Chapter 39

Servlets

Objectives

- To explain how a servlet works (§39.2).
- To create/develop/run servlets (§39.3).
- To deploy servlets on application servers such as Tomcat and GlassFish (§39.3).
- To describe the servlets API (§39.4).
- To create simple servlets (§39.5).
- To create and process HTML forms (§39.6).
- To develop servlets to access databases (§39.7).
- To use hidden fields, cookies, and `HttpSession` to track sessions (§39.8).
- To send images from servlets (§39.9).

39.1 Introduction

servlet

Servlet technology is designed primarily for use with the HTTP protocol of the Web. *Servlets* are Java programs that run on a Web server. They can be used to process client requests or produce dynamic Web pages. For example, you can write servlets to generate dynamic Web pages 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

why NetBeans

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 favoriate IDE or no IDE for this chapter.

Note

why servlets

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.

39.2 HTML and Common Gateway Interface

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

39.2.1 Static Web Contents

You create Web pages using HTML. Your Web pages are stored as files on the Web server. The files are usually stored in the /htdocs directory on Unix, as shown in Figure 39.1. 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.

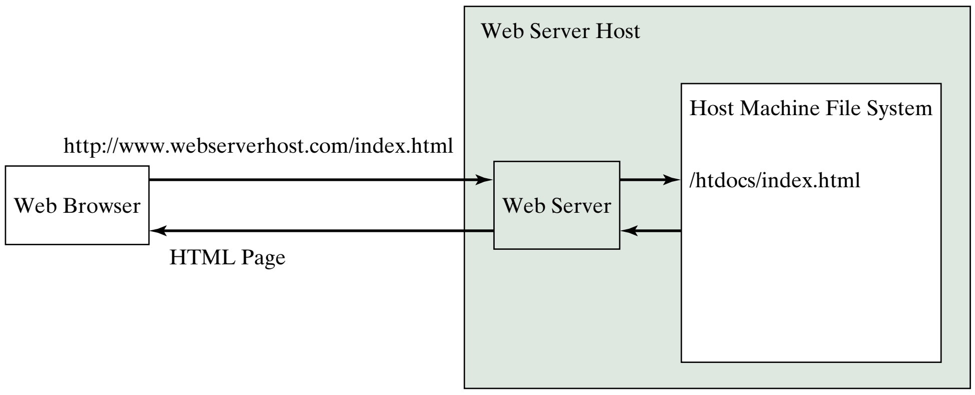


FIGURE 39.1 A Web browser requests a static HTML page from a Web server.

39.2.2 Dynamic Web Contents and Common Gateway Interface

Not all information, however, is static in nature. Stock quotes are updated every minute. 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.

CGI

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 39.2, 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.

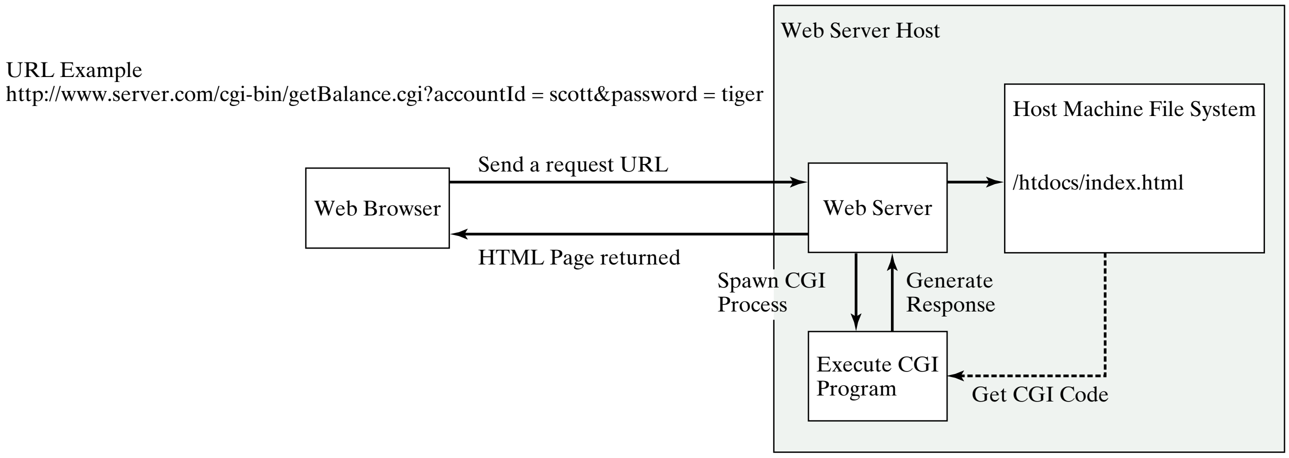


FIGURE 39.2 A Web browser requests a dynamic HTML page from a Web server.

39.2.3 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 §39.6, "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:

query string

```
http://www.webserverhost.com/cgi-bin/
  getBalance.cgi?accounted=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.

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

GET vs. POST

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 Web pages 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 Web pages for the URL may be displayed. To ensure that a new Web page 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.

39.2.4 From CGI to Java Servlets

CGI vs. 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 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.

servlet engine

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.

39.3 Creating and Running Servlets

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 6. When you run a servlet from NetBeans, Tomcat or GlassFish will be

Tomcat
GlassFish

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.

39.3.1 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 an applet. Every Java applet is a subclass of the `Applet` class. You need to override appropriate methods in the `Applet` class to implement the applet. Every servlet is a subclass of the `HttpServlet` class. You need to override appropriate methods in the `HttpServlet` class to implement the servlet. Listing 39.1 is a servlet that generates a response in HTML using the `doGet` method.

LISTING 39.1 FirstServlet.java

```
 1 package chapter39;
 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 }
```

process GET

content type

output to browser

close stream

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 outputing data to the browser.

request
response

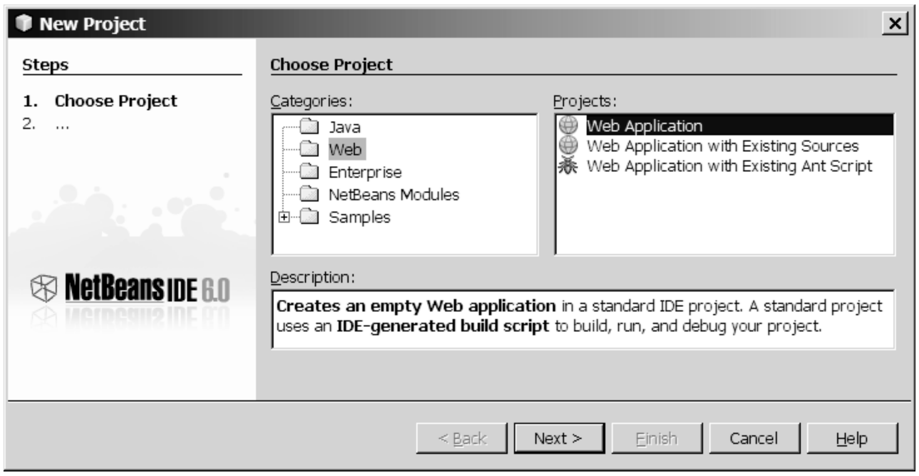
PrintWriter

39.3.2 Creating Servlets in NetBeans

To create a servlet in NetBeans, you have to first create a Web project, as follows:

create a Web project

1. Choose **File** › **New Project** to display the New Project dialog box. Choose Web in the Categories section and Web Application in the Projects section, as shown in Figure 39.3. Click *Next* to display the New Web Application dialog box, as shown in Figure 39.4.


New Project

Steps
1. Choose Project
2. ...

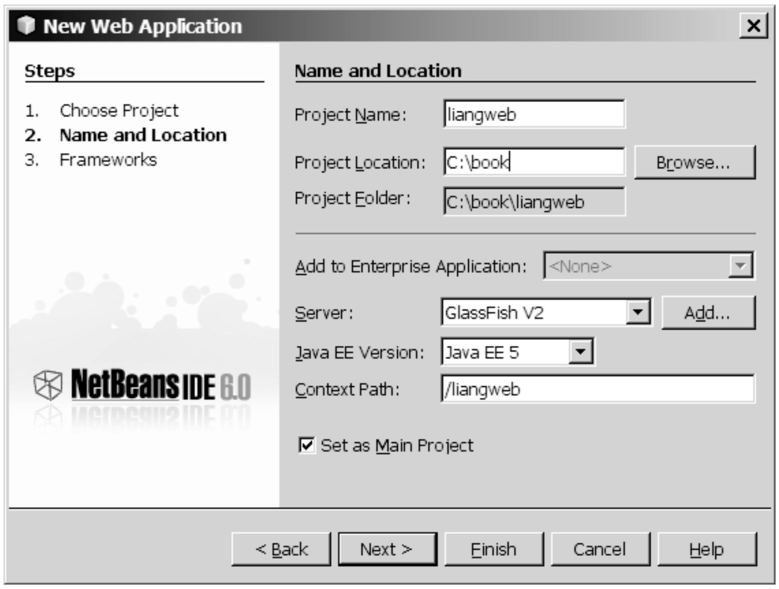
Choose Project

Categories: Java, Web, Enterprise, NetBeans Modules, Samples

Projects: Web Application, Web Application with Existing Sources, Web Application with Existing Ant Script

Description:
Creates an empty Web application in a standard IDE project. A standard project uses an **IDE-generated build script** to build, run, and debug your project.

FIGURE 39.3 Choose Web Application to create a Web project.


New Web Application

Steps
1. Choose Project
2. **Name and Location**
3. Frameworks

Name and Location

Project Name: liangweb
Project Location: C:\book
Project Folder: C:\book\liangweb
Add to Enterprise Application: <None>
Server: GlassFish V2
Java EE Version: Java EE 5
Context Path: /liangweb
☑ Set as Main Project

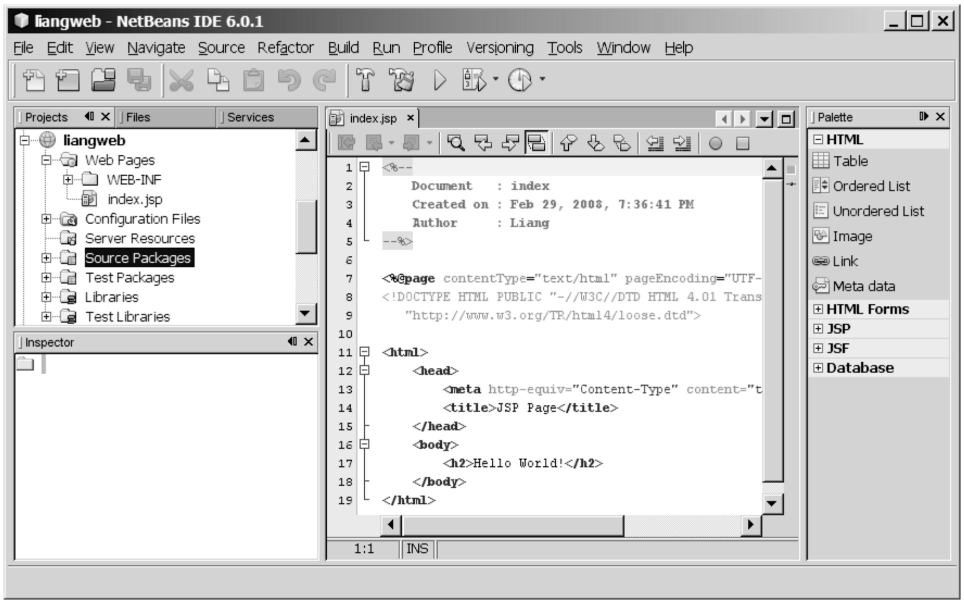
FIGURE 39.4 The New Web Application dialog box enables you to create a new Web project.

2. Enter `liangweb` in the Project Name field and `c:\book` in the Project Location field. Choose GlassFish as the server and Java EE 5 as the Java EE version. Click *Finish* to create the Web project, as shown in Figure 39.5.

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

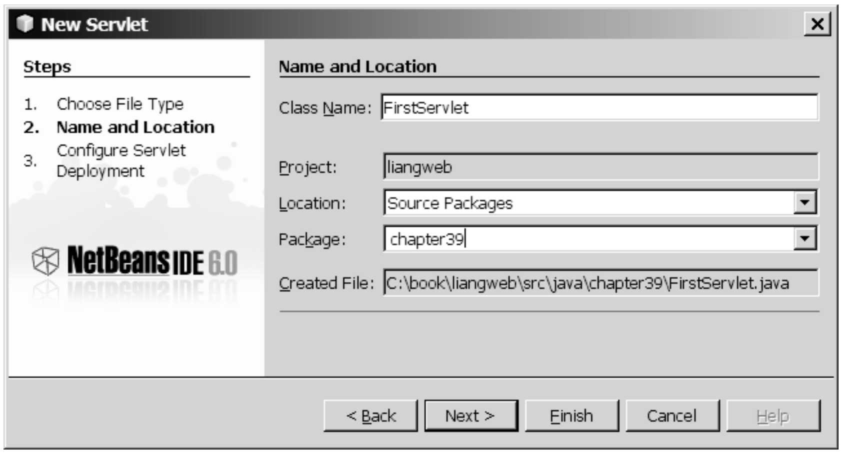
create a servlet

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 39.6.
2. Enter `FirstServlet` in the Class Name field and `chapter39` in the Package field. Click *Finish* to create the servlet. A servlet template is now created in the project, as shown in Figure 39.7.
3. Replace the code in the content pane for the servlet using the code in Listing 39.1.



