

SERVLETS

Objectives

- To explain how a servlet works (§37.2).
- To create/develop/run servlets (§37.3).
- To deploy servlets on application servers such as Tomcat and GlassFish (§37.3).
- To describe the servlets API (§37.4).
- To create simple servlets (§37.5).
- To create and process HTML forms (§37.6).
- To develop servlets to access databases (§37.7).
- To use hidden fields, cookies, and **HttpSession** to track sessions (§37.8).

CHAPTER 37





37.1 Introduction

Java Servlets is the foundation for developing Web applications using Java.

Servlets are Java programs that run on a Web server. They can be used to process client requests or produce dynamic webpages. For example, you can write servlets to generate dynamic webpages that display stock quotes or process client registration forms and store registration data in a database. This chapter introduces the concept of Java servlets. You will learn how to develop Java servlets using NetBeans.



Note

You can develop servlets without using an IDE. However, using an IDE such as NetBeans can greatly simplify the development task. The tool can automatically create the supporting directories and files. We choose NetBeans because it has the best support for Java Web development. You can still use your favorite IDE or no IDE for this chapter.



Note

Servlets are the foundation of Java Web technologies. JSP, JSF, and Java Web services are based on servlets. A good understanding of servlets helps you see the big picture of Java Web technology and learn JSP, JSF, and Web services.



37.2 HTML and Common Gateway Interface

Java servlets are Java programs that function like CGI programs. They are executed upon request from a Web browser.

Java servlets run in the Web environment. To understand Java servlets, let us review HTML and the Common Gateway Interface (CGI).

37.2.1 Static Web Contents

You create webpages using HTML. Your webpages are stored as files on the Web server. The files are usually stored in the /htdocs directory on Unix, as shown in Figure 37.1. A user types a URL for the file from a Web browser. The browser contacts the Web server and requests the file. The server finds the file and returns it to the browser. The browser then displays the file to the user. This works fine for static information that does not change regardless of who requests it or when it is requested. Static information is stored in files. The information in the files can be updated, but at any given time every request for the same document returns exactly the same result.

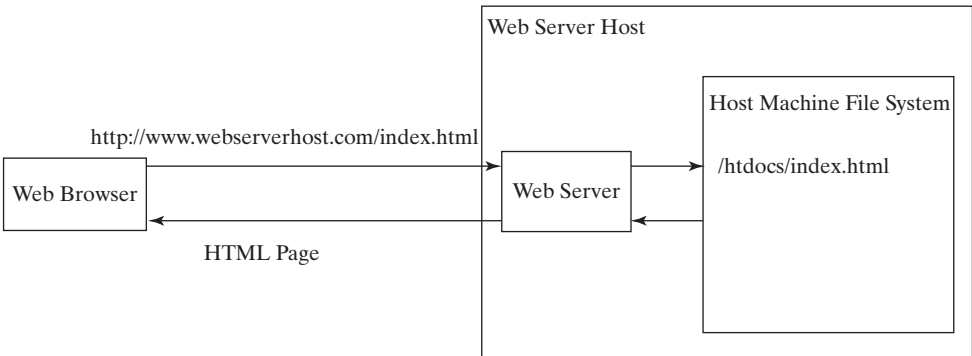


FIGURE 37.1 A Web browser requests a static HTML page from a Web server.

37.2.2 Dynamic Web Contents and Common Gateway Interface

Not all information, however, is static in nature. Stock quotes are updated whenever a trade takes place. Election vote counts are updated constantly on Election Day. Weather reports are frequently updated. The balance in a customer's bank account is updated whenever a transaction takes place. To view up-to-date information on the Web, the HTML pages for displaying this information must be generated dynamically. Dynamic Web pages are generated by Web servers. The Web server needs to run certain programs to process user requests from Web browsers in order to produce a customized response.

The *Common Gateway Interface*, or *CGI*, was proposed to generate dynamic Web content. The interface provides a standard framework for Web servers to interact with external programs, known as *CGI programs*. As shown in Figure 37.2, the Web server receives a request from a Web browser and passes it to the CGI program. The CGI program processes the request and generates a response at runtime. CGI programs can be written in any language, but the *Perl* language is the most popular choice. CGI programs are typically stored in the `/cgi-bin` directory. Here is a pseudocode example of a CGI program for displaying a customer's bank account balance:

1. Obtain account ID and password.
2. Verify account ID and password. If it fails, generate an HTML page to report incorrect account ID and password, and exit.
3. Retrieve account balance from the database; generate an HTML page to display the account ID and balance.

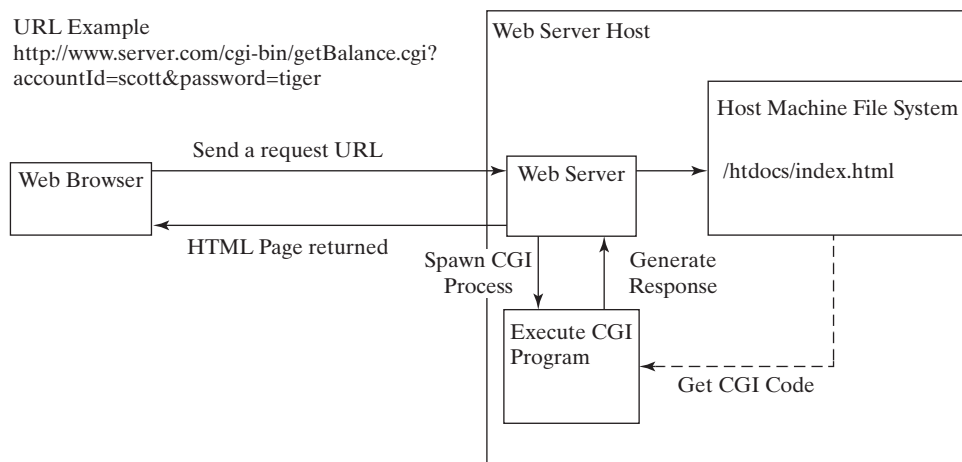


FIGURE 37.2 A Web browser requests a dynamic HTML page from a Web server.

37.2.3 The GET and POST Methods

The two most common HTTP requests, also known as *methods*, are GET and POST. The Web browser issues a request using a URL or an HTML form to trigger the Web server to execute a CGI program. HTML forms will be introduced in §37.6, “HTML Forms.” When issuing a CGI request directly from a URL, the GET method is used. This URL is known as a *query string*. The URL query string consists of the location of the CGI program, the parameters, and their values. For example, the following URL causes the CGI program `getBalance` to be invoked on the server side:

```

http://www.webserverhost.com/cgi-bin/
getBalance.cgi?accountId=scott+smith&password=tiger
  
```

The `?` symbol separates the program from the parameters. The parameter name and value are associated using the `=` symbol. Parameter pairs are separated using the `&` symbol. The `+` symbol denotes a space character. So, here `accountId` is `scott smith`.

When issuing a request from an HTML form, either a GET method or a POST method can be used. The form explicitly specifies one of these. If the GET method is used, the data in the form are appended to the request string as if it were submitted using a URL. If the POST method is used, the data in the form are packaged as part of the request file. The server program obtains the data by reading the file. The POST method is more secure than the GET method.



Note

The GET and POST methods both send requests to the Web server. The POST method always triggers the execution of the corresponding CGI program. The GET method may not cause the CGI program to be executed, if the previous same request is cached in the Web browser. Web browsers often cache webpages so that the same request can be quickly responded to without contacting the Web server. The browser checks the request sent through the GET method as a URL query string. If the results for the exact same URL are cached on a disk, then the previous webpages for the URL may be displayed. To ensure that a new webpage is always displayed, use the POST method. For example, use a POST method if the request will actually update the database. If your request is not time sensitive, such as finding the address of a student in the database, use the GET method to speed up performance.

37.2.4 From CGI to Java Servlets

CGI provides a relatively simple approach for creating dynamic Web applications that accept a user request, process it on the server side, and return responses to the Web browser. But CGI is very slow when handling a large number of requests simultaneously, because the Web server spawns a process for executing each CGI program. Each process has its own runtime environment that contains and runs the CGI program. It is not difficult to imagine what will happen if many CGI programs were executed simultaneously. System resource would be quickly exhausted, potentially causing the server to crash.

Several new approaches have been developed to remedy the performance problem of CGI programs. Java servlets are one successful technology for this purpose. Java servlets are Java programs that function like CGI programs. They are executed upon request from a Web browser. All servlets run inside a *servlet container*, also referred to as a *servlet server* or a *servlet engine*. A servlet container is a single process that runs in a Java Virtual Machine. The JVM creates a thread to handle each servlet. Java threads have much less overhead than full-blown processes. All the threads share the same memory allocated to the JVM. Since the JVM persists beyond the life cycle of a single servlet execution, servlets can share objects already created in the JVM. For example, if multiple servlets access the same database, they can share the connection object. Servlets are much more efficient than CGI.

Servlets have other benefits that are inherent in Java. As Java programs, they are object oriented, portable, and platform independent. Since you know Java, you can develop servlets immediately with the support of Java API for accessing databases and network resources.



Check
Point

- 37.2.1** What is the common gateway interface?
- 37.2.2** What are the differences between the GET and POST methods in an HTML form?
- 37.2.3** Can you submit a GET request directly from a URL? Can you submit a POST request directly from a URL?
- 37.2.4** What is wrong in the following URL for submitting a GET request to the servlet `FindScore` on host `liang` at port `8084` with parameter `name`?
`http://liang:8084/findScore?name="P Yates"`
- 37.2.5** What are the differences between CGI and servlets?

37.3 Creating and Running Servlets

An IDE such as NetBeans is an effective tool for creating Java servlet.

To run Java servlets, you need a servlet container. Many servlet containers are available for free. Two popular ones are *Tomcat* (developed by Apache, www.apache.org) and *GlassFish* (developed by Sun, glassfish.dev.java.net). Both Tomcat and GlassFish are bundled and integrated with NetBeans 7 (Java EE version). When you run a servlet from NetBeans, Tomcat or GlassFish will be automatically started. You can choose to use either of them, or any other application server. GlassFish has more features than Tomcat and it takes more system resource.



37.3.1 Creating a Servlet

Before our introduction to the servlet API, let us look at a simple example to see how servlets work. A servlet to some extent resembles a JavaFX program. Every Java applet is a subclass of the **Application** class. You need to override appropriate methods in the **Application** class to implement the application. Every servlet is a subclass of the **HttpServlet** class. You need to override appropriate methods in the **HttpServlet** class to implement the servlet. Listing 37.1 is a servlet that generates a response in HTML using the **doGet** method.

LISTING 37.1 FirstServlet.java

```

1  package chapter37;
2
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5
6  public class FirstServlet extends HttpServlet {
7      /** Handle the HTTP GET method.
8       * @param request servlet request
9       * @param response servlet response
10     */
11     protected void doGet(HttpServletRequest request,
12         HttpServletResponse response)
13         throws ServletException, java.io.IOException {
14         response.setContentType("text/html");
15         java.io.PrintWriter out = response.getWriter();
16         // output your page here
17         out.println("<html>");
18         out.println("<head>");
19         out.println("<title>Servlet</title>");
20         out.println("</head>");
21         out.println("<body>");
22         out.println("Hello, Java Servlets");
23         out.println("</body>");
24         out.println("</html>");
25         out.close();
26     }
27 }
```

The **doGet** method (line 11) is invoked when the Web browser issues a request using the GET method. The **doGet** method has two parameters: **request** and **response**. **request** is for obtaining data from the Web browser, and **response** is for sending data back to the browser. Line 14 indicates that data are sent back to the browser as text/html. Line 15 obtains an instance of **PrintWriter** for actually outputting data to the browser.

37.3.2 Creating Servlets in NetBeans

NetBeans is updated frequently. The current version is 8 at the time of this writing. To create a servlet in NetBeans 8, you have to first create a Web project, as follows:

1. Choose **File, New Project** to display the New Project dialog box. Choose **Java Web** in the Categories section and **Web Application** in the Projects section, as shown in Figure 37.3a. Click *Next* to display the New Web Application dialog box, as shown in Figure 37.3b.
2. Enter **liangweb** in the Project Name field and **c:\book** in the Project Location field. Check *Set as Main Project*. Click *Next* to display the dialog box for specifying server and settings, as shown in Figure 37.4.
3. Select **GlassFish Server 4.1** for server and **Java EE 7 Web** for Java EE Version. Click *Finish* to create the Web project, as shown in Figure 37.5.

Now you can create a servlet in the project, as follows:

1. Right-click the **liangweb** node in the project pane to display a context menu. Choose **New, Servlet** to display the New Servlet dialog box, as shown in Figure 37.6.

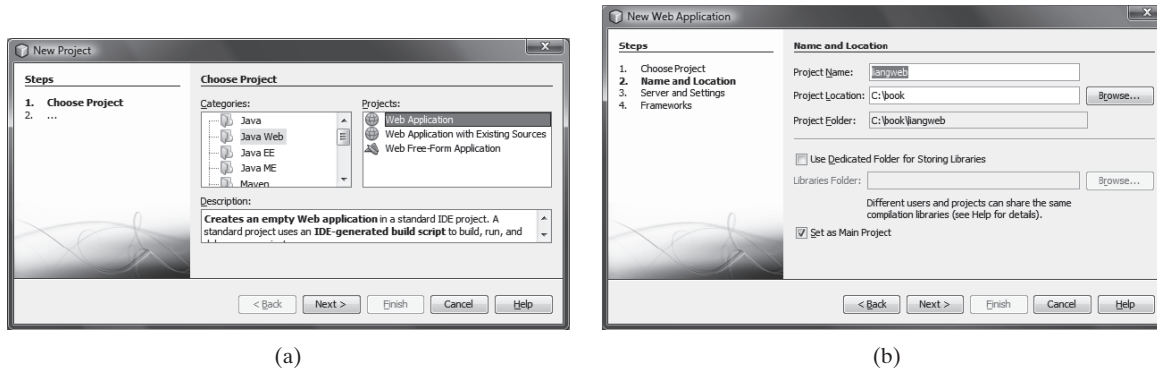


FIGURE 37.3 (a) Choose Web Application to create a Web project. (b) Specify project name and location.

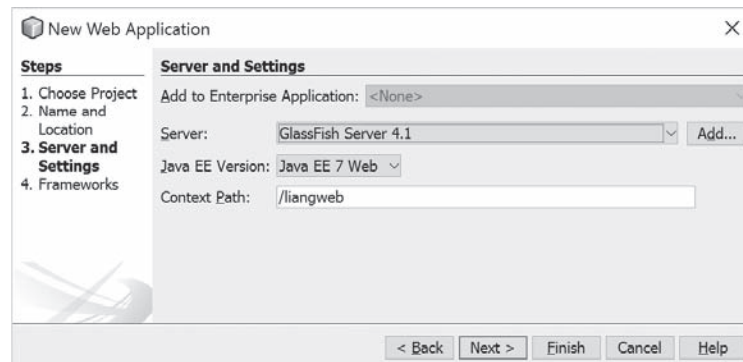


FIGURE 37.4 Choose servers and settings.

