like the ones shown in this figure. And that means that the web designer can develop the JSPs with minimal interaction with the Java programmer, and the Java programmer can develop the servlets without worrying about the HTML.

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 the old days, this sometimes led to a delay when the JSP was requested for the first time. Today, most servlet/JSP engines precompile JSPs to avoid that delay.
The thanks.jsp file

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>

<body>
<h1>Thanks for joining our email list</h1>
<p>Here is the information that you entered:</p>
<label>Email:</label>
<span>${user.email}</span><br>
<label>First Name:</label>
<span>${user.firstName}</span><br>
<label>Last Name:</label>
<span>${user.lastName}</span><br>
<p>To enter another email address, click on the Back button in your browser or the Return button shown below.</p>
<form action="" method="get">
    <input type="hidden" name="action" value="join">
    <input type="submit" value="Return">
</form>
</body>
</html>
```

Description

- A JavaServer Page (JSP) consists of special Java tags such as Expression Language (EL) tags that are embedded within HTML code. An EL tag begins with a dollar sign ($).
- 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. In the old days, this often led to a delay the first time a JSP was requested after it was compiled. Today, most servlet/JSP engines precompile JSPs in order to avoid that delay.
Perspective

The primary goal of this chapter has been to show you how to use the MVC pattern to structure a web application. In addition, this chapter has presented the different types of files that you need to work with in a servlet/JSP application. These files include HTML, JSP, CSS, servlet, Java, and XML files. In the next chapter, you’ll learn how to use NetBeans to work with these types of files.

As you review this chapter, you may notice that it doesn’t include the code for the UserDB class. For now, all you need to know is that the insert method of the UserDB class adds the data that’s stored in a User object to the database for the application. Later, in section 3, you’ll learn about two different approaches to working with databases.

Summary

- The Model 1 pattern uses JSPs to handle all of the processing and presentation for the application. It’s generally considered a bad practice.
- The Model 2 pattern, also known as the Model-View-Controller (MVC) pattern, uses business objects to define the model, HTML pages and JSPs to define the view, and servlets to act as the controller.
- The data for the business objects in a web application are stored in data stores like files and databases. This can be referred to as persistent data storage. To work with the data store, you typically use a data access class.
- An HTML file contains the tags that define the content of the web page and a CSS (Cascading Style Sheet) file contains the formatting for the web pages.
- A servlet is a special type of Java class that runs on a server and does the processing for the dynamic web pages of a web application.
- A web.xml file describes how a web application should be configured when it is deployed on a server. As a result, it’s known as the deployment descriptor (DD).
- A JavaBean, or bean, is a Java class that (1) provides a zero-argument constructor, (2) provides get and set methods for all of its private instance variables, and (3) implements the Serializable or Externalizable interface.
- An Enterprise JavaBean (EJB) is similar in some ways to a regular JavaBean. However, EJBs are more complex and difficult to code than JavaBeans and aren’t necessary for most websites.
- Since JavaBeans are just Java classes, they are a type of plain old Java object (POJO).
- A JavaServer Page (JSP) consists of special Java tags such as Expression Language (EL) tags that are embedded within HTML code. An EL tag begins with a dollar sign ($).
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 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!

Mike