```
<%--
    Document   : index
    Created on : Feb 29, 2008, 7:36:41 PM
    Author     : Liang
--%>

<%@page contentType="text/html" pageEncoding="UTF-
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Trans
   "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="t
        <title>JSP Page</title>
    </head>
    <body>
        <h2>Hello World!</h2>
    </body>
</html>
```

FIGURE 39.5 A new Web project is created.



Class Name: FirstServlet

Project: liangweb

Location: Source Packages

Package: chapter39

Created File: C:\book\liangweb\src\java\chapter39\FirstServlet.java

FIGURE 39.6 You can create a servlet in the New Servlet dialog box.

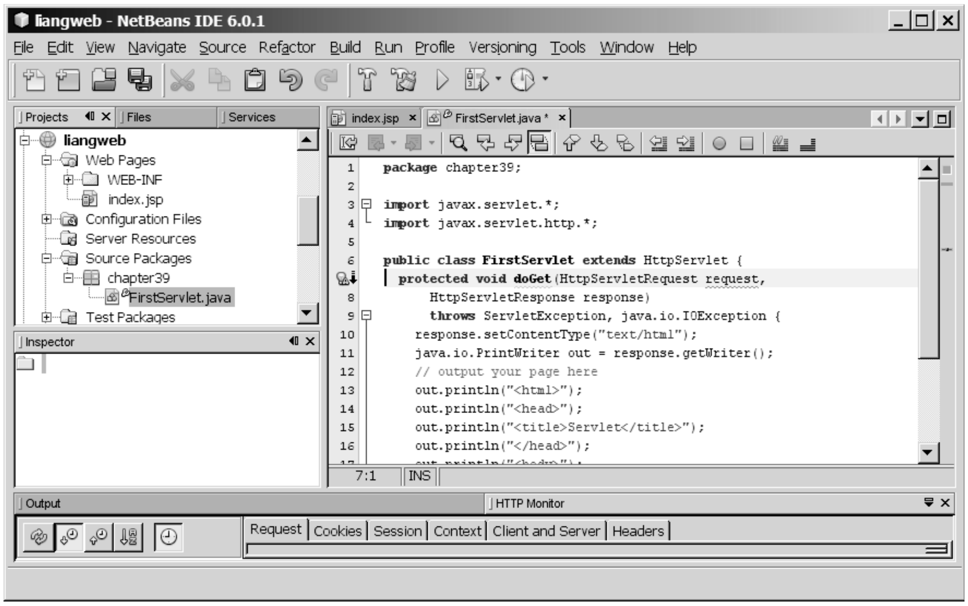
4. Right-click `FirstServlet` node under Source Packages, chapter39, in the liangweb project to display a context menu and choose **Run**. Now you will see a dialog box displayed, prompting you to set execution URI, as shown in Figure 39.8. Click *OK* to continue. You will now see the servlet result displayed in your Web browser, as shown in Figure 39.9.

run a servlet

Tip

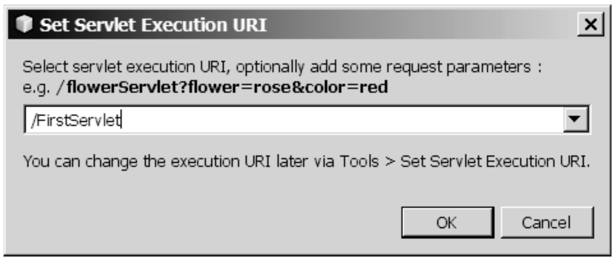
You can deploy a Web application using a Web archive file (WAR) to a Web application server (e.g., Tomcat). To create a WAR file for the liangweb project, right-click liangweb and choose **Build Project**. You can now locate liangweb.war in the `c:\book\liangweb\dist` folder. To deploy on Tomcat, simply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

deploy Web project
WAR file



```
package chapter39;

import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, java.io.IOException {
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    // output your page here
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet</title>");
    out.println("</head>");
```

FIGURE 39.7 A new servlet class is created in the project.

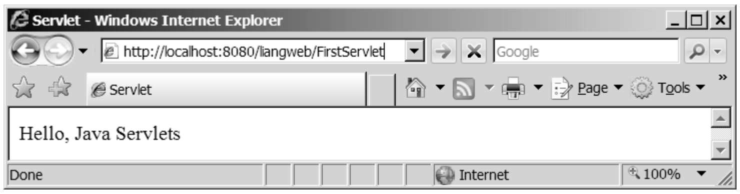


Select servlet execution URI, optionally add some request parameters :
e.g. /**flowerServlet?flower=rose&color=red**

/FirstServlet

You can change the execution URI later via Tools > Set Servlet Execution URI.

FIGURE 39.8 You can add request parameters in the URL string.


Servlet - Windows Internet Explorer
http://localhost:8080/liangweb/FirstServlet
Google
Servlet
Page
Tools
Done
Internet
100%

Hello, Java Servlets

FIGURE 39.9 Servlet result is displayed in a Web browser.

Note

Tomcat Tutorial

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

39.4 The Servlet API

You have to know the servlet API in order to understand the source code 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 39.10. The `javax.servlet` package provides basic interfaces, and the

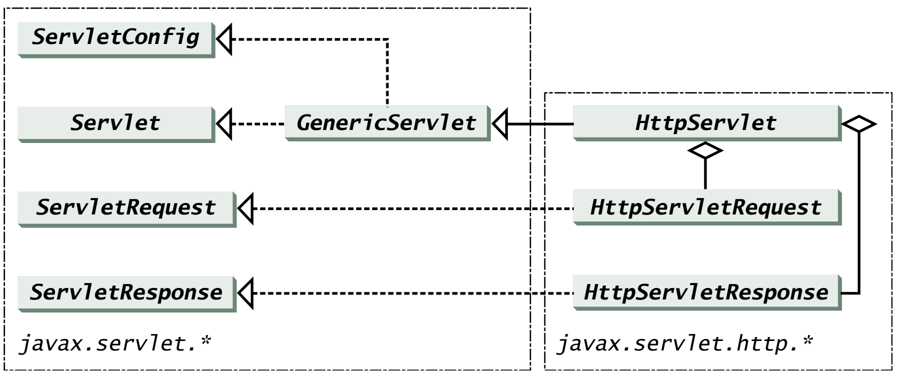


FIGURE 39.10 The servlet API contains interfaces and classes that you use to develop and run servlets.

`javax.servlet.http` package provides classes and interfaces derived from them, which provide specific means for servicing HTTP requests.

39.4.1 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();
```

servlet life cycle

The `init`, `service`, and `destroy` methods are known as *life-cycle methods* and are called in the following sequence (see Figure 39.11):

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.
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.