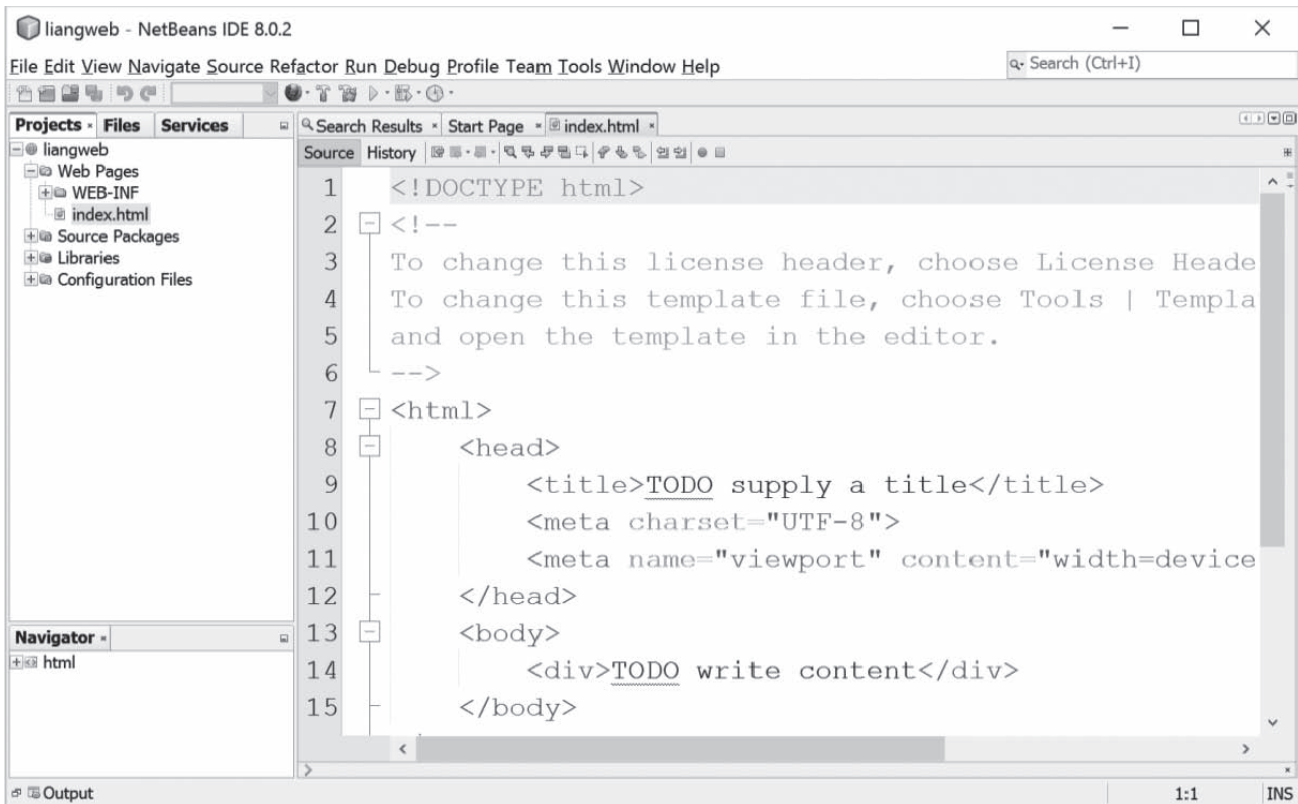


FIGURE 37.5 A new Web project is created.

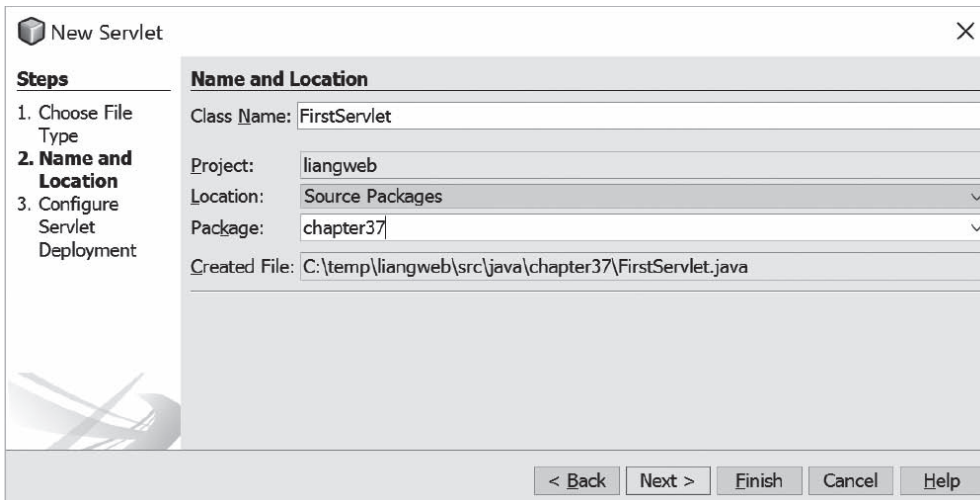
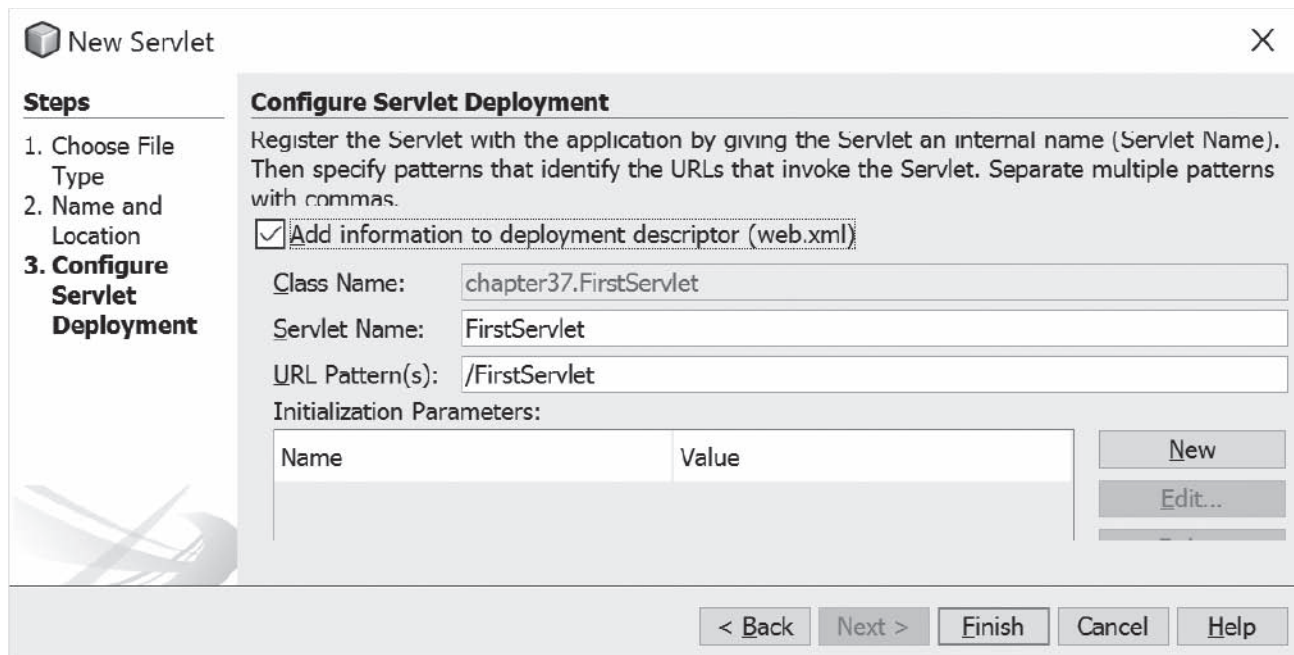


FIGURE 37.6 You can create a servlet in the New Servlet dialog box.

2. Enter **FirstServlet** in the Class Name field and chapter37 in the Package field and click *Next* to display the Configure Servlet Deployment dialog box, as shown in Figure 37.7.
3. Select the checkbox to add the servlet information to web.xml and click *Finish* to create the servlet. A servlet template is now created in the project, as shown in Figure 37.8.

37-8 Chapter 37 Servlets



The "New Servlet" dialog box is shown with the "Configure Servlet Deployment" tab selected. The "Steps" list on the left indicates the current step is "3. Configure Servlet Deployment". The "Configure Servlet Deployment" section contains instructions to register the Servlet with the application by giving it an internal name and specifying URL patterns. A checkbox labeled "Add information to deployment descriptor (web.xml)" is checked. Below this, the "Class Name" is "chapter37.FirstServlet", the "Servlet Name" is "FirstServlet", and the "URL Pattern(s)" is "/FirstServlet". There is a table for "Initialization Parameters" with columns "Name" and "Value", and buttons for "New" and "Edit...". At the bottom are buttons for "< Back", "Next >", "Finish", "Cancel", and "Help".

New Servlet

Steps

1. Choose File Type
2. Name and Location
3. Configure Servlet Deployment

Configure Servlet Deployment

Register the Servlet with the application by giving the Servlet an internal name (Servlet Name). Then specify patterns that identify the URLs that invoke the Servlet. Separate multiple patterns with commas.

☒ Add information to deployment descriptor (web.xml)

Class Name: chapter37.FirstServlet

Servlet Name: FirstServlet

URL Pattern(s): /FirstServlet

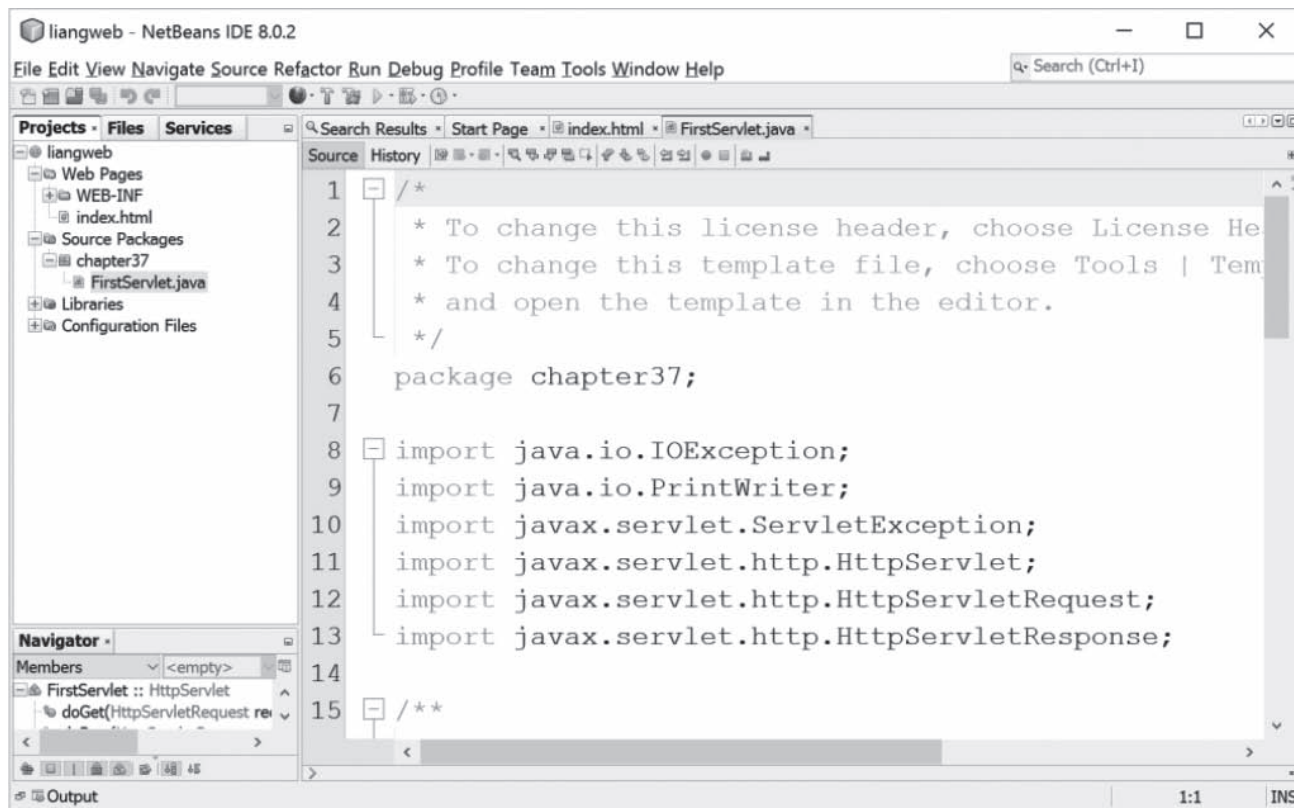
Initialization Parameters:

Name	Value
------	-------

New Edit...

< Back Next > Finish Cancel Help

FIGURE 37.7 You need to click the checkbox to add servlet information to `web.xml`.



The NetBeans IDE 8.0.2 window shows the "liangweb" project. The "Projects" pane on the left shows the project structure, including "Web Pages", "WEB-INF", "index.html", "Source Packages", "chapter37", "FirstServlet.java", "Libraries", and "Configuration Files". The "Navigator" pane shows the "FirstServlet" class with the "doGet" method selected. The "Source" pane shows the code for "FirstServlet.java".