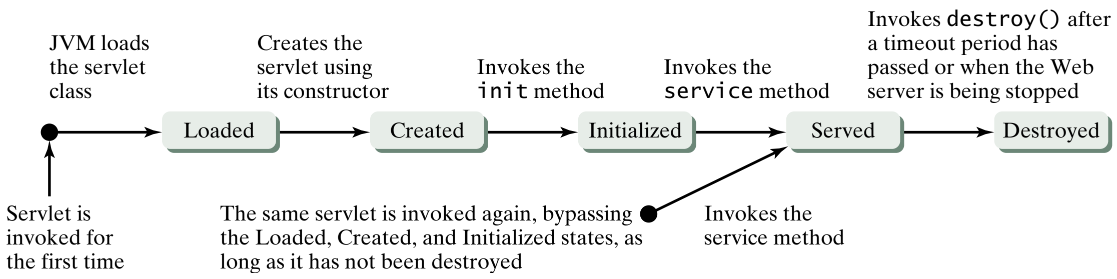

JVM loads the servlet class
Creates the servlet using its constructor
Invokes the init method
Invokes the service method
Invokes destroy() after a timeout period has passed or when the Web server is being stopped
Loaded
Created
Initialized
Served
Destroyed
Servlet is invoked for the first time
The same servlet is invoked again, bypassing the Loaded, Created, and Initialized states, as long as it has not been destroyed
Invokes the service method

FIGURE 39.11 The JVM uses the `init`, `service`, and `destroy` methods to control the servlet.

39.4.2 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 have the same 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 39.12.

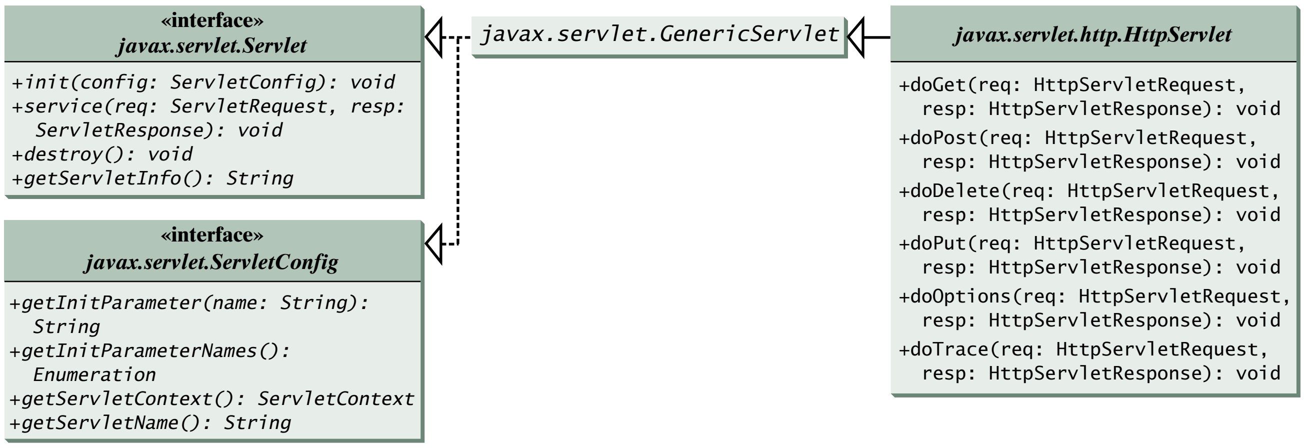

«interface»
javax.servlet.Servlet
+init(config: ServletConfig): void
+service(req: ServletRequest, resp: ServletResponse): void
+destroy(): void
+getServletInfo(): String
«interface»
javax.servlet.ServletConfig
+getInitParameter(name: String): String
+getInitParameterNames(): Enumeration
+getServletContext(): ServletContext
+getServletName(): String
javax.servlet.GenericServlet
javax.servlet.http.HttpServlet
+doGet(req: HttpServletRequest, resp: HttpServletResponse): void
+doPost(req: HttpServletRequest, resp: HttpServletResponse): void
+doDelete(req: HttpServletRequest, resp: HttpServletResponse): void
+doPut(req: HttpServletRequest, resp: HttpServletResponse): void
+doOptions(req: HttpServletRequest, resp: HttpServletResponse): void
+doTrace(req: HttpServletRequest, resp: HttpServletResponse): void

FIGURE 39.12 `HttpServlet` inherits abstract class `GenericServlet`, which implements interfaces `Servlet` and `ServletConfig`.

39.4.3 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 39.13.

39.4.4 The `ServletResponse` Interface and `HttpServletResponse` Interface

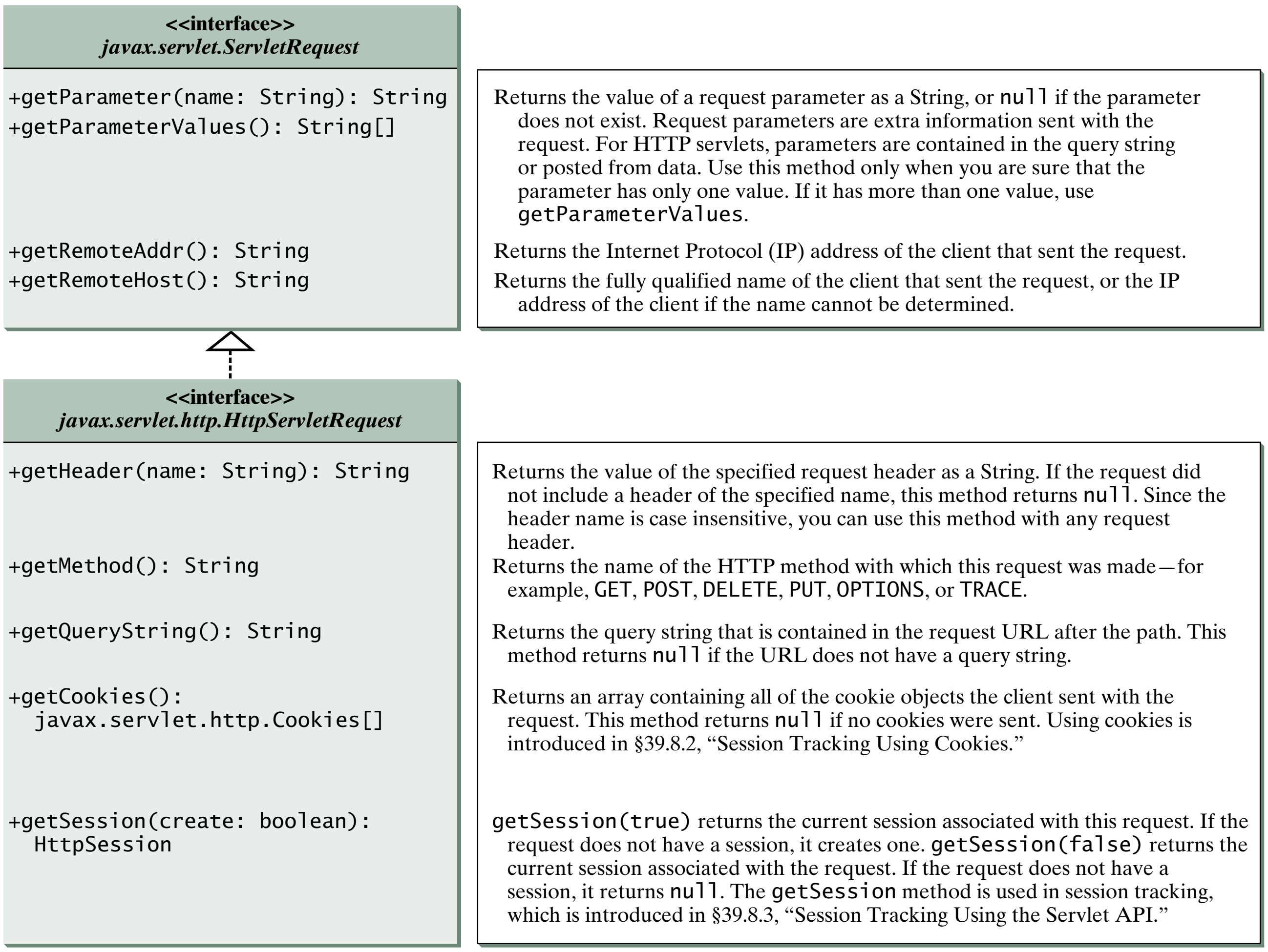
Every `doXxx` method in the `HttpServlet` class has a parameter of the `HttpServletResponse` type, which is an object that assists a servlet in sending a response to the client. `HttpServletResponse` is a subinterface of `ServletResponse`. `ServletResponse` defines a more general interface for sending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

The frequently used methods in these two interfaces are shown in Figure 39.14.

39.5 Creating Servlets

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 server to call the methods. Every servlet has a structure like the one shown below:

```
package chapter39;

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



<<interface>>
javax.servlet.ServletRequest

```
+getParameter(name: String): String
+getParameterValues(): String[]
+getRemoteAddr(): String
+getRemoteHost(): String
```

Returns the value of a request parameter as a String, or `null` if the parameter does not exist. Request parameters are extra information sent with the request. For HTTP servlets, parameters are contained in the query string or posted from data. Use this method only when you are sure that the parameter has only one value. If it has more than one value, use `getParameterValues`.

Returns the Internet Protocol (IP) address of the client that sent the request.

Returns the fully qualified name of the client that sent the request, or the IP address of the client if the name cannot be determined.

<<interface>>
javax.servlet.http.HttpServletRequest