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates in the IDE window.
4   * and open the template in the editor.
5   */
6  package chapter37;
7
8  import java.io.IOException;
9  import java.io.PrintWriter;
10 import javax.servlet.ServletException;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 /**
```

FIGURE 37.8 A new servlet class is created in the project.

4. Replace the code in the content pane for the servlet using the code in Listing 37.1.
5. Right-click **liangweb** node in the Project pane to display a context menu and choose **Run** to launch the Web server. In the Web browser, enter <http://localhost:8084/liangweb/FirstServlet> in the URL. You will now see the servlet result displayed, as shown in Figure 37.9.

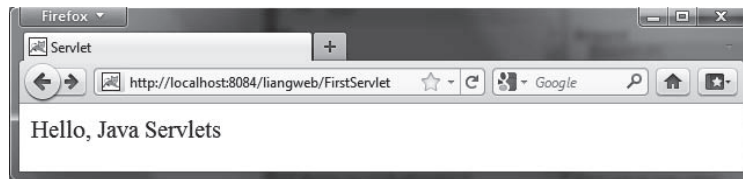


FIGURE 37.9 Servlet result is displayed in a Web browser.



Note

If the servlet is not displayed in the browser, do the following: 1. Make sure that you have added the servlet in the `xml.web` file. 2. Right-click **liangweb** and choose *Clean and Build*. 3. Right-click **liangweb** and choose *Run*. Reenter <http://localhost:8084/liangweb/FirstServlet> in the URL. If still not working, exit NetBeans and restart it.



Note

Depending on the server setup, you may have a port number other than **8084**.



Tip

You can deploy a Web application using a Web archive file (WAR) to a Web application server (e.g., Tomcat). To create a WAR file for the **liangweb** project, right-click **liangweb** and choose **Build Project**. You can now locate **liangweb.war** in the `c:\book\liangweb\dist` folder. To deploy on Tomcat, simply place **liangweb.war** into the `webapps` directory. When Tomcat starts, the `.war` file will be automatically installed.



Note

If you wish to use NetBeans as the development tool and Tomcat as the deployment server, please see Supplement V.E, “Tomcat Tutorial.”

- 37.3.1** Can you display an HTML file (e.g. `c:\test.html`) by typing the complete file name in the Address field of Internet Explorer? Can you run a servlet by simply typing the servlet class file name?
- 37.3.2** How do you create a Web project in NetBeans?
- 37.3.3** How do you create a servlet in NetBeans?
- 37.3.4** How do you run a servlet in NetBeans?
- 37.3.5** When you run a servlet from NetBeans, what is the port number by default? What happens if the port number is already in use?
- 37.3.6** What is the `.war` file? How do you obtain a `.war` file for a Web project in NetBeans?



37.4 The Servlet API

The **Servlet** interface defines the methods **init**, **service**, and **destroy** to managing the life-cycle of a servlet.



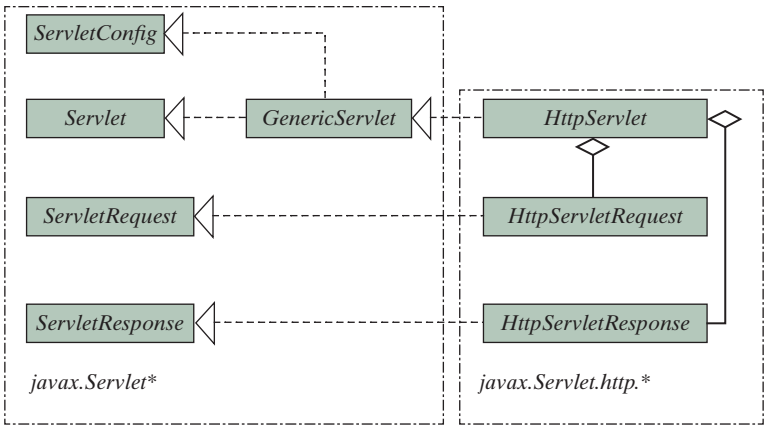


FIGURE 37.10 The servlet API contains interfaces and classes that you use to develop and run servlets.

You have to know the servlet API in order to understand the source code in Listing 37.1, in `FirstServlet.java`. The servlet API provides the interfaces and classes that support servlets. These interfaces and classes are grouped into two packages, `javax.servlet` and `javax.servlet.http`, as shown in Figure 37.10. The `javax.servlet` package provides basic interfaces, and the `javax.servlet.http` package provides classes and interfaces derived from them, which provide specific means for servicing HTTP requests.

37.4.1 The `Servlet` Interface

The `javax.servlet.Servlet` interface defines the methods that all servlets must implement. The methods are listed below:

```
/** Invoked for every servlet constructed */
public void init() throws ServletException;

/** Invoked to respond to incoming requests */
public void service(ServletRequest request, ServletResponse response)
    throws ServletException, IOException;

/** Invoked to release resource by the servlet */
public void destroy();
```

The `init`, `service`, and `destroy` methods are known as *life-cycle methods* and are called in the following sequence (see Figure 37.11):

1. The `init` method is called when the servlet is first created and is not called again as long as the servlet is not destroyed. This resembles an applet's `init` method, which is invoked after the applet is created and is not invoked again as long as the applet is not destroyed.

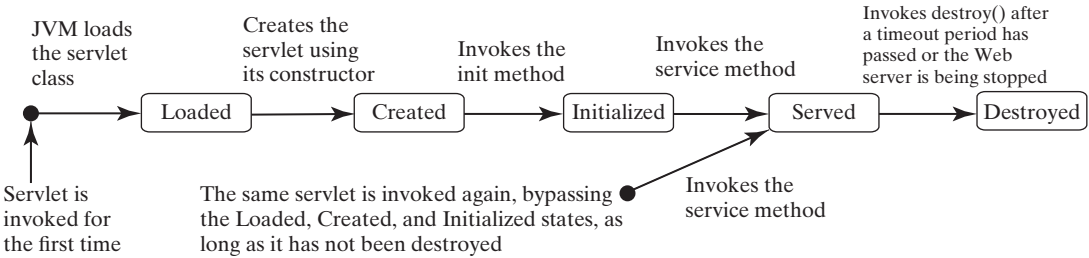


FIGURE 37.11 The JVM uses the `init`, `service`, and `destroy` methods to control the servlet.

2. The **service** method is invoked each time the server receives a request for the servlet. The server spawns a new thread and invokes **service**.
3. The **destroy** method is invoked after a timeout period has passed or as the Web server is terminated. This method releases resources for the servlet.

37.4.2 The **GenericServlet** Class, **ServletConfig** Interface, and **HttpServlet** Class

The **javax.servlet.GenericServlet** class defines a generic, protocol-independent servlet. It implements **javax.servlet.Servlet** and **javax.servlet.ServletConfig**. **ServletConfig** is an interface that defines four methods (**getInitParameter**, **getInitParameterNames**, **getServletContext**, and **getServletName**) for obtaining information from a Web server during initialization. All the methods in **Servlet** and **ServletConfig** are implemented in **GenericServlet** except **service**. Therefore, **GenericServlet** is an abstract class.

The **javax.servlet.http.HttpServlet** class defines a servlet for the HTTP protocol. It extends **GenericServlet** and implements the **service** method. The **service** method is implemented as a dispatcher of HTTP requests. The HTTP requests are processed in the following methods:

- **doGet** is invoked to respond to a GET request.
- **doPost** is invoked to respond to a POST request.
- **doDelete** is invoked to respond to a DELETE request. Such a request is normally used to delete a file on the server.
- **doPut** is invoked to respond to a PUT request. Such a request is normally used to send a file to the server.
- **doOptions** is invoked to respond to an OPTIONS request. This returns information about the server, such as which HTTP methods it supports.
- **doTrace** is invoked to respond to a TRACE request. Such a request is normally used for debugging. This method returns an HTML page that contains appropriate trace information.

All these methods use the following signature:

```
protected void doXxx(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, java.io.IOException
```

The **HttpServlet** class provides default implementation for these methods. You need to override **doGet**, **doPost**, **doDelete**, and **doPut** if you want the servlet to process a GET, POST, DELETE, or PUT request. By default, nothing will be done. Normally, you should not override the **doOptions** method unless the servlet implements new HTTP methods beyond those implemented by HTTP 1.1. Nor is there any need to override the **doTrace** method.



Note

GET and POST requests are often used, whereas DELETE, PUT, OPTIONS, and TRACE are not. For more information about these requests, please refer to the HTTP 1.1 specification from www.cis.ohio-state.edu/htbin/rfc/rfc2068.html.



Note

Although the methods in **HttpServlet** are all nonabstract, **HttpServlet** is defined as an abstract class. Thus you cannot create a servlet directly from **HttpServlet**. Instead you have to define your servlet by extending **HttpServlet**.

The relationship of these interfaces and classes is shown in Figure 37.12.

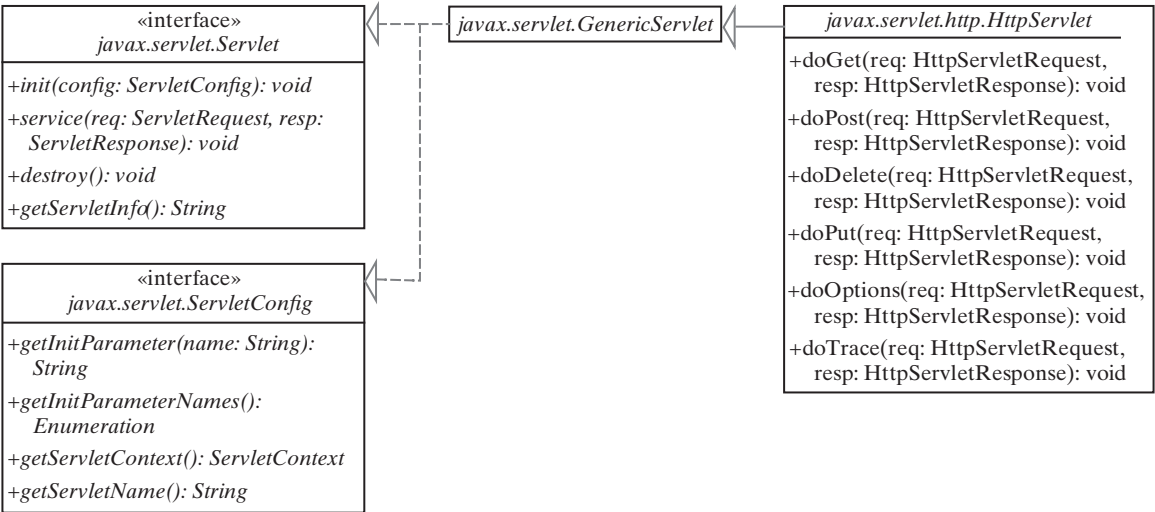


FIGURE 37.12 `HttpServlet` inherits abstract class `GenericServlet`, which implements interfaces `Servlet` and `ServletConfig`.

37.4.3 The `ServletRequest` Interface and `HttpServletRequest` Interface

Every `doXxx` method in the `HttpServlet` class has a parameter of the `HttpServletRequest` type, which is an object that contains HTTP request information, including parameter name and values, attributes, and an input stream. `HttpServletRequest` is a subinterface of `ServletRequest`. `ServletRequest` defines a more general interface to provide information for all kinds of clients. The frequently used methods in these two interfaces are shown in Figure 37.13.

37.4.4 The `ServletResponse` Interface and `HttpServletResponse` Interface

Every `doXxx` method in the `HttpServlet` class has a parameter of the `HttpServletResponse` type, which is an object that assists a servlet in sending a response to the client. `HttpServletResponse` is a subinterface of `ServletResponse`. `ServletResponse` defines a more general interface for sending output to the client.

The frequently used methods in these two interfaces are shown in Figure 37.14.



37.4.1 Describe the life cycle of a servlet.

37.4.2 Suppose you started the Web server, ran the following servlet twice by issuing an appropriate URL from the Web browser, and finally stopped Tomcat. What was displayed on the console when the servlet was first invoked? What was displayed on the console when the servlet was invoked for the second time? What was displayed on the console when Tomcat was shut down?

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Test extends HttpServlet {
    public Test() {
        System.out.println("Constructor called");
    }

    /** Initialize variables */
    public void init() throws ServletException {
```

```
        System.out.println("init called");
    }

    /** Process the HTTP Get request */
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        System.out.println("doGet called");
    }

    /** Clean up resources */
    public void destroy() {
        System.out.println("destroy called");
    }
}
```

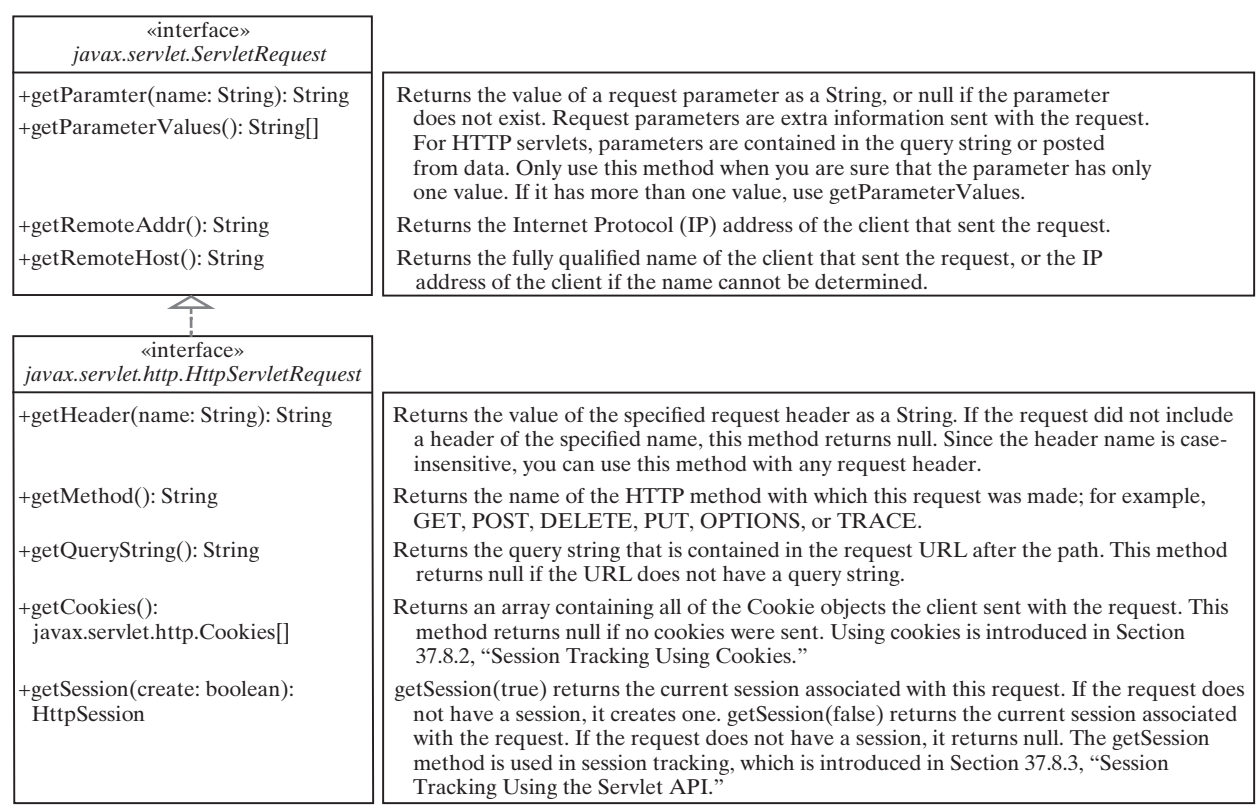


FIGURE 37.13 `HttpServletRequest` is a subinterface of `ServletRequest`.

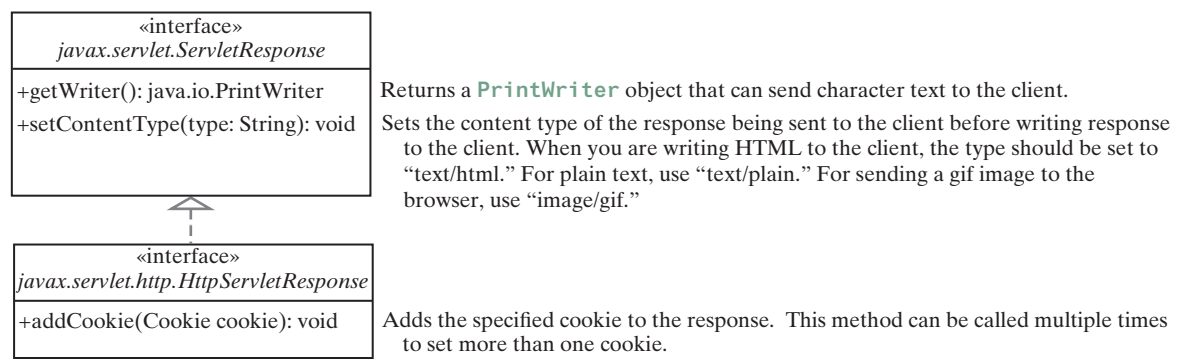


FIGURE 37.14 `HttpServletResponse` is a subinterface of `ServletResponse`.



37.5 Creating Servlets

*You can define a servlet class by extending the **HttpServlet** class and implementing the **doGet** and **doPost** methods.*

Servlets are the opposite of Java applets. Java applets run from a Web browser on the client side. To write Java programs, you define classes. To write a Java applet, you define a class that extends the **Applet** class. The Web browser runs and controls the execution of the applet through the methods defined in the **Applet** class. Similarly, to write a Java servlet, you define a class that extends the **HttpServlet** class. The servlet container runs and controls the execution of the servlet through the methods defined in the **HttpServlet** class. Like a Java applet, a servlet does not have a **main** method. A servlet depends on the servlet engine to call the methods. Every servlet has a structure like the one shown below:

```
package chapter37;

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

public class MyServlet extends HttpServlet {
    /** Called by the servlet engine to initialize servlet */
    public void init() throws ServletException {
        ...
    }

    /** Process the HTTP Get request */
    public void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        ...
    }

    /** Process the HTTP Post request */
    public void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        ...
    }

    /** Called by the servlet engine to release resource */
    public void destroy() {
        ...
    }

    // Other methods if necessary
}
```

The servlet engine controls the servlets using **init**, **doGet**, **doPost**, **destroy**, and other methods. By default, the **doGet** and **doPost** methods do nothing. To handle a GET request, you need to override the **doGet** method; to handle a POST request, you need to override the **doPost** method.

Listing 37.2 gives a simple Java servlet that generates a dynamic webpage for displaying the current time, as shown in Figure 37.15.

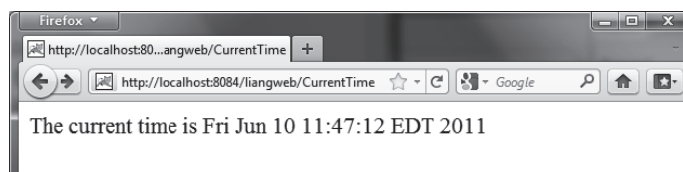


FIGURE 37.15 Servlet **CurrentTime** displays the current time.

LISTING 37.2 CurrentTime.java

```

1 package chapter37;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6
7 public class CurrentTime extends HttpServlet {
8     /** Process the HTTP Get request */
9     public void doGet(HttpServletRequest request, HttpServletResponse
10         response) throws ServletException, IOException {
11         response.setContentType("text/html");
12         PrintWriter out = response.getWriter();
13         out.println("<p>The current time is " + new java.util.Date());
14         out.close(); // Close stream
15     }
16 }

```

The `HttpServlet` class has a `doGet` method. The `doGet` method is invoked when the browser issues a request to the servlet using the GET method. Your servlet class should override the `doGet` method to respond to the GET request. In this case, you write the code to display the current time.

Servlets return responses to the browser through an `HttpServletResponse` object. Since the `setContentType("text/html")` method sets the MIME type to “text/html,” the browser will display the response in HTML. The `getWriter` method returns a `PrintWriter` object for sending HTML back to the client.

**Note**

The URL query string uses the GET method to issue a request to the servlet. The current time may not be current if the webpage for displaying the current time is cached. To ensure that a new current time is displayed, refresh the page in the browser. In the next example, you will write a new servlet that uses the POST method to obtain the current time.

37.6 HTML Forms

HTML forms are used to collect and submit data from a client to a Web server.

HTML forms enable you to submit data to the Web server in a convenient form. As shown in Figure 37.16, the form can contain text fields, text area, check boxes, combo boxes, lists, radio buttons, and buttons.



FIGURE 37.16 An HTML form may contain text fields, radio buttons, combo boxes, lists, check boxes, text areas, and buttons.

The HTML code for creating the form in Figure 37.16 is given in Listing 37.3. (If you are unfamiliar with HTML, please see Supplement V.A, “HTML and XHTML Tutorial.”)

LISTING 37.3 StudentRegistrationForm.html

```

1  <!--An HTML Form Demo -->
2  <html>
3    <head>
4      <title>Student Registration Form</title>
5    </head>
6    <body>
7      <h3>Student Registration Form</h3>
8
9      <form action = "GetParameters"
10         method = "get">
11        <!-- Name text fields -->
12        <p><label>Last Name</label>
13        <input type = "text" name = "lastName" size = "20" />
14        <label>First Name</label>
15        <input type = "text" name = "firstName" size = "20" />
16        <label>MI</label>
17        <input type = "text" name = "mi" size = "1" /></p>
18
19        <!-- Gender radio buttons -->
20        <p><label>Gender:</label>
21        <input type = "radio" name = "gender" value = "M" checked />
22        Male
23        <input type = "radio" name = "gender" value = "F" /> Female</p>
24
25        <!-- Major combo box -->
26        <p><label>Major</label>
27        <select name = "major" size = "1">
28          <option value = "CS">Computer Science</option>
29          <option value = "Math">Mathematics</option>
30          <option>English</option>
31          <option>Chinese</option>
32        </select>
33
34        <!-- Minor list -->
35        <label>Minor</label>
36        <select name = "minor" size = "2" multiple>
37          <option>Computer Science</option>
38          <option>Mathematics</option>
39          <option>English</option>
40          <option>Chinese</option>
41        </select></p>
42
43        <!-- Hobby check boxes -->
44        <p><label>Hobby:</label>
45        <input type = "checkbox" name = "tennis" /> Tennis
46        <input type = "checkbox" name = "golf" /> Golf
47        <input type = "checkbox" name = "pingPong" checked /> Ping Pong
48      </p>
49
50      <!-- Remark text area -->
51      <p>Remarks:</p>
52      <p><textarea name = "remarks" rows = "3" cols = "56">
53        </textarea></p>
54
```

```

55     <!-- Submit and Reset buttons -->
56     <p><input type = "submit" value = "Submit" />
57     <input type = "reset" value = "Reset" /></p>
58 </form>
59 </body>
60 </html>

```

The following HTML tags are used to construct HTML forms:

- **<form> . . . </form>** defines a form body. The attributes for the **<form>** tag are **action** and **method**. The **action** attribute specifies the server program to be executed on the Web server when the form is submitted. The **method** attribute is either **get** or **post**.
- **<label> . . . </label>** simply defines a label.
- **<input>** defines an input field. The attributes for this tag are **type**, **name**, **value**, **checked**, **size**, and **maxlength**. The **type** attribute specifies the input type. Possible types are **text** for a one-line text field, **radio** for a radio button, and **checkbox** for a check box. The **name** attribute gives a formal name for the attribute. This **name** attribute is used by the servlet program to retrieve its associated value. The names of the radio buttons in a group must be identical. The **value** attribute specifies a default value for a text field and text area. The **checked** attribute indicates whether a radio button or a check box is initially checked. The **size** attribute specifies the size of a text field, and the **maxlength** attribute specifies the maximum length of a text field.
- **<select> . . . </select>** defines a combo box or a list. The attributes for this tag are **name**, **size**, and **multiple**. The **size** attribute specifies the number of rows visible in the list. The **multiple** attribute specifies that multiple values can be selected from a list. Set **size** to **1** and do not use a **multiple** for a combo box.
- **<option> . . . </option>** defines a selection list within a **<select> . . . </select>** tag. This tag may be used with the **value** attribute to specify a value for the selected option (e.g., **<option value = "CS">Computer Science**). If no value is specified, the selected option is the value.
- **<textarea> . . . </textarea>** defines a text area. The attributes are **name**, **rows**, and **cols**. The **rows** and **cols** attributes specify the number of rows and columns in a text area.



Note

You can create the HTML file from NetBeans. Right-click **liangweb** and choose *New, HTML*, to display the New HTML File dialog box. Enter **StudentRegistrationForm** as the file name and click *Finish* to create the file.

37.6.1 Obtaining Parameter Values from HTML Forms

To demonstrate how to obtain parameter values from an HTML form, Listing 37.4 creates a servlet to obtain all the parameter values from the preceding student registration form in Figure 37.16 and display their values, as shown in Figure 37.17.

LISTING 37.4 GetParameters.java

```

1 package chapter37;
2
3 import javax.servlet.*;

```

37-18 Chapter 37 Servlets

```

4  import javax.servlet.http.*;
5  import java.io.*;
6
7  public class GetParameters extends HttpServlet {
8      /** Process the HTTP Post request */
9      public void doGet(HttpServletRequest request, HttpServletResponse
10         response) throws ServletException, IOException {
11         response.setContentType("text/html");
12         PrintWriter out = response.getWriter();
13
14         // Obtain parameters from the client
15         String lastName = request.getParameter("lastName");
16         String firstName = request.getParameter("firstName");
17         String mi = request.getParameter("mi");
18         String gender = request.getParameter("gender");
19         String major = request.getParameter("major");
20         String[] minors = request.getParameterValues("minor");
21         String tennis = request.getParameter("tennis");
22         String golf = request.getParameter("golf");
23         String pingPong = request.getParameter("pingPong");
24         String remarks = request.getParameter("remarks");
25
26         out.println("Last Name: <b>" + lastName + "</b> First Name: <b>"
27             + firstName + "</b> MI: <b>" + mi + "</b><br>");
28         out.println("Gender: <b>" + gender + "</b><br>");
29         out.println("Major: <b>" + major + "</b> Minor: <b>");
30
31         if (minors != null)
32             for (int i = 0; i < minors.length; i++)
33                 out.println(minors[i] + " ");
34
35         out.println("</b><br> Tennis: <b>" + tennis + "</b> Golf: <b>" +
36             golf + "</b> PingPong: <b>" + pingPong + "</b><br>");
37         out.println("Remarks: <b>" + remarks + "</b>");
38         out.close(); // Close stream
39     }
40 }

```

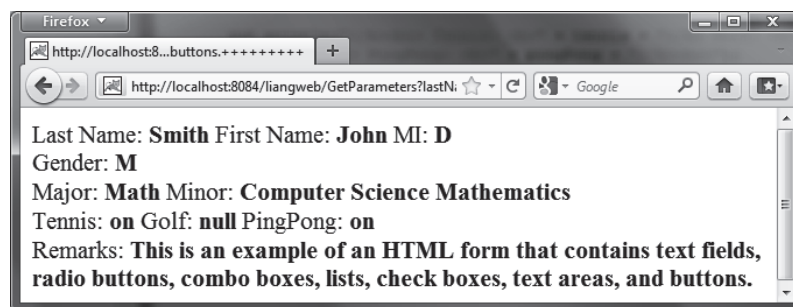


FIGURE 37.17 The servlet displays the parameter values entered in Figure 37.16.

The HTML form is already created in `StudentRegistrationForm.html` and displayed in Figure 37.16. Since the action for the form is `GetParameters`, clicking the *Submit* button invokes the `GetParameters` servlet.

Each GUI component in the form has a name attribute. The servlet uses the name attribute in the `getParameter(attributeName)` method to obtain the parameter value as a string. In case of a list with multiple values, use the `getParameterValues(attributeName)` method to return the parameter values in an array of strings (line 20).

You may optionally specify the `value` attribute in a text field, text area, combo box, list, check box, or radio button in an HTML form. For text field and text area, the `value` attribute specifies a default value to be displayed in the text field and text area. The user can type in new values to replace it. For combo box, list, check box, and radio button, the `value` attribute specifies the parameter value to be returned from the `getParameter` and `getParameterValues` methods. If the `value` attribute is not specified for a combo box or a list, it returns the selected string from the combo box or the list. If the `value` attribute is not specified for a radio button or a check box, it returns string `on` for a checked radio button or a checked check box, and returns `null` for an unchecked check box.



Note

If an attribute does not exist, the `getParameter(attributeName)` method returns `null`. If an empty value of the parameter is passed to the servlet, the `getParameter(attributeName)` method returns a string with an empty value. In this case, the length of the string is `0`.

37.6.2 Obtaining Current Time Based on Locale and Time Zone

This example creates a servlet that processes the GET and POST requests. The GET request generates a form that contains a combo box for locale and a combo box for time zone, as shown in Figure 37.18a. The user can choose a locale and a time zone from this form to submit a POST request to obtain the current time based on the locale and time zone, as shown in Figure 37.18b.

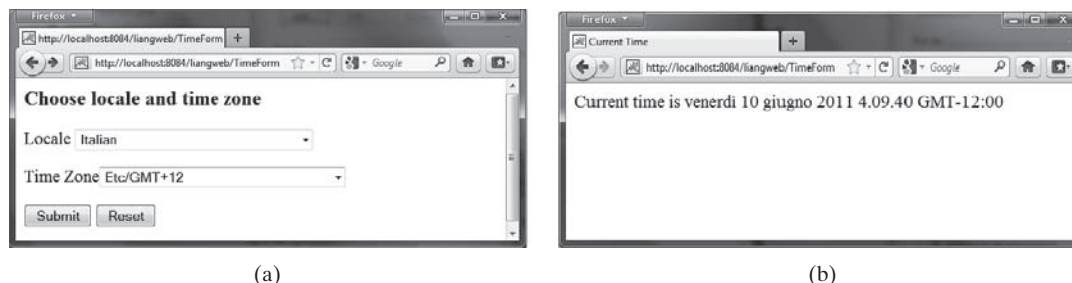


FIGURE 37.18 The GET method in the `TimeForm` servlet displays a form in (a), and the POST method in the `TimeForm` servlet displays the time based on locale and time zone in (b).

Listing 37.5 gives the servlet.

LISTING 37.5 `TimeForm.java`

```
1 package chapter37;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6 import java.util.*;
7 import java.text.*;
8
9 public class TimeForm extends HttpServlet {
10     private static final String CONTENT_TYPE = "text/html";
11     private Locale[] allLocale = Locale.getAvailableLocales();
12     private String[] allTimeZone = TimeZone.getAvailableIDs();
13
14     /** Process the HTTP Get request */
15     public void doGet(HttpServletRequest request, HttpServletResponse
```

```

16         response) throws ServletException, IOException {
17     response.setContentType(CONTENT_TYPE);
18     PrintWriter out = response.getWriter();
19     out.println("<h3>Choose locale and time zone</h3>");
20     out.println("<form method=\"post\" action=\"" +
21         "TimeForm">");
22     out.println("Locale <select size=\"1\" name=\"locale\">");
23
24     // Fill in all locales
25     for (int i = 0; i < allLocale.length; i++) {
26         out.println("<option value=\"" + i + "\">" +
27             allLocale[i].getDisplayNames() + "</option>");
28     }
29     out.println("</select>");
30
31     // Fill in all time zones
32     out.println("<p>Time Zone<select size=\"1\" name=\"timezone\">");
33     for (int i = 0; i < allTimeZone.length; i++) {
34         out.println("<option value=\"" + allTimeZone[i] + "\">" +
35             allTimeZone[i] + "</option>");
36     }
37     out.println("</select>");
38
39     out.println("<p><input type=\"submit\" value=\"Submit\" >");
40     out.println("<input type=\"reset\" value=\"Reset\"></p>");
41     out.println("</form>");
42     out.close(); // Close stream
43 }
44
45 /** Process the HTTP Post request */
46 public void doPost(HttpServletRequest request, HttpServletResponse
47     response) throws ServletException, IOException {
48     response.setContentType(CONTENT_TYPE);
49     PrintWriter out = response.getWriter();
50     out.println("<html>");
51     int localeIndex = Integer.parseInt(
52         request.getParameter("locale"));
53     String timeZoneID = request.getParameter("timezone");
54     out.println("<head><title>Current Time</title></head>");
55     out.println("<body>");
56     Calendar calendar =
57         new GregorianCalendar(allLocale[localeIndex]);
58     TimeZone timeZone = TimeZone.getTimeZone(timeZoneID);
59     DateFormat dateFormat = DateFormat.getDateInstance(
60         DateFormat.FULL, DateFormat.FULL, allLocale[localeIndex]);
61     dateFormat.setTimeZone(timeZone);
62     out.println("Current time is " +
63         dateFormat.format(calendar.getTime()) + "</p>");
64     out.println("</body></html>");
65     out.close(); // Close stream
66 }
67 }

```

When you run this servlet, the servlet `TimeForm`'s `doGet` method is invoked to generate the time form dynamically. The method of the form is POST, and the action invokes the same servlet, `TimeForm`. When the form is submitted to the server, the `doPost` method is invoked to process the request.