```
+getHeader(name: String): String
+getMethod(): String
+getQueryString(): String
+getCookies():
  javax.servlet.http.Cookies[]
+getSession(create: boolean):
  HttpSession
```

Returns the value of the specified request header as a String. If the request did not include a header of the specified name, this method returns `null`. Since the header name is case insensitive, you can use this method with any request header.

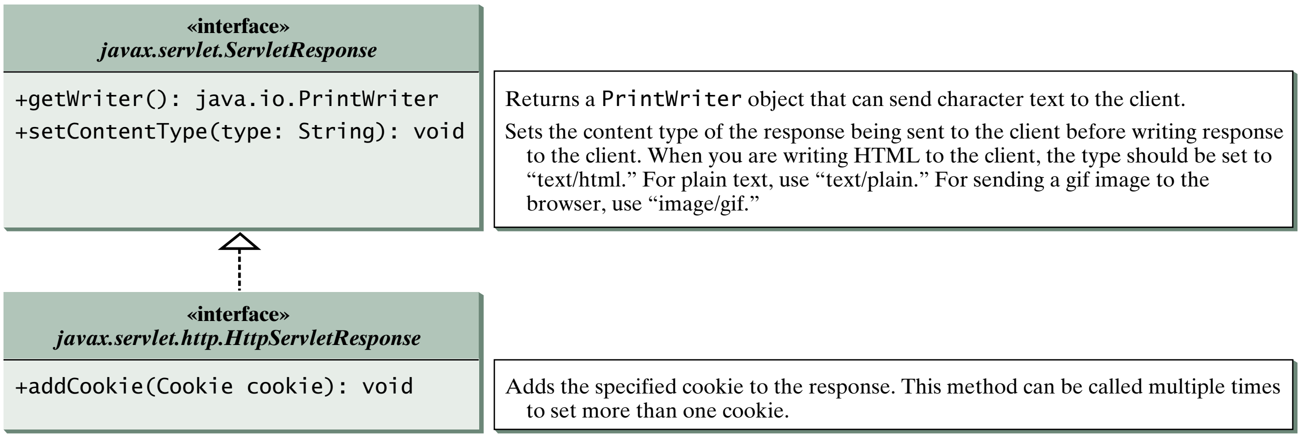
Returns the name of the HTTP method with which this request was made—for example, `GET`, `POST`, `DELETE`, `PUT`, `OPTIONS`, or `TRACE`.

Returns the query string that is contained in the request URL after the path. This method returns `null` if the URL does not have a query string.

Returns an array containing all of the cookie objects the client sent with the request. This method returns `null` if no cookies were sent. Using cookies is introduced in §39.8.2, "Session Tracking Using Cookies."

`getSession(true)` returns the current session associated with this request. If the request does not have a session, it creates one. `getSession(false)` returns the current session associated with the request. If the request does not have a session, it returns `null`. The `getSession` method is used in session tracking, which is introduced in §39.8.3, "Session Tracking Using the Servlet API."

FIGURE 39.13 `HttpServletRequest` is a subinterface of `ServletRequest`.



«interface»
javax.servlet.ServletResponse

```
+getWriter(): java.io.PrintWriter
+setContentType(type: String): void
```

Returns a `PrintWriter` object that can send character text to the client.

Sets the content type of the response being sent to the client before writing response to the client. When you are writing HTML to the client, the type should be set to "text/html." For plain text, use "text/plain." For sending a gif image to the browser, use "image/gif."

«interface»
javax.servlet.http.HttpServletResponse

```
+addCookie(Cookie cookie): void
```

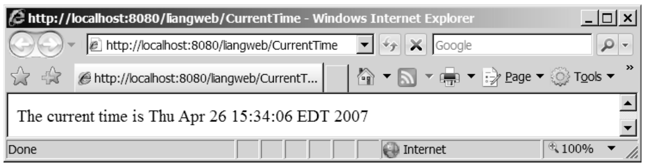
Adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.

FIGURE 39.14 `HttpServletResponse` is a subinterface of `ServletResponse`.

```
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 39.2 gives a simple Java servlet that generates a dynamic Web page for displaying the current time, as shown in Figure 39.15.


http://localhost:8080/liangweb/CurrentTime - Windows Internet Explorer
http://localhost:8080/liangweb/CurrentTime
Google
http://localhost:8080/liangweb/CurrentT...
Page
Tools
Done
Internet
100%

The current time is Thu Apr 26 15:34:06 EDT 2007

FIGURE 39.15 Servlet `CurrentTime` displays the current time.

LISTING 39.2 CurrentTime.java

```
package chapter39;

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

public class CurrentTime extends HttpServlet {
  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse       process GET
      response) throws ServletException, IOException {
    response.setContentType("text/html");                                 content type
    PrintWriter out = response.getWriter();                               output to browser
    out.println("<p>The current time is " + new java.util.Date());
    out.close(); // Close stream                                          close stream
  }
}
```

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.

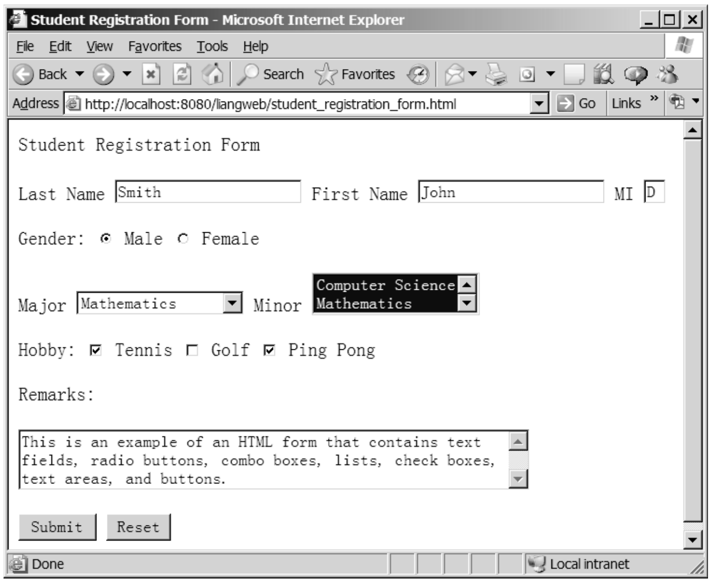
Servlets return responses to the browser through an `HttpServletResponse` object. Since the `setContentType("text/html")` method sets the MIME type to "text/html," the browser will display the response in HTML. The `getWriter` method returns a `PrintWriter` stream (`out`) for sending HTML back to the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

39.6 HTML Forms

HTML forms enable you to submit data to the Web server in a convenient form. As shown in Figure 39.16, the form can contain text fields, text area, check boxes, combo boxes, lists, radio buttons, and buttons.


Student Registration Form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/student_registration_form.html
Go Links
Done
Local intranet

Student Registration Form

Last Name Smith First Name John MI D

Gender: ● Male ○ Female

Major Mathematics Minor Computer Science Mathematics

Hobby: ✓ Tennis ☐ Golf ✓ Ping Pong

Remarks:

This is an example of an HTML form that contains text fields, radio buttons, combo boxes, lists, check boxes, text areas, and buttons.

Submit Reset

FIGURE 39.16 An HTML form may contain text fields, radio buttons, combo boxes, lists, check boxes, text areas, and buttons.

HTML/XHTML Tutorial

The HTML code for creating the form in Figure 39.16 is given in Listing 39.3. (If you are unfamiliar with HTML, please see Supplement V.A, "HTML and XHTML Tutorial.")

LISTING 39.3 Student_Registration_Form.html

```
<!--An HTML Form Demo -->
<html>
  <head>
    <title>Student Registration Form</title>
  </head>
  <body>
    <h3>Student Registration Form</h3>

    <form action = "GetParameters"
      method = "get">
      <!-- Name text fields -->
      <p><label>Last Name</label>
      <input type = "text"name = "lastName"size = "20" />
      <label>First Name</label>
      <input type = "text" name = "firstName" size = "20" />
      <label>MI</label>
      <input type = "text" name = "mi" size = "1" /></p>

      <!-- Gender radio buttons -->
      <p><label>Gender:</label>
      <input type = "radio" name = "gender" value = "M" checked />
        Male
      <input type = "radio" name = "gender" value = "F" /> Female</p>

      <!-- Major combo box -->
      <p><label>Major</label>
        <select name = "major"size = "1">
          <option value = "CS">Computer Science</option>
          <option value = "Math">Mathematics</option>
          <option>English</option>
          <option>Chinese</option>
        </select>

      <!-- Minor list -->
      <label>Minor</label>
        <select name = "minor" size = "2" multiple>
          <option>Computer Science</option>
          <option>Mathematics</option>
          <option>English</option>
          <option>Chinese</option>
        </select></p>

      <!-- Hobby check boxes -->
      <p><label>Hobby:</label>
        <input type = "checkbox" name = "tennis" /> Tennis
        <input type = "checkbox" name = "golf" /> Golf
        <input type = "checkbox" name = "pingPong" checked />Ping Pong
      </p>

      <!-- Remark text area -->
      <p>Remarks:</p>
      <p><textarea name = "remarks" rows = "3" cols = "56">
        </textarea> </p>

      <!-- Submit and Reset buttons -->
      <p><input type = "submit" value = "Submit"/>
      <input type = "reset" value = "Reset"/></p>
    </form>
  </body>
</html>
```

form tag

label

text field

radio button

combo box

list

check box

text area

submit button

reset button

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

`<from>`

`action`

method

on the Web server when the form is submitted. The **method** attribute is either **get** or **post**.

<label>

- **<label> . . . </label>** simply defines a label.

<input>

- **<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> . . . </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> . . . </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> . . . </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.

create an HTML file

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 **Student_Registration_Form** as the file name and click *Finish* to create the file.

39.6.1 Obtaining Parameter Values from HTML Forms

To demonstrate how to obtain parameter values from an HTML form, Listing 39.4 creates a servlet to obtain all the parameter values from the preceding student registration form in Figure 39.16 and display their values, as shown in Figure 39.17.

LISTING 39.4 GetParameters.java

```
package chapter39;

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

public class GetParameters extends HttpServlet {
  /** Process the HTTP Post request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain parameters from the client
    String lastName = request.getParameter("lastName");
```

process GET

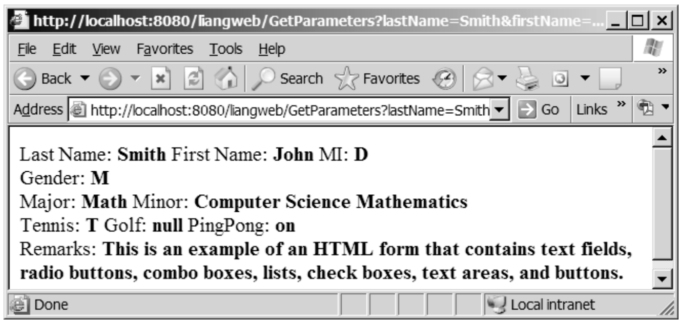
content type

output to browser

get parameters

```
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 }
```

close stream


http://localhost:8080/liangweb/GetParameters?lastName=Smith&firstName=...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/GetParameters?lastName=Smith
Go Links
Done
Local intranet

Last Name: **Smith** First Name: **John** MI: **D**
Gender: **M**
Major: **Math** Minor: **Computer Science Mathematics**
Tennis: **T** Golf: **null** PingPong: **on**
Remarks: **This is an example of an HTML form that contains text fields, radio buttons, combo boxes, lists, check boxes, text areas, and buttons.**

FIGURE 39.17 The servlet displays the parameter values entered in Figure 39.16.