The variables `allLocale` and `allTimeZone` (lines 11–12), respectively, hold all the available locales and time zone IDs. The names of the locales are displayed in the locale list. The values for the locales are the indexes of the locales in the array `allLocale`. The time zone IDs

are strings. They are displayed in the time zone list. They are also the values for the list. The indexes of the locale and the time zone are passed to the servlet as parameters. The `doPost` method obtains the values of the parameters (lines 51–53) and finds the current time based on the locale and time zone.



Note

If you choose an Asian locale (e.g., Chinese, Korean, or Japanese), the time will not be displayed properly, because the default character encoding is UTF-8. To fix this problem, insert the following statement in line 48 to set an international character encoding:

```
response.setCharacterEncoding("GB18030");
```

For information on encoding, see Section 36.6.6, “Character Encoding.”

37.7 Database Programming in Servlets

Servlets can access and manipulate databases using JDBC.

Many dynamic Web applications use databases to store and manage data. Servlets can connect to any relational database via JDBC. In Chapter 34, Java Database Programming, you learned how to create Java programs to access and manipulate relational databases via JDBC. Connecting a servlet to a database is no different from connecting a Java application or applet to a database. If you know Java servlets and JDBC, you can combine them to develop interesting and practical Web-based interactive projects.

To demonstrate connecting to a database from a servlet, let us create a servlet that processes a registration form. The client enters data in an HTML form and submits the form to the server, as shown in Figure 37.19. The result of the submission is shown in Figure 37.20. The server collects the data from the form and stores them in a database.



FIGURE 37.19 The HTML form enables the user to enter student information.

FIGURE 37.20 The servlet processes the form and stores data in a database.

The registration data are stored in an **Address** table consisting of the following fields: **firstName**, **mi**, **lastName**, **street**, **city**, **state**, **zip**, **telephone**, and **email**, defined in the following statement:

```
create table Address (
    firstname varchar(25),
    mi char(1),
    lastname varchar(25),
    street varchar(40),
    city varchar(20),
    state varchar(2),
    zip varchar(5),
    telephone varchar(10),
    email varchar(30)
)
```

MySQL, Oracle, and Access were used in Chapter 34. You can use any relational database. If the servlet uses a database driver other than the JDBC-ODBC driver (e.g., the MySQL JDBC driver and the Oracle JDBC driver), you need to add the JDBC driver (e.g., mysqljdbc.jar for MySQL and ojdbc6.jar for Oracle) into the Libraries node in the project.

Create an HTML file named **SimpleRegistration.html** in Listing 37.6 for collecting the data and sending them to the database using the post method.

LISTING 37.6 SimpleRegistration.html

```
1  <!-- SimpleRegistration.html -->
2  <html>
3    <head>
4      <title>Simple Registration without Confirmation</title>
5    </head>
6    <body>
7      Please register to your instructor's student address book.
8
9      <form method = "post" action = "SimpleRegistration">
10         <p>Last Name <font color = "#FF0000">*</font>
11           <input type = "text" name = "lastName">&nbsp;
12           First Name <font color = "#FF0000">*</font>
13           <input type = "text" name = "firstName">&nbsp;
14           MI <input type = "text" name = "mi" size = "3">
15         </p>
16         <p>Telephone
17           <input type = "text" name = "telephone" size = "20">&nbsp;
18           Email
19           <input type = "text" name = "email" size = "28">&nbsp;
20         </p>
21         <p>Street <input type = "text" name = "street" size = "50">
22         </p>
23         <p>City <input type = "text" name = "city" size = "23">&nbsp;
24           State
25           <select size = "1" name = "state">
26             <option value = "GA">Georgia-GA</option>
27             <option value = "OK">Oklahoma-OK</option>
28             <option value = "IN">Indiana-IN</option>
29           </select>&nbsp;
30           Zip <input type = "text" name = "zip" size = "9">
31         </p>
32         <p><input type = "submit" name = "Submit" value = "Submit">
33           <input type = "reset" value = "Reset">
34         </p>
35       </form>
```

```

36     <p><font color = "#FF0000">* required fields</font></p>
37     </body>
38 </html>

```

Create the servlet named `SimpleRegistration` in Listing 37.7.

LISTING 37.7 SimpleRegistration.java

```

1  package chapter37;
2
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import java.io.*;
6  import java.sql.*;
7
8  public class SimpleRegistration extends HttpServlet {
9      // Use a prepared statement to store a student into the database
10     private PreparedStatement pstmt;
11
12     /** Initialize variables */
13     public void init() throws ServletException {
14         initializeJdbc();
15     }
16
17     /** Process the HTTP Post request */
18     public void doPost(HttpServletRequest request, HttpServletResponse
19         response) throws ServletException, IOException {
20         response.setContentType("text/html");
21         PrintWriter out = response.getWriter();
22
23         // Obtain parameters from the client
24         String lastName = request.getParameter("lastName");
25         String firstName = request.getParameter("firstName");
26         String mi = request.getParameter("mi");
27         String phone = request.getParameter("telephone");
28         String email = request.getParameter("email");
29         String address = request.getParameter("street");
30         String city = request.getParameter("city");
31         String state = request.getParameter("state");
32         String zip = request.getParameter("zip");
33
34         try {
35             if (lastName.length() == 0 || firstName.length() == 0) {
36                 out.println("Last Name and First Name are required");
37             }
38             else {
39                 storeStudent(lastName, firstName, mi, phone, email,
40                     address, city, state, zip);
41
42                 out.println(firstName + " " + lastName +
43                     " is now registered in the database");
44             }
45         }
46         catch (Exception ex) {
47             out.println("Error: " + ex.getMessage());
48         }
49         finally {
50             out.close(); // Close stream
51         }
52     }
53

```

```

54  /** Initialize database connection */
55  private void initializeJdbc() {
56      try {
57          // Load the JDBC driver
58          Class.forName("com.mysql.jdbc.Driver");
59          System.out.println("Driver loaded");
60
61          // Establish a connection
62          Connection conn = DriverManager.getConnection
63              ("jdbc:mysql://localhost/javabook", "scott", "tiger");
64          System.out.println("Database connected");
65
66          // Create a Statement
67          pstmt = conn.prepareStatement("insert into Address " +
68              "(lastName, firstName, mi, telephone, email, street, city, "
69              + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
70      }
71      catch (Exception ex) {
72          ex.printStackTrace();
73      }
74  }
75
76  /** Store a student record to the database */
77  private void storeStudent(String lastName, String firstName,
78      String mi, String phone, String email, String address,
79      String city, String state, String zip) throws SQLException {
80      pstmt.setString(1, lastName);
81      pstmt.setString(2, firstName);
82      pstmt.setString(3, mi);
83      pstmt.setString(4, phone);
84      pstmt.setString(5, email);
85      pstmt.setString(6, address);
86      pstmt.setString(7, city);
87      pstmt.setString(8, state);
88      pstmt.setString(9, zip);
89      pstmt.executeUpdate();
90  }
91  }

```

The `init` method (line 13) is executed once when the servlet starts. After the servlet has started, the servlet can be invoked many times as long as it is alive in the servlet container. Load the driver and connect to the database from the servlet's `init` method (line 14). If a prepared statement or a callable statement is used, it should also be created in the `init` method. In this example, a prepared statement is desirable, because the servlet always uses the same insert statement with different values.

A servlet can connect to any relational database via JDBC. The `initializeJdbc` method in this example connects to a MySQL database (line 58). Once connected, it creates a prepared statement for inserting a student record into the database. MySQL is used in this example; you can replace it with any relational database.

Last name and first name are required fields. If either of them is empty, the servlet sends an error message to the client (lines 35–36). Otherwise, the servlet stores the data in the database using the prepared statement.



37.7.1 What would be displayed if you changed the content type to `html/plain` in Listing 37.2, `CurrentTime.java`?

37.7.2 The statement `out.close()` is used to close the output stream to response. Why isn't this statement enclosed in a try-catch block?

- 37.7.3** What happens when you invoke `request.getParameter(paramName)` if `paramName` does not exist?
- 37.7.4** How do you write a text field, combo box, check box, and text area in an HTML form?
- 37.7.5** How do you retrieve the parameter value for a text field, combo box, list, check box, radio button, and text area from an HTML form?
- 37.7.6** If the servlet uses a database driver other than the JDBC-ODBC driver, where should the driver be placed in NetBeans?

37.8 Session Tracking

You can perform session tracking using hidden values in a form, using cookies, or using `HttpSession`.



Web servers use the Hyper-Text Transport Protocol (HTTP). HTTP is a stateless protocol. An HTTP Web server cannot associate requests from a client, and therefore treats each request independently. This protocol works fine for simple Web browsing, where each request typically results in an HTML file or a text file being sent back to the client. Such simple requests are isolated. However, the requests in interactive Web applications are often related. Consider the two requests in the following scenario:

Request 1: A client sends registration data to the server; the server then returns the data to the user for confirmation.

Request 2: The client confirms the data that was submitted in Request 1.

In Request 2, the data submitted in Request 1 are confirmed. These two requests are related in a session. A *session* can be defined as a series of related interactions between a single client and the Web server over a period of time. Tracking data among requests in a session is known as *session tracking*.

This section introduces three techniques for session tracking: using hidden values, using cookies, and using the session tracking tools from servlet API.

37.8.1 Session Tracking Using Hidden Values

You can track a session by passing data from the servlet to the client as hidden values in a dynamically generated HTML form by including a field like this one:

```
<input type = "hidden" name = "lastName" value = "Smith">
```

The next request will submit the data back to the servlet. The servlet retrieves this hidden value just like any other parameter value, using the `getParameter` method.

Let us use an example to demonstrate using hidden values in a form. The example creates a servlet that processes a registration form. The client submits the form using the GET method, as shown in Figure 37.21. The server collects the data in the form, displays them to the client, and asks the client for confirmation, as shown in Figure 37.22. The client confirms the data by submitting the request with the hidden values using the POST method. Finally, the servlet writes the data to a database.

Create an HTML form named `Registration.html` in Listing 37.8 for collecting the data and sending it to the database using the GET method for confirmation. This file is almost identical to Listing 37.6, **SimpleRegistration.html** except that the action is replaced by **Registration** (line 9).

Firefox - Using Hidden Data for Session Tracking
http://localhost:8084/liangweb/Registration.html

Please register to your instructor's student address book.

Last Name * Johnson First Name * Pete MI T

Telephone 9124541021 Email

Street 4543 River Run Trail

City Savannah State Georgia-GA Zip 31419

Submit Reset

* required fields

FIGURE 37.21 The registration form collects user information.

Firefox - http://localhost:8084/liangweb/Registration?lastName=Johnson
http://localhost:8084/liangweb/Registration?lastName=Johnson

You entered the following data

Last name: Johnson
First name: Pete
MI: T
Telephone: 9124541021
Email:
Address: 4543 River Run Trail
City: Savannah
State: GA
Zip: 31419

Confirm

FIGURE 37.22 The servlet asks the client for confirmation of the input.

LISTING 37.8 Registration.html

```
1 <!-- Registration.html -->
2 <html>
3   <head>
4     <title>Using Hidden Data for Session Tracking</title>
5   </head>
6   <body>
7     Please register to your instructor's student address book.
8
9     <form method = "get" action = "Registration">
10      <p>Last Name <font color = "#FF0000">*</font>
11      <input type = "text" name = "lastName"> &nbsp;
12      First Name <font color = "#FF0000">*</font>
13      <input type = "text" name = "firstName"> &nbsp;
14      MI <input type = "text" name = "mi" size = "3">
15    </p>
16    <p>Telephone
17      <input type = "text" name = "telephone" size = "20"> &nbsp;
18      Email
```



```

19     <input type = "text" name = "email" size = "28"> &nbsp;
20 </p>
21 <p>Street <input type = "text" name = "street" size = "50">
22 </p>
23 <p>City <input type = "text" name = "city" size = "23"> &nbsp;
24     State
25     <select size = "1" name = "state">
26         <option value = "GA">Georgia-GA</option>
27         <option value = "OK">Oklahoma-OK</option>
28         <option value = "IN">Indiana-IN</option>
29     </select> &nbsp;
30     Zip <input type = "text" name = "zip" size = "9">
31 </p>
32 <p><input type = "submit" name = "Submit" value = "Submit">
33     <input type = "reset" value = "Reset">
34 </p>
35 </form>
36 <p><font color = "#FF0000">* required fields</font></p>
37 </body>
38 </html>

```

Create the servlet named Registration in Listing 37.9.