The HTML form is already created in student_registration_form.html and displayed in Figure 39.16. 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 (e.g., `getParameterValues("minor")` in line 20).

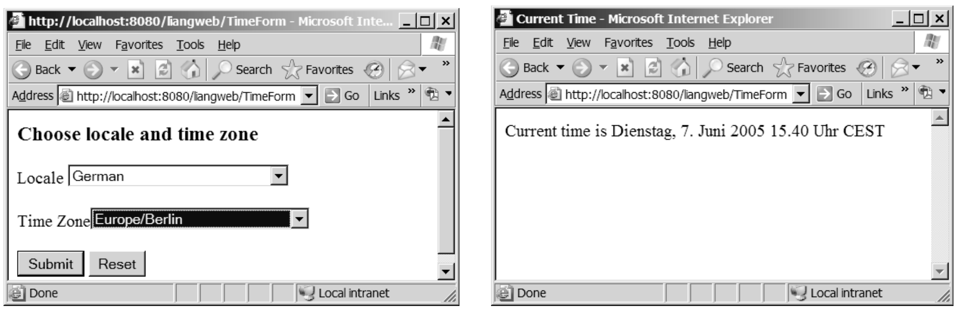
You may optionally specify the `value` attribute in a text field, text area, combo box, list, check box, or radio button in an HTML form. For text field and text area, the `value` attribute specifies a default value to be displayed in the text field and text area. The user can type in new values to replace 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`.

39.6.2 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 39.18(a). 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 39.18(b).


http://localhost:8080/liangweb/TimeForm - Microsoft Inte...
Current Time - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/TimeForm Go Links
Done
Local intranet

Choose locale and time zone

Locale German

Time Zone Europe/Berlin

Submit Reset

Current time is Dienstag, 7. Juni 2005 15.40 Uhr CEST

(a) (b)

FIGURE 39.18 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 39.5 gives the servlet.

LISTING 39.5 `TimeForm.java`

```
package chapter39;

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

public class TimeForm extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  private Locale[] allLocale = Locale.getAvailableLocales();
  private String[] allTimeZone = TimeZone.getAvailableIDs();

  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<h3>Choose locale and time zone</h3>");
    out.println("<form method=\"post\" action=" +
      "TimeForm>");
```

process GET
content type
output to browser
create form

```
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].getDisplayName() + "</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                                 close stream
43   }
44
45   /** Process the HTTP Post request */
46   public void doPost(HttpServletRequest request, HttpServletResponse   process POST
47       response) throws ServletException, IOException {
48     response.setContentType(CONTENT_TYPE);                        content type
49     PrintWriter out = response.getWriter();                       output to browser
50     out.println("<html>");
51     int localeIndex = Integer.parseInt(                           get locale
52       request.getParameter("locale"));
53     String timeZoneID = request.getParameter("timezone");         get time zone
54     out.println("<head><title>Current Time</title></head>");
55     out.println("<body>");
56     Calendar calendar =                                           create calendar
57       new GregorianCalendar(allLocale[localeIndex]);
58     TimeZone timeZone = TimeZone.getTimeZone(timeZoneID);
59     DateFormat dateFormat = DateFormat.getDateTimeInstance(
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                                 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

set character encoding

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, add the following statement in line 48 to set an international character encoding:

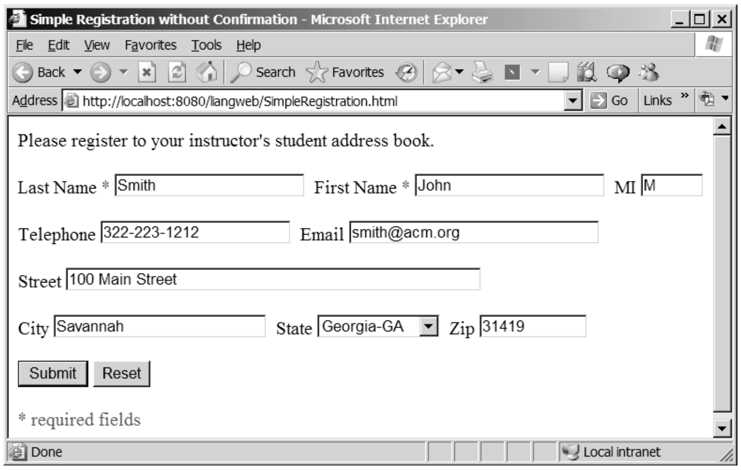
```
response.setCharacterEncoding("GB18030");
```

For information on encoding, see §31.6, "Character Encoding."

39.7 Database Programming in Servlets

Many dynamic Web applications use databases to store and manage data. Servlets can connect to any relational database via JDBC. In Chapter 37, "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 39.19. The result of the submission is shown in Figure 39.20. The server collects the data from the form and stores them in a database.


Simple Registration without Confirmation - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/SimpleRegistration.html
Go Links
Done
Local intranet

Please register to your instructor's student address book.

Last Name * Smith First Name * John MI M

Telephone 322-223-1212 Email smith@acm.org

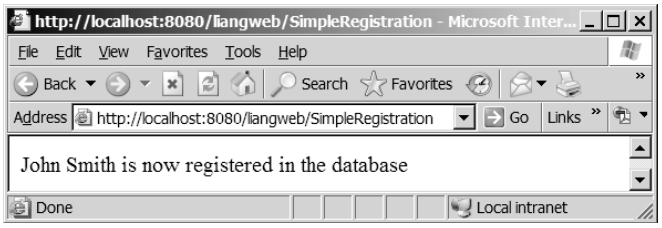
Street 100 Main Street

City Savannah State Georgia-GA Zip 31419

Submit Reset

* required fields

FIGURE 39.19 The HTML form enables the user to enter student information.


http://localhost:8080/liangweb/SimpleRegistration - Microsoft Inter...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/SimpleRegistration
Go Links
Done
Local intranet

John Smith is now registered in the database

FIGURE 39.20 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 37. 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 ojdbc14.jar for Oracle) into libaries node in the project.

mysqljdbc.jar
ojdbc14.jar
place .html file

Create an HTML file named SimpleRegistration.html in Listing 39.6 for collecting the data and sending them to the database using the post method.

LISTING 39.6 SimpleRegistration.html

```
<!-- SimpleRegistration.html -->
<html>
  <head>
    <title>Simple Registration without Confirmation</title>
  </head>
  <body>
    Please register to your instructor's student address book.

    <form method = "post" action = "SimpleRegistration">
      <p>Last Name <font color = "#FF0000">*</font>
         <input type = "text" name = "lastName"> &nbsp;
         First Name <font color = "#FF0000">*</font>
         <input type = "text" name = "firstName"> &nbsp;
         MI <input type = "text" name = "mi" size = "3">
      </p>
      <p>Telephone
         <input type = "text" name = "telephone" size = "20"> &nbsp;
         Email
         <input type = "text" name = "email" size = "28"> &nbsp;
      </p>
      <p>Street <input type = "text" name = "street" size = "50">
      </p>
      <p>City <input type = "text" name = "city" size = "23"> &nbsp;
         State
         <select size = "1" name = "state">
           <option value = "GA">Georgia-GA</option>
           <option value = "OK">Oklahoma-OK</option>
           <option value = "IN">Indiana-IN</option>
         </select> &nbsp;
         Zip <input type = "text" name = "zip" size = "9">
      </p>
      <p><input type = "submit" name = "Submit" value = "Submit">
```

action

submit form

```
        <input type = "reset" value = "Reset">
      </p>
    </form>
    <p><font color = "#FF0000">* required fields</font></p>
  </body>
</html>
```

place .class file

Create the servlet named `SimpleRegistration` in Listing 39.7.

LISTING 39.7 SimpleRegistration.java

initialize db · process POST · content type · output to browser · get parameters · store record

```
package chapter39;

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

public class SimpleRegistration extends HttpServlet {
  // Use a prepared statement to store a student into the database
  private PreparedStatement pstmt;

  /** Initialize variables */
  public void init() throws ServletException {
    initializeJdbc();
  }

  /** Process the HTTP Post request */
  public void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain parameters from the client
    String lastName = request.getParameter("lastName");
    String firstName = request.getParameter("firstName");
    String mi = request.getParameter("mi");
    String phone = request.getParameter("telephone");
    String email = request.getParameter("email");
    String address = request.getParameter("street");
    String city = request.getParameter("city");
    String state = request.getParameter("state");
    String zip = request.getParameter("zip");

    try {
      if (lastName.length() == 0 || firstName.length() == 0) {
        out.println("Last Name and First Name are required");
      }
      else {
        storeStudent(lastName, firstName, mi, phone, email,
          address, city, state, zip);

        out.println(firstName + " " + lastName +
          " is now registered in the database");
      }
    }
    catch(Exception ex) {
      out.println("Error: " + ex.getMessage());
    }
49       finally {
50         out.close(); // Close stream                                      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                    load driver
63           ("jdbc:mysql://localhost/javabook" , "scott", "tiger");        connect db
64         System.out.println("Database connected");
65
66         // Create a Statement
67         pstmt = conn.prepareStatement("insert into Address " +           prepare statement
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);                                      set values
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();                                             execute SQL
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. 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 loads a MySQL JDBC driver (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 35–37). Otherwise, the servlet stores the data in the database using the prepared statement.

39.8 Session Tracking

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 by resubmitting them.

In Request 2, the data submitted in Request 1 are sent back to the server. 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*.

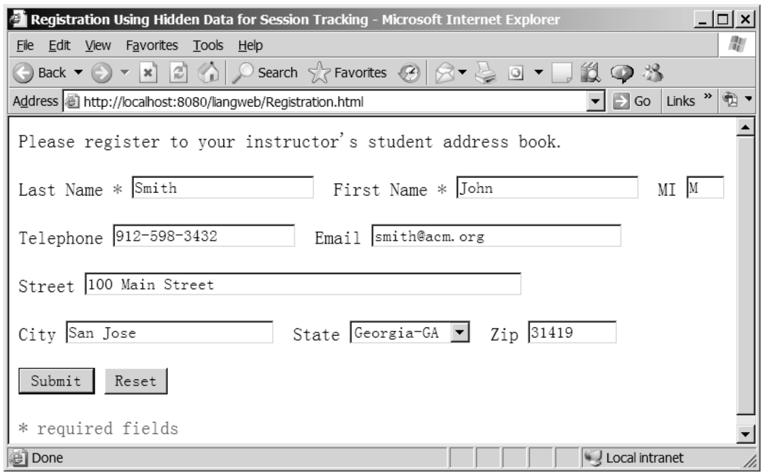
39.8.1 Session Tracking Using Hidden Values

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 39.21. The server collects the data in the form, displays them to the client, and asks the client for confirmation, as shown in Figure 39.22. 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.


Registration Using Hidden Data for Session Tracking - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/Registration.html
Go Links

Please register to your instructor's student address book.

Last Name * Smith First Name * John MI M

Telephone 912-598-3432 Email smith@acm.org

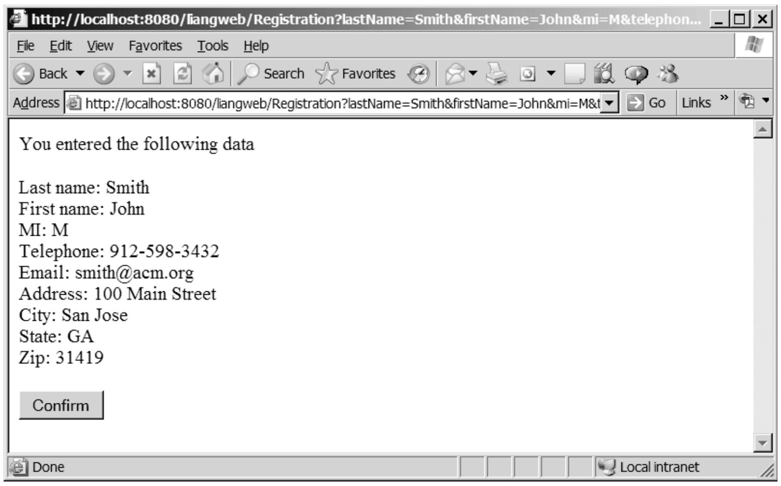
Street 100 Main Street

City San Jose State Georgia-GA Zip 31419

Submit Reset

* required fields

FIGURE 39.21 The registration form collects user information.


http://localhost:8080/liangweb/Registration?lastName=Smith&firstName=John&mi=M&telephon...

You entered the following data

Last name: Smith
First name: John
MI: M
Telephone: 912-598-3432
Email: smith@acm.org
Address: 100 Main Street
City: San Jose
State: GA
Zip: 31419

Confirm

File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/Registration?lastName=Smith&firstName=John&mi=M&t
Go Links
Done
Local intranet

FIGURE 39.22 The servlet asks the client for confirmation of the input.

Create an HTML form named Registration.html in Listing 39.8 for collecting the data and sending it to the database using the GET method for confirmation. This file is almost identical to Listing 39.6, SimpleRegistration.html.

LISTING 39.8 Registration.html

```
<!-- Registration.html -->
<html>
  <head>
    <title>Using Hidden Data for Session Tracking</title>
  </head>
  <body>
    Please register to your instructor's student address book.

    <form method = "get" action = "Registration">                action
      <p>Last Name <font color = "#FF0000">*</font>
         <input type = "text" name = "lastName"> &nbsp;
         First Name <font color = "#FF0000">*</font>
         <input type = "text" name = "firstName"> &nbsp;
         MI <input type = "text" name = "mi" size = "3">
      </p>
      <p>Telephone
         <input type = "text" name = "telephone" size = "20"> &nbsp;
         Email
         <input type = "text" name = "email" size = "28"> &nbsp;
      </p>
      <p>Street <input type = "text" name = "street" size = "50">
      </p>
      <p>City <input type = "text" name = "city" size = "23"> &nbsp;
         State
         <select size = "1" name = "state">
           <option value = "GA">Georgia-GA</option>
           <option value = "OK">Oklahoma-OK</option>
           <option value = "IN">Indiana-IN</option>
         </select> &nbsp;
         Zip <input type = "text" name = "zip" size = "9">
      </p>
      <p><input type = "submit" name = "Submit" value = "Submit">    submit form
         <input type = "reset" value = "Reset">
      </p>
    </form>
    <p><font color = "#FF0000">* required fields</font></p>
  </body>
</html>
```

Create the servlet named Registration in Listing 39.9.

LISTING 39.9 Registration.java

```
package chapter39;

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

public class Registration extends HttpServlet {
  // Use a prepared statement to store a student into the database
  private PreparedStatement pstmt;

  /** Initialize variables */
  public void init() throws ServletException {
    initializeJdbc();
  }

  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain data from the form
    String lastName = request.getParameter("lastName");
    String firstName = request.getParameter("firstName");
    String mi = request.getParameter("mi");
    String telephone = request.getParameter("telephone");
    String email = request.getParameter("email");
    String street = request.getParameter("street");
    String city = request.getParameter("city");
    String state = request.getParameter("state");
    String zip = request.getParameter("zip");

    if (lastName.length() == 0 || firstName.length() == 0) {
      out.println("Last Name and First Name are required");
    }
    else {
      // Ask for confirmation
      out.println("You entered the following data");
      out.println("<p>Last name: " + lastName);
      out.println("<br>First name: " + firstName);
      out.println("<br>MI: " + mi);
      out.println("<br>Telephone: " + telephone);
      out.println("<br>Email: " + email);
      out.println("<br>Address: " + street);
      out.println("<br>City: " + city);
      out.println("<br>State: " + state);
      out.println("<br>Zip: " + zip);

      // Set the action for processing the answers
      out.println("<p><form method=\"post\" action=" +
```

initialize db

process GET

content type

output to browser

get parameters

client verification

```
        "Registration>");
      // Set hidden values
      out.println("<p><input type=\"hidden\" " +
        "value=" + lastName + " name=\"lastName\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + firstName + " name=\"firstName\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + mi + " name=\"mi\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + telephone + " name=\"telephone\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + email + " name=\"email\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + street + " name=\"street\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + city + " name=\"city\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + state + " name=\"state\">");
      out.println("<p><input type=\"hidden\" " +
        "value=" + zip + " name=\"zip\">");
      out.println("<p><input type=\"submit\" value=\"Confirm\" >");
      out.println("</form>");
    }

    out.close(); // Close stream
  }

  /** Process the HTTP Post request */
  public void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    try {
      String lastName = request.getParameter("lastName");
      String firstName = request.getParameter("firstName");
      String mi = request.getParameter("mi");
      String telephone = request.getParameter("telephone");
      String email = request.getParameter("email");
      String street = request.getParameter("street");
      String city = request.getParameter("city");
      String state = request.getParameter("state");
      String zip = request.getParameter("zip");

      storeStudent(lastName, firstName, mi, telephone, email,
        street, city, state, zip);

      out.println(firstName + " " + lastName +
        " is now registered in the database");
    }
    catch(Exception ex) {
      out.println("Error: " + ex.getMessage());
    }
  }

  /** Initialize database connection */
  private void initializeJdbc() {
    try {
      // Load the JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
```

process POST

content type

output to browser

get parameters

```
      System.out.println("Driver loaded");

      // Establish a connection
      Connection conn = DriverManager.getConnection
        ("jdbc:mysql://localhost/javabook" , "scott", "tiger");
      System.out.println("Database connected");

      // Create a Statement
      pstmt = conn.prepareStatement("insert into Address " +
        "(lastName, firstName, mi, telephone, email, street, city, "
        + "state, zip) values (?, ?, ?, ?, ?, ?, ?, ?, ?)");
    }
    catch (Exception ex) {
      System.out.println(ex);
    }
  }

  /** Store a student record to the database */
  private void storeStudent(String lastName, String firstName,
      String mi, String phone, String email, String address,
      String city, String state, String zip) throws SQLException {
    pstmt.setString(1, lastName);
    pstmt.setString(2, firstName);
    pstmt.setString(3, mi);
    pstmt.setString(4, phone);
    pstmt.setString(5, email);
    pstmt.setString(6, address);
    pstmt.setString(7, city);
    pstmt.setString(8, state);
    pstmt.setString(9, zip);
    pstmt.executeUpdate();
  }
}
```

store record

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.

39.8.2 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 39.23.

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 be sent only 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 39.23 `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 39.9, Registration.java. Instead of using hidden values for session tracking, it uses cookies.

Create the servlet named RegistrationWithHttpCookie in Listing 39.10. Create an HTML file named RegistrationWithCookie.html that is identical to Registration.html except that the action is replaced by `RegistrationWithCookie`.