LISTING 37.9 Registration.java

```

1  package chapter37;
2
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import java.io.*;
6  import java.sql.*;
7
8  public class Registration extends HttpServlet {
9      // Use a prepared statement to store a student into the database
10     private PreparedStatement pstmt;
11
12     /** Initialize variables */
13     public void init() throws ServletException {
14         initializeJdbc();
15     }
16
17     /** Process the HTTP Get request */
18     public void doGet(HttpServletRequest request, HttpServletResponse
19         response) throws ServletException, IOException {
20         response.setContentType("text/html");
21         PrintWriter out = response.getWriter();
22
23         // Obtain data from the form
24         String lastName = request.getParameter("lastName");
25         String firstName = request.getParameter("firstName");
26         String mi = request.getParameter("mi");
27         String telephone = request.getParameter("telephone");
28         String email = request.getParameter("email");
29         String street = request.getParameter("street");
30         String city = request.getParameter("city");
31         String state = request.getParameter("state");
32         String zip = request.getParameter("zip");
33
34         if (lastName.length() == 0 || firstName.length() == 0) {
35             out.println("Last Name and First Name are required");
36         }

```

```

37     else {
38         // Ask for confirmation
39         out.println("You entered the following data");
40         out.println("<p>Last name: " + lastName);
41         out.println("<br>First name: " + firstName);
42         out.println("<br>MI: " + mi);
43         out.println("<br>Telephone: " + telephone);
44         out.println("<br>Email: " + email);
45         out.println("<br>Address: " + street);
46         out.println("<br>City: " + city);
47         out.println("<br>State: " + state);
48         out.println("<br>Zip: " + zip);
49
50         // Set the action for processing the answers
51         out.println("<p><form method=\"post\" action=\"" +
52             "Registration">");
53         // Set hidden values
54         out.println("<p><input type=\"hidden\" " +
55             "value=\"" + lastName + " name=\"lastName\">");
56         out.println("<p><input type=\"hidden\" " +
57             "value=\"" + firstName + " name=\"firstName\">");
58         out.println("<p><input type=\"hidden\" " +
59             "value=\"" + mi + " name=\"mi\">");
60         out.println("<p><input type=\"hidden\" " +
61             "value=\"" + telephone + " name=\"telephone\">");
62         out.println("<p><input type=\"hidden\" " +
63             "value=\"" + email + " name=\"email\">");
64         out.println("<p><input type=\"hidden\" " +
65             "value=\"" + street + " name=\"street\">");
66         out.println("<p><input type=\"hidden\" " +
67             "value=\"" + city + " name=\"city\">");
68         out.println("<p><input type=\"hidden\" " +
69             "value=\"" + state + " name=\"state\">");
70         out.println("<p><input type=\"hidden\" " +
71             "value=\"" + zip + " name=\"zip\">");
72         out.println("<p><input type=\"submit\" value=\"Confirm\" >");
73         out.println("</form>");
74     }
75
76     out.close(); // Close stream
77 }
78
79 /** Process the HTTP Post request */
80 public void doPost(HttpServletRequest request, HttpServletResponse
81     response) throws ServletException, IOException {
82     response.setContentType("text/html");
83     PrintWriter out = response.getWriter();
84
85     try {
86         String lastName = request.getParameter("lastName");
87         String firstName = request.getParameter("firstName");
88         String mi = request.getParameter("mi");
89         String telephone = request.getParameter("telephone");
90         String email = request.getParameter("email");
91         String street = request.getParameter("street");
92         String city = request.getParameter("city");
93         String state = request.getParameter("state");
94         String zip = request.getParameter("zip");
95
96         storeStudent(lastName, firstName, mi, telephone, email,
97             street, city, state, zip);

```

```

98
99     out.println(firstName + " " + lastName +
100         " is now registered in the database");
101 }
102 catch(Exception ex) {
103     out.println("Error: " + ex.getMessage());
104 }
105 }
106
107 /** Initialize database connection */
108 private void initializeJdbc() {
109     try {
110         // Load the JDBC driver
111         Class.forName("com.mysql.jdbc.Driver");
112         System.out.println("Driver loaded");
113
114         // Establish a connection
115         Connection conn = DriverManager.getConnection
116             ("jdbc:mysql://localhost/javabook" , "scott", "tiger");
117         System.out.println("Database connected");
118
119         // Create a Statement
120         pstmt = conn.prepareStatement("insert into Address " +
121             "(lastName, firstName, mi, telephone, email, street, city, "
122             + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
123     }
124     catch (Exception ex) {
125         System.out.println(ex);
126     }
127 }
128
129 /** Store a student record to the database */
130 private void storeStudent(String lastName, String firstName,
131     String mi, String phone, String email, String address,
132     String city, String state, String zip) throws SQLException {
133     pstmt.setString(1, lastName);
134     pstmt.setString(2, firstName);
135     pstmt.setString(3, mi);
136     pstmt.setString(4, phone);
137     pstmt.setString(5, email);
138     pstmt.setString(6, address);
139     pstmt.setString(7, city);
140     pstmt.setString(8, state);
141     pstmt.setString(9, zip);
142     pstmt.executeUpdate();
143 }
144 }

```

The servlet processes the GET request by generating an HTML page that displays the client's input and asks for the client's confirmation. The input data consist of hidden values in the newly generated forms, so they will be sent back in the confirmation request. The confirmation request uses the POST method. The servlet retrieves the hidden values and stores them in the database.

Since the first request does not write anything to the database, it is appropriate to use the GET method. Since the second request results in an update to the database, the POST method must be used.



Note

The hidden values could also be sent from the URL query string if the request used the GET method.

37.8.2 Session Tracking Using Cookies

You can track sessions using cookies, which are small text files that store sets of name/value pairs on the disk in the client’s computer. Cookies are sent from the server through the instructions in the header of the HTTP response. The instructions tell the browser to create a cookie with a given name and its associated value. If the browser already has a cookie with the key name, the value will be updated. The browser will then send the cookie with any request submitted to the same server. Cookies can have expiration dates set, after which they will not be sent to the server. The `javax.servlet.http.Cookie` is used to create and manipulate cookies, as shown in Figure 37.23.

javax.servlet.http.Cookie	
+Cookie(name: String, value: String)	Creates a cookie with the specified name-value pair.
+getName(): String	Returns the name of the cookie.
+getValue(): String	Returns the value of the cookie.
+setValue(newValue: String): void	Assigns a new value to a cookie after the cookie is created.
+getMaxAge(): int	Returns the maximum age of the cookie, specified in seconds.
+setMaxAge(expiration: int): void	Specifies the maximum age of the cookie. By default, this value is -1, which implies that the cookie persists until the browser exits. If you set this value to 0, the cookie is deleted.
+getSecure(): boolean	Returns true if the browser is sending cookies only over a secure protocol.
+setSecure(flag: boolean): void	Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL.
+getComment(): String	Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
+setComment(purpose: String): void	Sets the comment for this cookie.

FIGURE 37.23 `Cookie` stores a name/value pair and other information about the cookie.

To send a cookie to the browser, use the `addCookie` method in the `HttpServletResponse` class, as shown below:

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

where `response` is an instance of `HttpServletResponse`.

To obtain cookies from a browser, use

```
request.getCookies();
```

where `request` is an instance of `HttpServletRequest`.

To demonstrate the use of cookies, let us create an example that accomplishes the same task as Listing 37.9, `Registration.java`. Instead of using hidden values for session tracking, it uses cookies.

Create the servlet named `RegistrationWithCookie` in Listing 37.10. Create an HTML file named `RegistrationWithCookie.html` that is identical to `Registration.html` except that the action is replaced by `RegistrationWithCookie.java`.

LISTING 37.10 `RegistrationWithCookie.java`

```
1 package chapter37;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
```

```

6  import java.sql.*;
7
8  public class RegistrationWithCookie extends HttpServlet {
9      private static final String CONTENT_TYPE = "text/html";
10     // Use a prepared statement to store a student into the database
11     private PreparedStatement pstmt;
12
13     /** Initialize variables */
14     public void init() throws ServletException {
15         initializeJdbc();
16     }
17
18     /** Process the HTTP Get request */
19     public void doGet(HttpServletRequest request, HttpServletResponse
20         response) throws ServletException, IOException {
21         response.setContentType("text/html");
22         PrintWriter out = response.getWriter();
23
24         // Obtain data from the form
25         String lastName = request.getParameter("lastName");
26         String firstName = request.getParameter("firstName");
27         String mi = request.getParameter("mi");
28         String telephone = request.getParameter("telephone");
29         String email = request.getParameter("email");
30         String street = request.getParameter("street");
31         String city = request.getParameter("city");
32         String state = request.getParameter("state");
33         String zip = request.getParameter("zip");
34
35         if (lastName.length() == 0 || firstName.length() == 0) {
36             out.println("Last Name and First Name are required");
37         }
38         else {
39             // Create cookies and send cookies to browsers
40             Cookie cookieLastName = new Cookie("lastName", lastName);
41             // cookieLastName.setMaxAge(1000);
42             response.addCookie(cookieLastName);
43             Cookie cookieFirstName = new Cookie("firstName", firstName);
44             response.addCookie(cookieFirstName);
45             // cookieFirstName.setMaxAge(0);
46             Cookie cookieMi = new Cookie("mi", mi);
47             response.addCookie(cookieMi);
48             Cookie cookieTelephone = new Cookie("telephone", telephone);
49             response.addCookie(cookieTelephone);
50             Cookie cookieEmail = new Cookie("email", email);
51             response.addCookie(cookieEmail);
52             Cookie cookieStreet = new Cookie("street", street);
53             response.addCookie(cookieStreet);
54             Cookie cookieCity = new Cookie("city", city);
55             response.addCookie(cookieCity);
56             Cookie cookieState = new Cookie("state", state);
57             response.addCookie(cookieState);
58             Cookie cookieZip = new Cookie("zip", zip);
59             response.addCookie(cookieZip);
60
61             // Ask for confirmation
62             out.println("You entered the following data");
63             out.println("<p>Last name: " + lastName);
64             out.println("<br>First name: " + firstName);
65             out.println("<br>MI: " + mi);
66             out.println("<br>Telephone: " + telephone);

```

```

67         out.println("<br>Email: " + email);
68         out.println("<br>Street: " + street);
69         out.println("<br>City: " + city);
70         out.println("<br>State: " + state);
71         out.println("<br>Zip: " + zip);
72
73         // Set the action for processing the answers
74         out.println("<p><form method=\"post\" action=" +
75             "RegistrationWithCookie>");
76         out.println("<p><input type=\"submit\" value=\"Confirm\" >");
77         out.println("</form>");
78     }
79
80     out.close(); // Close stream
81 }
82
83 /** Process the HTTP Post request */
84 public void doPost(HttpServletRequest request, HttpServletResponse
85     response) throws ServletException, IOException {
86     response.setContentType(CONTENT_TYPE);
87     PrintWriter out = response.getWriter();
88
89     String lastName = "";
90     String firstName = "";
91     String mi = "";
92     String telephone = "";
93     String email = "";
94     String street = "";
95     String city = "";
96     String state = "";
97     String zip = "";
98
99     // Read the cookies
100    Cookie[] cookies = request.getCookies();
101
102    // Get cookie values
103    for (int i = 0; i < cookies.length; i++) {
104        if (cookies[i].getName().equals("lastName"))
105            lastName = cookies[i].getValue();
106        else if (cookies[i].getName().equals("firstName"))
107            firstName = cookies[i].getValue();
108        else if (cookies[i].getName().equals("mi"))
109            mi = cookies[i].getValue();
110        else if (cookies[i].getName().equals("telephone"))
111            telephone = cookies[i].getValue();
112        else if (cookies[i].getName().equals("email"))
113            email = cookies[i].getValue();
114        else if (cookies[i].getName().equals("street"))
115            street = cookies[i].getValue();
116        else if (cookies[i].getName().equals("city"))
117            city = cookies[i].getValue();
118        else if (cookies[i].getName().equals("state"))
119            state = cookies[i].getValue();
120        else if (cookies[i].getName().equals("zip"))
121            zip = cookies[i].getValue();
122    }
123
124    try {
125        storeStudent(lastName, firstName, mi, telephone, email, street,
126            city, state, zip);
127    }

```



```

128         out.println(firstName + " " + lastName +
129             " is now registered in the database");
130
131         out.close(); // Close stream
132     }
133     catch(Exception ex) {
134         out.println("Error: " + ex.getMessage());
135     }
136 }
137
138 /** Initialize database connection */
139 private void initializeJdbc() {
140     try {
141         // Load the JDBC driver
142         Class.forName("com.mysql.jdbc.Driver");
143         System.out.println("Driver loaded");
144
145         // Establish a connection
146         Connection conn = DriverManager.getConnection
147             ("jdbc:mysql://localhost/javabook", "scott", "tiger");
148         System.out.println("Database connected");
149
150         // Create a Statement
151         pstmt = conn.prepareStatement("insert into Address " +
152             "(lastName, firstName, mi, telephone, email, street, city, "
153             + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
154     }
155     catch (Exception ex) {
156         System.out.println(ex);
157     }
158 }
159
160 /** Store a student record to the database */
161 private void storeStudent(String lastName, String firstName,
162     String mi, String telephone, String email, String street,
163     String city, String state, String zip) throws SQLException {
164     pstmt.setString(1, lastName);
165     pstmt.setString(2, firstName);
166     pstmt.setString(3, mi);
167     pstmt.setString(4, telephone);
168     pstmt.setString(5, email);
169     pstmt.setString(6, street);
170     pstmt.setString(7, city);
171     pstmt.setString(8, state);
172     pstmt.setString(9, zip);
173     pstmt.executeUpdate();
174 }
175 }

```

You have to create a cookie for each value you want to track, using the `Cookie` class's only constructor, which defines a cookie's name and value as shown below (line 40):

```
Cookie cookieLastName = new Cookie("lastName", lastName);
```

To send the cookie to the browser, use a statement like this one (line 42):

```
response.addCookie(cookieLastName);
```

If a cookie with the same name already exists in the browser, its value is updated; otherwise, a new cookie is created.

Cookies are automatically sent to the Web server with each request from the client. The servlet retrieves all the cookies into an array using the `getCookies` method (line 100):

```
Cookie[] cookies = request.getCookies();
```

To obtain the name of the cookie, use the `getName` method (line 104):

```
String name = cookies[i].getName();
```

The cookie's value can be obtained using the `getValue` method:

```
String value = cookies[i].getValue();
```

Cookies are stored as strings just like form parameters and hidden values. If a cookie represents a numeric value, you have to convert it into an integer or a double, using the `parseInt` method in the `Integer` class or the `parseDouble` method in the `Double` class.

By default, a newly created cookie persists until the browser exits. However, you can set an expiration date, using the `setMaxAge` method, to allow a cookie to stay in the browser for up to 2,147,483,647 seconds (approximately 24,855 days).

37.8.3 Session Tracking Using the Servlet API

You have now learned both session tracking using hidden values and session tracking using cookies. These two session-tracking methods have problems. They send data to the browser either as hidden values or as cookies. The data are not secure, and anybody with knowledge of computers can obtain them. The hidden data are in HTML form, which can be viewed from the browser. Cookies are stored in the Cache directory of the browser. Because of security concerns, some browsers do not accept cookies. The client can turn the cookies off and limit their number. Another problem is that hidden data and cookies pass data as strings. You cannot pass objects using these two methods.

To address these problems, Java servlet API provides the `javax.servlet.http.HttpSession` interface, which provides a way to identify a user across more than one page request or visit to a website and to store information about that user. The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The session enables tracking of a large set of data. The data can be stored as objects and are secure because they are kept on the server side.

To use the Java servlet API for session tracking, first create a session object using the `getSession()` method in the `HttpServletRequest` interface:

```
HttpSession session = request.getSession();
```

This obtains the session or creates a new session if the client does not have a session on the server.

The `HttpSession` interface provides the methods for reading and storing data to the session, and for manipulating the session, as shown in Figure 37.24.



Note

HTTP is stateless. So how does the server associate a session with multiple requests from the same client? This is handled behind the scenes by the servlet container and is transparent to the servlet programmer.

To demonstrate using `HttpSession`, let us rewrite Listing 37.9, `Registration.java`, and Listing 37.10, `RegistrationWithCookie.java`. Instead of using hidden values or cookies for session tracking, it uses servlet `HttpSession`.

<pre><<interface>> javax.servlet.http.HttpSession</pre>	
<pre>+getAttribute(name: String): Object +setAttribute(name: String, value: Object): void +getId(): String +getLastAccessedTime(): long +invalidate(): void +isNew(): boolean +removeAttribute(name: String): void +getMaxInactiveInterval(): int +setMaxInactiveInterval(interval: int): void</pre>	<p>Returns the object bound with the specified name in this session, or null if no object is bound under the name.</p> <p>Binds an object to this session, using the specified name. If an object of the same name is already bound to the session, the object is replaced.</p> <p>Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.</p> <p>Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.</p> <p>Invalidates this session, then unbinds any objects bound to it.</p> <p>Returns true if the session was just created in the current request.</p> <p>Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.</p> <p>Returns the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates that the session will never time-out. Use <code>setMaxInactiveInterval</code> to specify this value.</p>

FIGURE 37.24 `HttpSession` establishes a persistent session between a client with multiple requests and the server.