LISTING 39.10 RegistrationWithCookie.java

```java
package chapter39;

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

public class RegistrationWithCookie extends HttpServlet {
  private static final String CONTENT_TYPE = "text/html";
  // Use a prepared statement to store a student into the database
  private PreparedStatement pstmt;

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

process GET

get parameters

create cookies

add cookies

client verification

```
    initializeJdbc();
  }

  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain data from the form
    String lastName = request.getParameter("lastName");
    String firstName = request.getParameter("firstName");
    String mi = request.getParameter("mi");
    String telephone = request.getParameter("telephone");
    String email = request.getParameter("email");
    String street = request.getParameter("street");
    String city = request.getParameter("city");
    String state = request.getParameter("state");
    String zip = request.getParameter("zip");

    if (lastName.length() == 0 || firstName.length() == 0) {
      out.println("Last Name and First Name are required");
    }
    else {
      // Create cookies and send cookies to browsers
      Cookie cookieLastName = new Cookie("lastName", lastName);
      // cookieLastName.setMaxAge(1000);
      response.addCookie(cookieLastName);
      Cookie cookieFirstName = new Cookie("firstName", firstName);
      response.addCookie(cookieFirstName);
      // cookieFirstName.setMaxAge(0);
      Cookie cookieMi = new Cookie("mi", mi);
      response.addCookie(cookieMi);
      Cookie cookieTelephone = new Cookie("telephone", telephone);
      response.addCookie(cookieTelephone);
      Cookie cookieEmail = new Cookie("email", email);
      response.addCookie(cookieEmail);
      Cookie cookieStreet = new Cookie("street", street);
      response.addCookie(cookieStreet);
      Cookie cookieCity = new Cookie("city", city);
      response.addCookie(cookieCity);
      Cookie cookieState = new Cookie("state", state);
      response.addCookie(cookieState);
      Cookie cookieZip = new Cookie("zip", zip);
      response.addCookie(cookieZip);

      // Ask for confirmation
      out.println("You entered the following data");
      out.println("<p>Last name: " + lastName);
      out.println("<br>First name: " + firstName);
      out.println("<br>MI: " + mi);
      out.println("<br>Telephone: " + telephone);
      out.println("<br>Email: " + email);
      out.println("<br>Street: " + street);
      out.println("<br>City: " + city);
      out.println("<br>State: " + state);
      out.println("<br>Zip: " + zip);

      // Set the action for processing the answers
      out.println("<p><form method=\"post\" action=" +
        "RegistrationWithCookie>");
      out.println("<p><input type=\"submit\" value=\"Confirm\" >");
      out.println("</form>");
    }

    out.close(); // Close stream
  }

  /** Process the HTTP Post request */
  public void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();

    String lastName = "";
    String firstName = "";
    String mi = "";
    String telephone = "";
    String email = "";
    String street = "";
    String city = "";
    String state = "";
    String zip = "";

    // Read the cookies
    Cookie[] cookies = request.getCookies();

    // Get cookie values
    for (int i = 0; i < cookies.length; i++) {
      if (cookies[i].getName().equals("lastName"))
        lastName = cookies[i].getValue();
      else if (cookies[i].getName().equals("firstName"))
        firstName = cookies[i].getValue();
      else if (cookies[i].getName().equals("mi"))
        mi = cookies[i].getValue();
      else if (cookies[i].getName().equals("telephone"))
        telephone = cookies[i].getValue();
      else if (cookies[i].getName().equals("email"))
        email = cookies[i].getValue();
      else if (cookies[i].getName().equals("street"))
        street = cookies[i].getValue();
      else if (cookies[i].getName().equals("city"))
        city = cookies[i].getValue();
      else if (cookies[i].getName().equals("state"))
        state = cookies[i].getValue();
      else if (cookies[i].getName().equals("zip"))
        zip = cookies[i].getValue();
    }

    try {
      storeStudent(lastName, firstName, mi, telephone, email, street,
        city, state, zip);

      out.println(firstName + " " + lastName +
        " is now registered in the database");

      out.close(); // Close stream
    }
    catch(Exception ex) {
      out.println("Error: " + ex.getMessage());
```

process POST

get cookies

store record

```
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).

39.8.3 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 Web site 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.

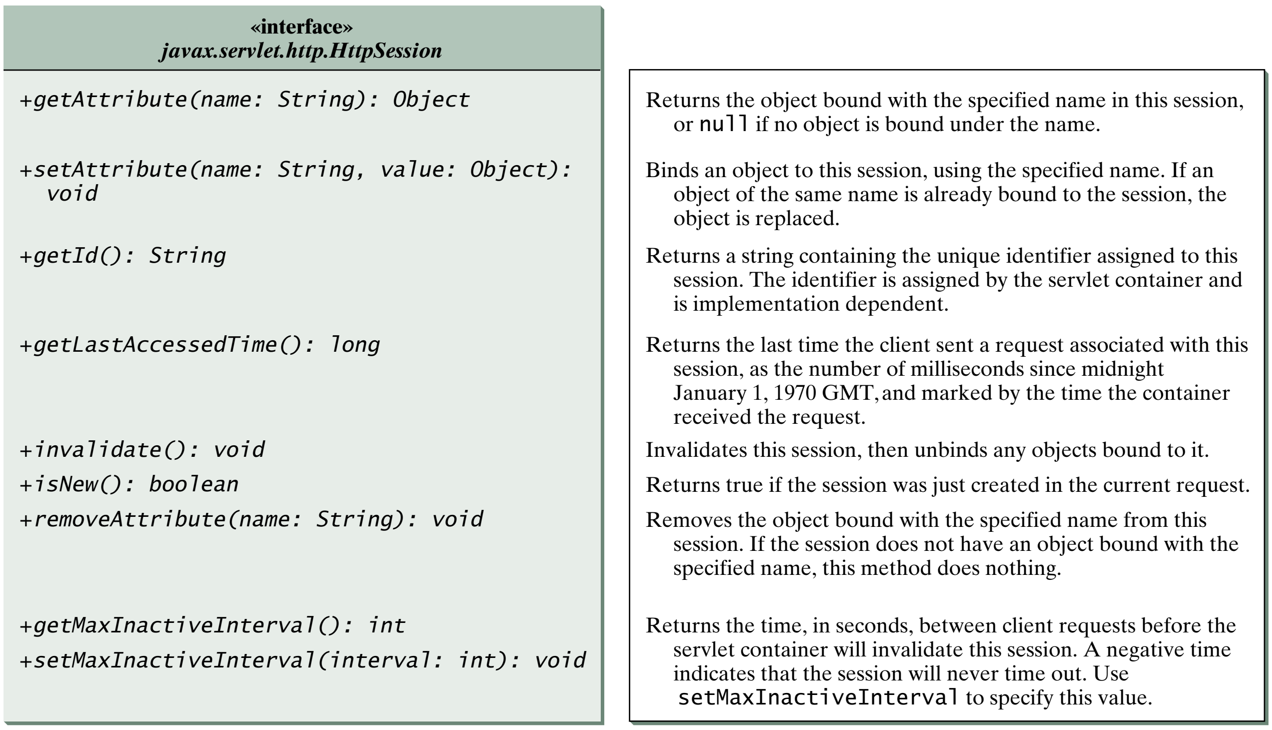
The **`HttpSession`** interface provides the methods for reading and storing data to the session, and for manipulating the session, as shown in Figure 39.24.

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 39.9, **`Registration.java`**, and Listing 39.10, **`RegistrationWithCookie.java`**. Instead of using hidden values or cookies for session tracking, it uses servlet **`HttpSession`**.

Create the servlet named **`RegistrationWithHttpSession`** in Listing 39.11. Create an HTML file named **`RegistrationWithHttpSession.html`** that is identical to **`Registration.html`** except that the action is replaced by **`RegistrationWithHttpSession`**.



«interface»
javax.servlet.http.HttpSession

+getAttribute(name: String): Object	Returns the object bound with the specified name in this session, or `null` if no object is bound under the name.
+setAttribute(name: String, value: Object): void	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.
+getId(): String	Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.
+getLastAccessedTime(): long	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.
+invalidate(): void	Invalidates this session, then unbinds any objects bound to it.
+isNew(): boolean	Returns true if the session was just created in the current request.
+removeAttribute(name: String): void	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.
+getMaxInactiveInterval(): int +setMaxInactiveInterval(interval: int): void	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 `setMaxInactiveInterval` to specify this value.

FIGURE 39.24 `HttpSession` establishes a persistent session between a client with multiple requests and the server.

LISTING 39.11 `RegistrationWithHttpSession.java`

process GET

get parameters

```java
package chapter39;

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

public class RegistrationWithHttpSession extends HttpServlet {
  // Use a prepared statement to store a student into the database
  private PreparedStatement pstmt;

  /** Initialize variables */
  public void init() throws ServletException {
    initializeJdbc();
  }

  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    // Set response type and output stream to the browser
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain data from the form
    String lastName = request.getParameter("lastName");
    String firstName = request.getParameter("firstName");
    String mi = request.getParameter("mi");
    String telephone = request.getParameter("telephone");
    String email = request.getParameter("email");
    String street = request.getParameter("street");
    String city = request.getParameter("city");
    String state = request.getParameter("state");
    String zip = request.getParameter("zip");

    if (lastName.length() == 0 || firstName.length() == 0) {
      out.println("Last Name and First Name are required");
    }
    else {
      // Create an Address object
      Address address = new Address();
      address.setLastName(lastName);
      address.setFirstName(firstName);
      address.setMi(mi);
      address.setTelephone(telephone);
      address.setEmail(email);
      address.setStreet(street);
      address.setCity(city);
      address.setState(state);
      address.setZip(zip);

      // Get an HttpSession or create one if it does not exist
      HttpSession httpSession = request.getSession();

      // Store student object to the session
      httpSession.setAttribute("address", address);

      // Ask for confirmation
      out.println("You entered the following data");
      out.println("<p>Last name: " + lastName);
      out.println("<p>First name: " + firstName);
      out.println("<p>MI: " + mi);
      out.println("<p>Telephone: " + telephone);
      out.println("<p>Email: " + email);
      out.println("<p>Address: " + street);
      out.println("<p>City: " + city);
      out.println("<p>State: " + state);
      out.println("<p>Zip: " + zip);

      // Set the action for processing the answers
      out.println("<p><form method=\"post\" action=" +
        "RegistrationWithHttpSession>");
      out.println("<p><input type=\"submit\" value=\"Confirm\" >");
      out.println("</form>");
    }

    out.close(); // Close stream
  }

  /** Process the HTTP Post request */
  public void doPost(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    // Set response type and output stream to the browser
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtain the HttpSession
```

create address

create session

set attribute

process POST

get session

get address

store address

```
 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:

create **`Address`** class

1. Choose *New*, *Java Class* from the context menu of the `liangweb` node in the project pane to display the New Java Class dialog box.
2. Enter `Address` as the Class Name and `chapter39` as the package name. Click *Finish* to create the class.
3. Enter the code, as shown in Listing 39.12.

LISTING 39.12 Address.java

package chapter39

class Address