Create the servlet named `RegistrationWithHttpSession` in Listing 37.11. Create an HTML file named `RegistrationWithHttpSession.html` that is identical to `Registration.html` except that the action is replaced by `RegistrationWithHttpSession`.

LISTING 37.11 `RegistrationWithHttpSession.java`

```
1 package chapter37;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6 import java.sql.*;
7
8 public class RegistrationWithHttpSession extends HttpServlet {
9     // Use a prepared statement to store a student into the database
10    private PreparedStatement pstmt;
11
12    /** Initialize variables */
13    public void init() throws ServletException {
14        initializeJdbc();
15    }
16
17    /** Process the HTTP Get request */
18    public void doGet(HttpServletRequest request, HttpServletResponse
19        response) throws ServletException, IOException {
```

```

20 // Set response type and output stream to the browser
21 response.setContentType("text/html");
22 PrintWriter out = response.getWriter();
23
24 // Obtain data from the form
25 String lastName = request.getParameter("lastName");
26 String firstName = request.getParameter("firstName");
27 String mi = request.getParameter("mi");
28 String telephone = request.getParameter("telephone");
29 String email = request.getParameter("email");
30 String street = request.getParameter("street");
31 String city = request.getParameter("city");
32 String state = request.getParameter("state");
33 String zip = request.getParameter("zip");
34
35 if (lastName.length() == 0 || firstName.length() == 0) {
36     out.println("Last Name and First Name are required");
37 }
38 else {
39     // Create an Address object
40     Address address = new Address();
41     address.setLastName(lastName);
42     address.setFirstName(firstName);
43     address.setMi(mi);
44     address.setTelephone(telephone);
45     address.setEmail(email);
46     address.setStreet(street);
47     address.setCity(city);
48     address.setState(state);
49     address.setZip(zip);
50
51     // Get an HttpSession or create one if it does not exist
52     HttpSession httpSession = request.getSession();
53
54     // Store student object to the session
55     httpSession.setAttribute("address", address);
56
57     // Ask for confirmation
58     out.println("You entered the following data");
59     out.println("<p>Last name: " + lastName);
60     out.println("<p>First name: " + firstName);
61     out.println("<p>MI: " + mi);
62     out.println("<p>Telephone: " + telephone);
63     out.println("<p>Email: " + email);
64     out.println("<p>Address: " + street);
65     out.println("<p>City: " + city);
66     out.println("<p>State: " + state);
67     out.println("<p>Zip: " + zip);
68
69     // Set the action for processing the answers
70     out.println("<p><form method=\"post\" action=" +
71         "RegistrationWithHttpSession>");
72     out.println("<p><input type=\"submit\" value=\"Confirm\" >");
73     out.println("</form>");
74 }
75
76 out.close(); // Close stream
77 }
78
79 /** Process the HTTP Post request */

```

```

80 public void doPost(HttpServletRequest request, HttpServletResponse
81     response) throws ServletException, IOException {
82     // Set response type and output stream to the browser
83     response.setContentType("text/html");
84     PrintWriter out = response.getWriter();
85
86     // Obtain the HttpSession
87     HttpSession httpSession = request.getSession();
88
89     // Get the Address object in the HttpSession
90     Address address = (Address)(httpSession.getAttribute("address"));
91
92     try {
93         storeStudent(address);
94
95         out.println(address.getFirstName() + " " + address.getLastName()
96             + " is now registered in the database");
97         out.close(); // Close stream
98     }
99     catch(Exception ex) {
100         out.println("Error: " + ex.getMessage());
101     }
102 }
103
104 /** Initialize database connection */
105 private void initializeJdbc() {
106     try {
107         // Load the JDBC driver
108         Class.forName("com.mysql.jdbc.Driver");
109         System.out.println("Driver loaded");
110
111         // Establish a connection
112         Connection conn = DriverManager.getConnection
113             ("jdbc:mysql://localhost/javabook", "scott", "tiger");
114         System.out.println("Database connected");
115
116         // Create a Statement
117         pstmt = conn.prepareStatement("insert into Address " +
118             " (lastName, firstName, mi, telephone, email, street, city, "
119             + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
120     }
121     catch (Exception ex) {
122         System.out.println(ex);
123     }
124 }
125
126 /** Store an address to the database */
127 private void storeStudent(Address address) throws SQLException {
128     pstmt.setString(1, address.getLastName());
129     pstmt.setString(2, address.getFirstName());
130     pstmt.setString(3, address.getMi());
131     pstmt.setString(4, address.getTelephone());
132     pstmt.setString(5, address.getEmail());
133     pstmt.setString(6, address.getStreet());
134     pstmt.setString(7, address.getCity());
135     pstmt.setString(8, address.getState());
136     pstmt.setString(9, address.getZip());
137     pstmt.executeUpdate();
138 }
139 }

```

The statement (line 52)

```
HttpSession httpSession = request.getSession();
```

obtains a session, or creates a new session if the session does not exist.

Since objects can be stored in `HttpSession`, this program defines an `Address` class. An `Address` object is created and is stored in the session using the `setAttribute` method, which binds the object with a name like the one shown below (line 55):

```
httpSession.setAttribute("address", address);
```

To retrieve the object, use the following statement (line 90):

```
Address address = (Address)(httpSession.getAttribute("address"));
```

There is only one session between a client and a servlet. You can store any number of objects in a session. By default, the maximum inactive interval on many Web servers including Tomcat and GlassFish is 1800 seconds (i.e., a half-hour), meaning that the session expires if there is no activity for 30 minutes. You can change the default using the `setMaxInactiveInterval` method. For example, to set the maximum inactive interval to one hour, use

```
httpSession.setMaxInactiveInterval(3600);
```

If you set a negative value, the session will never expire.

For this servlet program to work, you have to create the `Address` class in NetBeans, as follows:

1. Choose *New, Java Class* from the context menu of the `1iangweb` node in the project pane to display the New Java Class dialog box.
2. Enter `Address` as the Class Name and `chapter37` as the package name. Click *Finish* to create the class.
3. Enter the code, as shown in Listing 37.12.

LISTING 37.12 Address.java

```

1 package chapter37;
2
3 public class Address {
4     private String firstName;
5     private String mi;
6     private String lastName;
7     private String telephone;
8     private String street;
9     private String city;
10    private String state;
11    private String email;
12    private String zip;
13
14    public String getFirstName() {
15        return this.firstName;
16    }
17
18    public void setFirstName(String firstName) {
19        this.firstName = firstName;
20    }
21
22    public String getMi() {
23        return this.mi;
24    }

```

```
25
26 public void setMi(String mi) {
27     this.mi = mi;
28 }
29
30 public String getLastName() {
31     return this.lastName;
32 }
33
34 public void setLastName(String lastName) {
35     this.lastName = lastName;
36 }
37
38 public String getTelephone() {
39     return this.telephone;
40 }
41
42 public void setTelephone(String telephone) {
43     this.telephone = telephone;
44 }
45
46 public String getEmail() {
47     return this.email;
48 }
49
50 public void setEmail(String email) {
51     this.email = email;
52 }
53
54 public String getStreet() {
55     return this.street;
56 }
57
58 public void setStreet(String street) {
59     this.street = street;
60 }
61
62 public String getCity() {
63     return this.city;
64 }
65
66 public void setCity(String city) {
67     this.city = city;
68 }
69
70 public String getState() {
71     return this.state;
72 }
73
74 public void setState(String state) {
75     this.state = state;
76 }
77
78 public String getZip() {
79     return this.zip;
80 }
81
82 public void setZip(String zip) {
83     this.zip = zip;
84 }
85 }
```


This support class will also be reused in the upcoming chapters.



- 37.8.1** What is session tracking? What are three techniques for session tracking?
- 37.8.2** How do you create a cookie, send a cookie to a browser, get cookies from a browser, get the name of a cookie, set a new value in the cookie, and set cookie expiration time?
- 37.8.3** Do you have to create five **Cookie** objects in the servlet in order to send five cookies to the browser?
- 37.8.4** How do you get a session, set object value for the session, and get object value from the session?
- 37.8.5** Suppose you inserted the following code in line 53 in Listing 37.11:

```
httpSession.setMaxInactiveInterval(1);
```

What would happen after the user clicked the *Confirm* button from the browser? Test your answer by running the program.

- 37.8.6** Suppose you inserted the following code in line 53 in Listing 37.11:

```
httpSession.setMaxInactiveInterval(-1);
```

What would happen after the user clicked the *Confirm* button from the browser?

KEY TERMS

Common Gateway Interface	37-3	URL query string	37-3
CGI programs	37-3	servlet	37-2
cookie	37-30	servlet container (servlet engine)	37-4
GET and POST methods	37-3	servlet life-cycle methods	37-10
GlassFish	37-5	Tomcat	37-5
HTML form	37-15		

CHAPTER SUMMARY

1. A servlet is a special kind of program that runs from a Web server. Tomcat and GlassFish are Web servers that can run servlets.
2. A servlet URL is specified by the host name, port, and request string (e.g., <http://localhost:8084/liangweb/ServletClass>). There are several ways to invoke a servlet: (1) by typing a servlet URL from a Web browser, (2) by placing a hyper link in an HTML page, and (3) by embedding a servlet URL in an HTML form. All the requests trigger the GET method, except that in the HTML form you can explicitly specify the POST method.
3. You develop a servlet by defining a class that extends the **HttpServlet** class, implements the **doGet(HttpServletRequest, HttpServletResponse)** method to respond to the GET method, and implements the **doPost(HttpServletRequest, HttpServletResponse)** method to respond to the POST method.

- 4. The request information passed from a client to the servlet is contained in an object of `HttpServletRequest`. You can use the methods `getParameter`, `getParameterValues`, `getRemoteAddr`, `getRemoteHost`, `getHeader`, `getQueryString`, `getCookies`, and `getSession` to obtain the information from the request.
- 5. The content sent back to the client is contained in an object of `HttpServletResponse`. To send content to the client, first set the type of the content (e.g., `html/plain`) using the `setContentType(contentType)` method, then output the content through an I/O stream on the `HttpServletResponse` object. You can obtain a character `PrintWriter` stream using the `getWriter()` method and obtain a binary `OutputStream` using the `getOutputStream()` method.
- 6. A servlet may be shared by many clients. When the servlet is first created, its `init` method is called. It is not called again as long as the servlet is not destroyed. The `service` method is invoked each time the server receives a request for the servlet. The server spawns a new thread and invokes `service`. The `destroy` method is invoked after a timeout period has passed or the Web server is stopped.
- 7. There are three ways to track a session. You can track a session by passing data from the servlet to the client as a hidden value in a dynamically generated HTML form by including a field such as `<input type="hidden" name="lastName" value="Smith">`. The next request will submit the data back to the servlet. The servlet retrieves this hidden value just like any other parameter value using the `getParameter` method.
- 8. You can track sessions using cookies. A cookie is created using the constructor `new Cookie(String name, String value)`. Cookies are sent from the server through the object of `HttpServletResponse` using the `addCookie(aCookie)` method to tell the browser to add a cookie with a given name and its associated value. If the browser already has a cookie with the key name, the value will be updated. The browser will then send the cookie with any request submitted to the same server. Cookies can have expiration dates set, after which they will not be sent to the server.
- 9. Java servlet API provides a session-tracking tool that enables tracking of a large set of data. A session can be obtained using the `getSession()` method through an `HttpServletRequest` object. The data can be stored as objects and are secure because they are kept on the server side using the `setAttribute(String name, Object value)` method.

Quiz

Answer the quiz for this chapter online at the book Companion Website.

a. ☐

b. ☒

c. ☐

PROGRAMMING EXERCISES

MyProgrammingLab™

Section 37.5

***37.1** (*Factorial table*) Write a servlet to display a table that contains factorials for the numbers from 0 to 10, as shown in Figure 37.25.

37-42 Chapter 37 Servlets

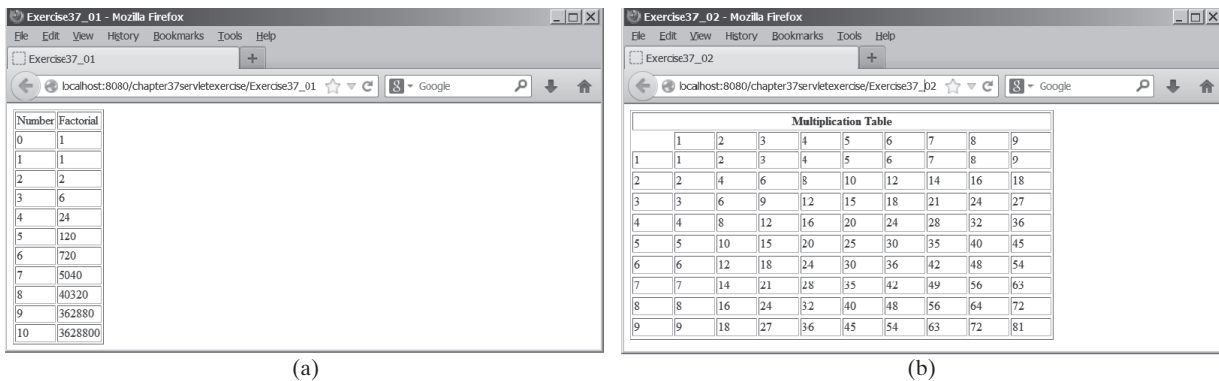


FIGURE 37.25 (a) The servlet displays factorials for the numbers from 0 to 10 in a table. (b) The servlet displays the multiplication table.

- *37.2** (*Multiplication table*) Write a servlet to display a multiplication table, as shown in Figure 37.25b.
- *37.3** (*Visit count*) Develop a servlet that displays the number of visits on the servlet. Also display the client’s host name and IP address, as shown in Figure 37.26.

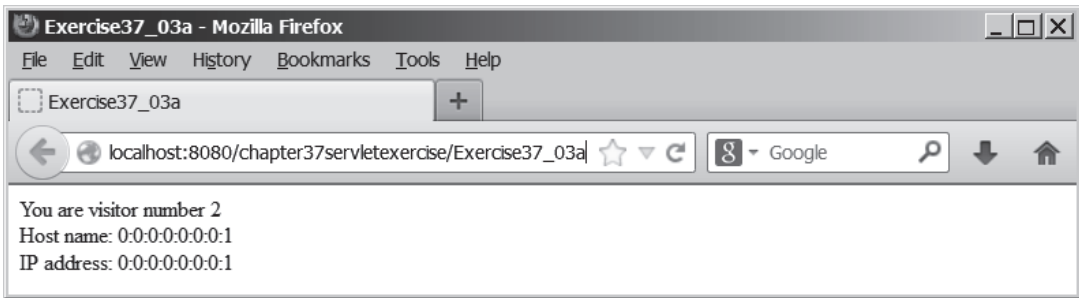


FIGURE 37.26 The servlet displays the number of visits and the client’s host name, IP address, and request URL.

Implement this program in three different ways:

1. Use an instance variable to store **count**. When the servlet is created for the first time, **count** is 0. **count** is incremented every time the servlet’s **doGet** method is invoked. When the Web server stops, **count** is lost.
2. Store the count in a file named **Exercise39_3.dat**, and use **RandomAccess-File** to read the count in the servlet’s **init** method. The count is incremented every time the servlet’s **doGet** method is invoked. When the Web server stops, store the count back to the file.
3. Instead of counting total visits from all clients, count the visits by each client identified by the client’s IP address. Use **Map** to store a pair of IP addresses and visit counts. For the first visit, an entry is created in the map. For subsequent visits, the visit count is updated.

Section 37.6

- *37.4** (*Calculate tax*) Write an HTML form to prompt the user to enter taxable income and filing status, as shown in Figure 37.27a. Clicking the *Compute Tax* button invokes a servlet to compute and display the tax, as shown in Figure 37.27b. Use the **computeTax** method introduced in Listing 3.7, *ComputingTax.java*, to compute tax.

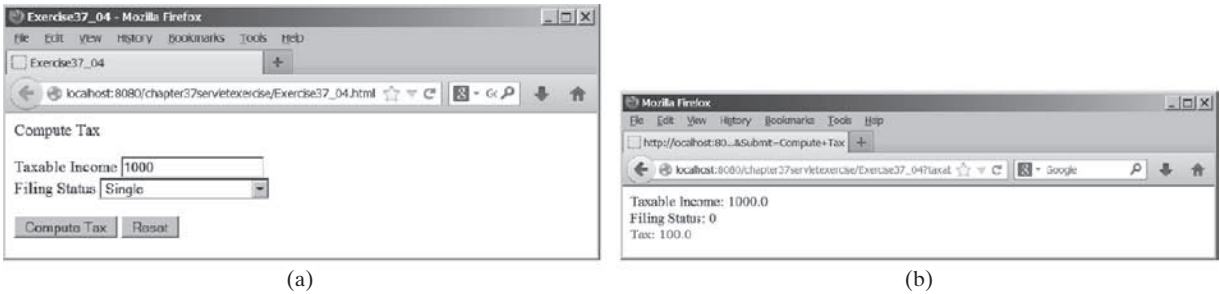


FIGURE 37.27 The servlet computes the tax.

***37.5** (*Calculate loan*) Write an HTML form that prompts the user to enter loan amount, interest rate, and number of years, as shown in Figure 37.28a. Clicking the *Compute Loan Payment* button invokes a servlet to compute and display the monthly and total loan payments, as shown in Figure 37.28b. Use the *Loan* class given in Listing 10.2, *Loan.java*, to compute the monthly and total payments.

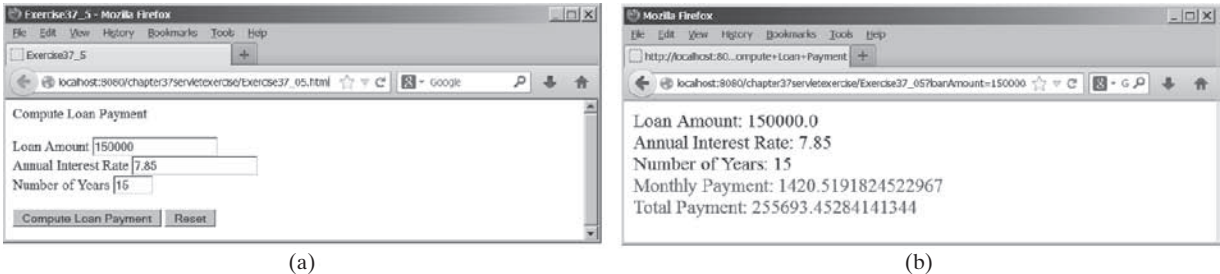


FIGURE 37.28 The servlet computes the loan payment.

****37.6** (*Find scores from text files*) Write a servlet that displays the student name and the current score, given the SSN and class ID. For each class, a text file is used to store the student name, SSN, and current score. The file is named after the class ID with .txt extension. For instance, if the class ID were csci1301, the file name would be csci1301.txt. Suppose each line consists of student name, SSN, and score. These three items are separated by the # sign. Create an HTML form that enables the user to enter the SSN and class ID, as shown in Figure 37.29a. Upon clicking the *Submit* button, the result is displayed, as shown in Figure 37.29b. If the SSN or the class ID does not match, report an error. Assume three courses are available: CSCI1301, CSCI1302, and CSCI13720.

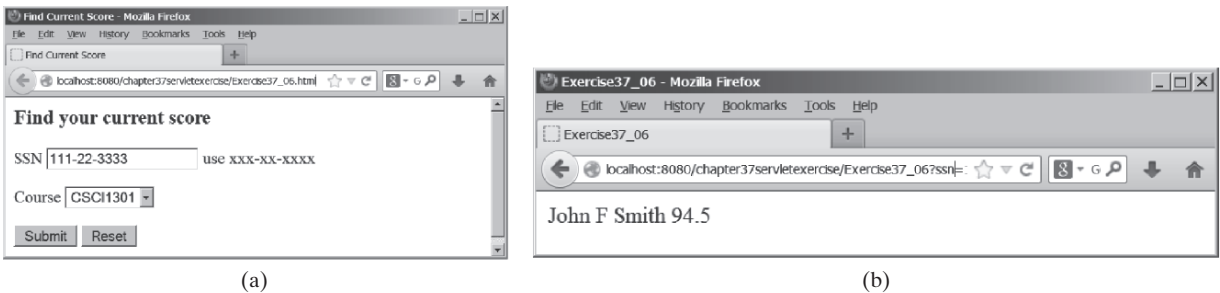


FIGURE 37.29 The HTML form accepts the SSN and class ID from the user and sends them to the servlet to obtain the score.

Section 37.7

****37.7** (*Find scores from database tables*) Rewrite the preceding servlet. Assume for each class, a table is used to store the student name, ssn, and score. The table name is the same as the class ID. For instance, if the class ID were csci1301, the table name would be csci1301.

***37.8** (*Change the password*) Write a servlet that enables the user to change the password from an HTML form, as shown in Figure 37.30a. Suppose the user information is stored in a database table named Account with three columns: username, password, and name, where name is the real name of the user. The servlet performs the following tasks:

- a. Verify that the username and old password are in the table. If not, report the error and redisplay the HTML form.
- b. Verify that the new password and the confirmed password are the same. If not, report this error and redisplay the HTML form.
- c. If the user information is entered correctly, update the password and report the status of the update to the user, as shown in Figure 37.30b.

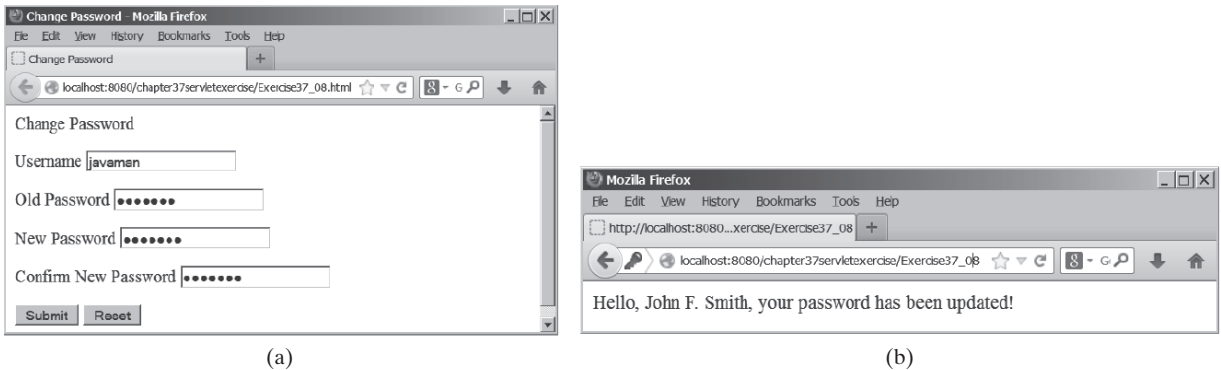


FIGURE 37.30 The user enters the username and the old password and sets a new password. The servlet reports the status of the update to the user.

****37.9** (*Display database tables*) Write an HTML form that prompts the user to enter or select a JDBC driver, database URL, username, password, and table name, as shown in Figure 37.31a. Clicking the *Submit* button displays the table content, as shown in Figure 37.31b.

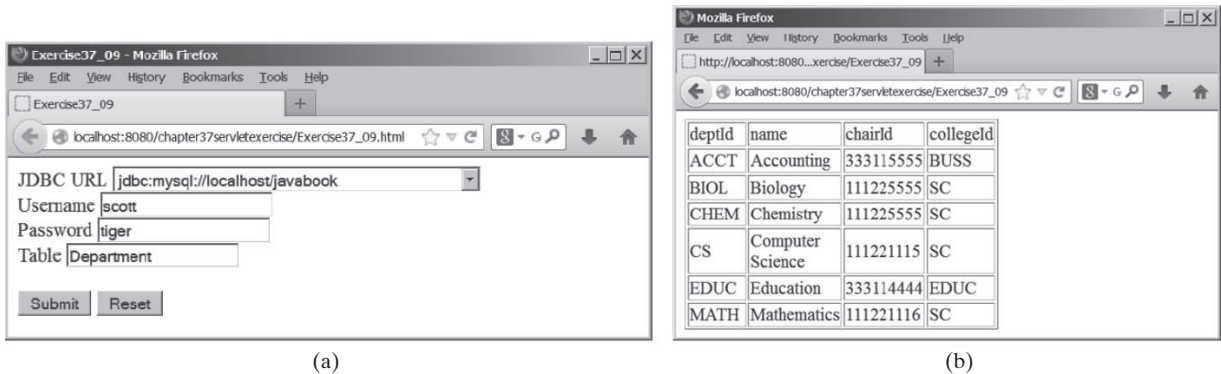


FIGURE 37.31 The user enters database information and specifies a table to display its content.

Section 37.8

- *37.10** (*Store cookies*) Write a servlet that stores the following cookies in a browser, and set their max age for two days.
- Cookie 1: name is “color” and value is red.
Cookie 2: name is “radius” and value is 5.5.
Cookie 3: name is “count” and value is 2.
- *37.11** (*Retrieve cookies*) Write a servlet that displays all the cookies on the client. The client types the URL of the servlet from the browser to display all the cookies stored on the browser. (see Figure 37.32.)

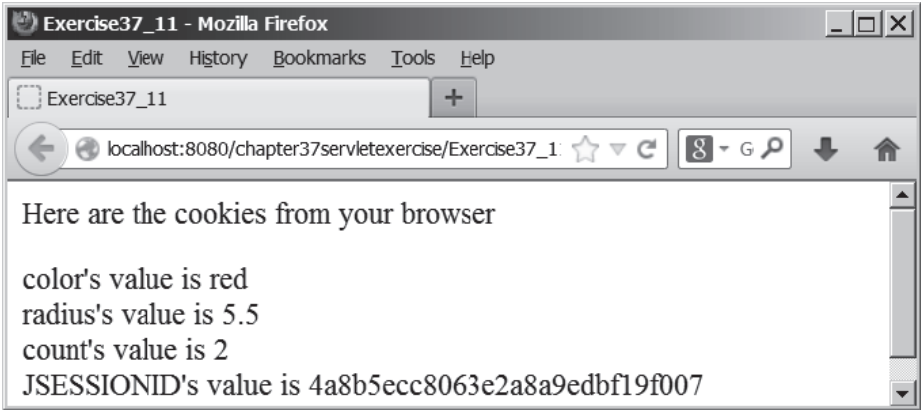


FIGURE 37.32 All the cookies on the client are displayed in the browser.

Comprehensive

- ***37.12** (*Syntax highlighting*) Create an HTML form that prompts the user to enter a Java program in a text area, as shown in Figure 37.33a. The form invokes a servlet that displays the Java source code in a syntax-highlighted HTML format, as shown in Figure 37.33b. The keywords, comments, and literals are displayed in bold navy, green, and blue, respectively.

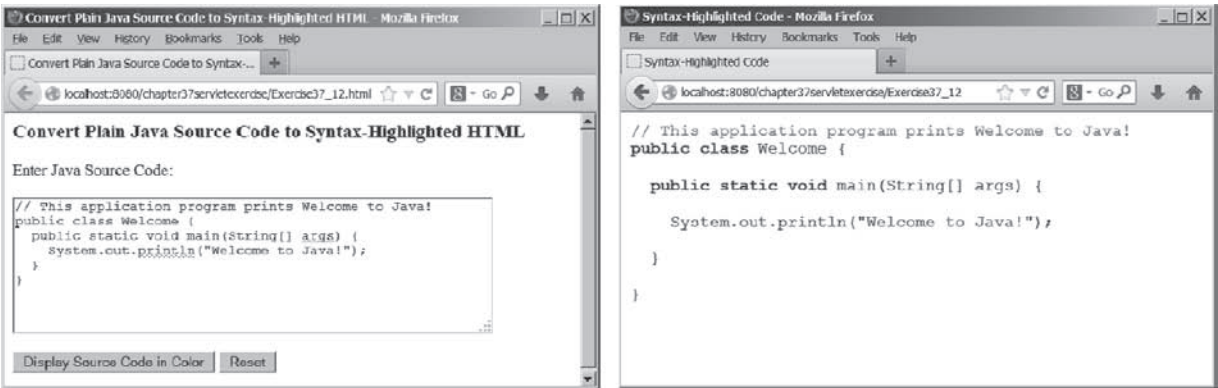


FIGURE 37.33 The Java code in plain text in (a) is displayed in HTML with syntax highlighted in (b).

****37.13** (Access and update a *Staff* table) Write a Java servlet for Exercise 33.1, as shown in Figure 37.34.

The screenshot shows a Mozilla Firefox browser window titled "Exercise37_13 - Mozilla Firefox". The address bar shows "localhost:8080/chapter37servletexercise/Exercise37_13.html". The page content is titled "Staff Information" and contains a form with the following fields: ID (45665), Last Name (Ford), First Name (Smith), MI (D), Address (123 Abercorn), City (Savannah), State (GA), and Telephone (9123445545). Below the form are four buttons: View, Insert, Update, and Clear.

FIGURE 37.34 The webpage lets you view, insert, and update staff information.

*****37.14** (*Opinion poll*) Create an HTML form that prompts the user to answer a question such as “Are you a CS major?”, as shown in Figure 37.35a. When the *Submit* button is clicked, the servlet increases the Yes or No count in a database and displays the current Yes and No counts, as shown in Figure 37.35b.

The figure consists of two screenshots labeled (a) and (b). Screenshot (a) shows a Mozilla Firefox browser window titled "Opinion Poll - Mozilla Firefox". The address bar shows "localhost:8080/chapter37servletexercise/Exercise37_14.html". The page content is titled "Opinion Poll" and contains a form with the question "Are you a CS major?" and two radio buttons labeled "Yes" and "No". Below the form are two buttons: Submit and Reset. Screenshot (b) shows the same browser window after the form has been submitted. The page content now displays "The current Yes count is 2" and "The current No count is 0".

FIGURE 37.35 The HTML form prompts the user to enter Yes or No for a question in (a), and the servlet updates the Yes or No counts in (b).

Create a table named **Poll**, as follows:

```
create table Poll (  
    question varchar(40) primary key,  
    yesCount int,  
    noCount int);
```

Insert one row into the table, as follows:

```
insert into Poll values ('Are you a CS major? ', 0, 0);
```