```
package chapter39;

public class Address {
  private String firstName;
  private String mi;
  private String lastName;
  private String telephone;
  private String street;
  private String city;
  private String state;
  private String email;
  private String zip;

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

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

  public String getMi() {
    return this.mi;
  }

  public void setMi(String mi) {
    this.mi = mi;
  }

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

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

  public String getTelephone() {
    return this.telephone;
  }

  public void setTelephone(String telephone) {
    this.telephone = telephone;
  }

  public String getEmail() {
    return this.email;
  }

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

  public String getStreet()  {
    return this.street;
  }

  public void setStreet(String street)  {
    this.street = street;
  }

  public String getCity() {
    return this.city;
  }

  public void setCity(String city) {
    this.city = city;
  }

  public String getState() {
    return this.state;
  }

  public void setState(String state) {
    this.state = state;
  }

  public String getZip() {
    return this.zip;
  }

  public void setZip(String zip) {
    this.zip = zip;
  }
}
```

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

39.9 Sending Images from Servlets

So far you have learned how to write Java servlets that generate dynamic HTML text. Java servlets are not limited to sending text to a browser. They can return images on demand. The images can be stored in files or created from programs.

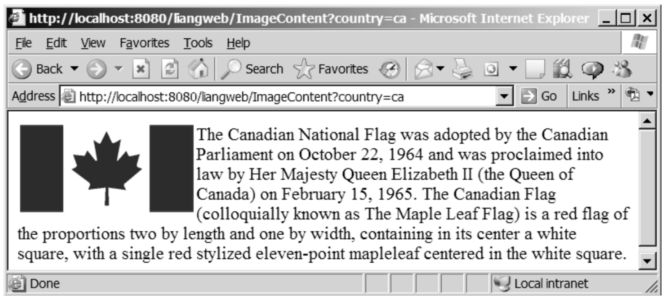
39.9.1 Sending Image from Files

You can use the HTML <img> tag to send images from files. The syntax for the tag is:

```
<img src = URL alt = text align = [top | middle | bottom | texttop]>
```

The attribute `src` specifies the source of the image. The attribute `alt` specifies an alternative text to be displayed in case the image cannot be displayed on the browser. The attribute `align` tells the browser where to place the image.

To demonstrate getting images from a file in a servlet, let us create a servlet that dynamically generates the flag of a country and a text that describes the flag, as shown in Figure 39.25. The flag is stored in an image file, and the text that describes the flag is stored in a text file.


http://localhost:8080/liangweb/ImageContent?country=ca - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/ImageContent?country=ca
Go Links
Done
Local intranet

The Canadian National Flag was adopted by the Canadian Parliament on October 22, 1964 and was proclaimed into law by Her Majesty Queen Elizabeth II (the Queen of Canada) on February 15, 1965. The Canadian Flag (colloquially known as The Maple Leaf Flag) is a red flag of the proportions two by length and one by width, containing in its center a white square, with a single red stylized eleven-point mapleleaf centered in the white square.

FIGURE 39.25 The servlet returns an image along with the text.

Create the servlet named `ImageContent` in Listing 39.13.

LISTING 39.13 ImageContent.java

```
package chapter39;

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

public class ImageContent extends HttpServlet {
  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

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

    out.println("<img src = \"images/" + country + ".gif"
      + "\" align=left>");

    // Read description from a file and send it to the browser
    java.util.Scanner input = new java.util.Scanner(
            new File("c:\\book\\" + country + ".txt"));

```

image tag

read file

```
23     // Read a line from the text file and send it to the browser
24     while (input.hasNext()) {
25       out.println(input.nextLine());
26     }
27
28     out.close();
29   }
30 }
```

image file location

You should create a directory **C:\book\liangweb\web\images** and store image files in this directory.

The **country** parameter determines which image file and text file are displayed. The servlet sends the HTML contents to the browser. The contents contain an **<img>** tag (lines 16–17) that references to the image file.

The servlet reads the characters from the text file and sends them to the browser (lines 20–26).

39.9.2 Sending Images from the **Image** Object

The preceding example displays an image stored in an image file. You can also display an image dynamically created in the program.

Before the image is sent to a browser, it must be encoded into a format acceptable to the browser. Image encoders are not part of Java API, but several free encoders are available. One of them is the **GifEncoder** class (http://www.acme.com/java/software/Acme.JPM.Encoders.GifEncoder.html), which is included in \book\lib\acme.jar. Use the following statement to encode and send the image to the browser:

```
new GifEncoder(image, out, true).encode();
```

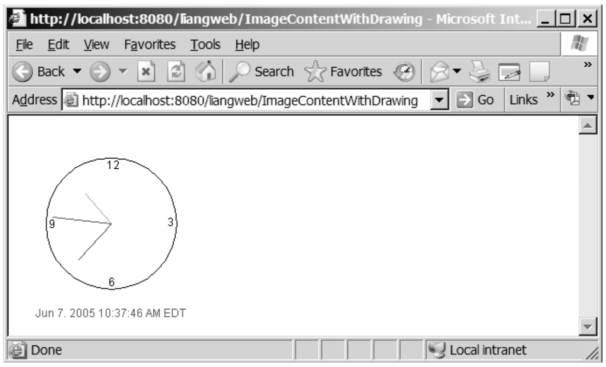
where **out** is a binary output stream from the servlet to the browser, which can be obtained using the following statement:

```
    OutputStream out = response.getOutputStream();
```

To demonstrate dynamically generating images from a servlet, let us create a servlet that displays a clock to show the current time, as shown in Figure 39.26.

acme.jar

Create the servlet named **ImageContentWithDrawing** in Listing 39.14. Add acme.jar in the Libraries node in the **liangweb** project in NetBeans. (acme.jar is in c:\book\lib.)


http://localhost:8080/liangweb/ImageContentWithDrawing - Microsoft Int...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/ImageContentWithDrawing Go Links
12
3
6
9
Done
Local intranet

Jun 7, 2005 10:37:46 AM EDT

FIGURE 39.26 The servlet returns a clock that displays the current time.

LISTING 39.14 ImageContentWithDrawing.java

```
package chapter39;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import java.text.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import Acme.JPM.Encoders.GifEncoder;                              import GifEncoder

public class ImageContentWithDrawing extends HttpServlet {
  /** Initialize variables */
  private final static int width = 300;
  private final static int height = 300;

  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse   process GET
      response) throws ServletException, IOException {
    response.setContentType("image/gif");                           gif type
    OutputStream out = response.getOutputStream();

    // Create image
    Image image = new BufferedImage(width, height,                  image
      BufferedImage.TYPE_INT_ARGB);

    // Get Graphics context of the image
    Graphics g = image.getGraphics();                               graphics

    drawClock(g); // Draw a clock on graphics                       draw graphics

    // Encode the image and send to the output stream
    new GifEncoder(image, out, true).encode();

    out.close(); // Close stream                                    close stream
  }

  private void drawClock(Graphics g) {                              draw clock
    // Initialize clock parameters
    int clockRadius =
      (int)(Math.min(width, height) * 0.7 * 0.5);
    int xCenter = (width) / 2;
    int yCenter = (height) / 2;

    // Draw circle
    g.setColor(Color.black);
    g.drawOval(xCenter - clockRadius,yCenter - clockRadius,
      2 * clockRadius, 2 * clockRadius);
    g.drawString("12", xCenter - 5, yCenter - clockRadius + 12);
    g.drawString("9", xCenter – clockRadius + 3, yCenter + 5);
    g.drawString("3", xCenter + clockRadius - 10, yCenter + 3);
    g.drawString("6", xCenter - 3, yCenter + clockRadius - 3);

    // Get current time using GregorianCalendar
    TimeZone timeZone = TimeZone.getDefault();
    GregorianCalendar cal = new GregorianCalendar(timeZone);
57
58     // Draw second hand
59     int second = (int)cal.get(GregorianCalendar.SECOND);
60     int sLength = (int)(clockRadius * 0.9);
61     int xSecond = (int)(xCenter + sLength * Math.sin(second *
62       (2 * Math.PI / 60)));
63     int ySecond = (int)(yCenter - sLength * Math.cos(second *
64       (2 * Math.PI / 60)));
65     g.setColor(Color.red);
66     g.drawLine(xCenter, yCenter, xSecond, ySecond);
67
68     // Draw minute hand
69     int minute = (int)cal.get(GregorianCalendar.MINUTE);
70     int mLength = (int)(clockRadius * 0.75);
71     int xMinute = (int)(xCenter + mLength * Math.sin(minute *
72       (2 * Math.PI / 60)));
73     int yMinute = (int)(yCenter - mLength * Math.cos(minute *
74       (2 * Math.PI / 60)));
75     g.setColor(Color.blue);
76     g.drawLine(xCenter, yCenter, xMinute, yMinute);
77
78     // Draw hour hand
79     int hour = (int)cal.get(GregorianCalendar.HOUR_OF_DAY);
80     int hLength = (int)(clockRadius * 0.6);
81     int xHour = (int)(xCenter + hLength * Math.sin((hour + minute
82       / 60.0) * (2 * Math.PI / 12)));
83     int yHour = (int)(yCenter - hLength * Math.cos((hour + minute
84       / 60.0) * (2 * Math.PI / 12)));
85     g.setColor(Color.green);
86     g.drawLine(xCenter, yCenter, xHour, yHour);
87
88     // Set display format in specified style, locale and time zone
89     DateFormat formatter = DateFormat.getDateTimeInstance
90       (DateFormat.MEDIUM, DateFormat.LONG);
91
92     // Display current date
93     g.setColor(Color.red);
94     String today = formatter.format(cal.getTime());
95     FontMetrics fm = g.getFontMetrics();
96     g.drawString(today, (width -
97       fm.stringWidth(today)) / 2, yCenter + clockRadius + 30);
98   }
99 }
```

Since the image is sent to the browser as binary data, the content type of the response is set to image/gif (line 20). The `GifEncoder` class is used to encode the image into content understood by the browser (line 33). The content is sent to the `OutputStream` object `out`.

The program creates an image with the specified width, height, and image type, using the `BufferedImage` class (lines 24–25):

```
Image image = new BufferedImage(width, height,
  BufferedImage.TYPE_INT_ARGB);
```

To draw things on the image, you need to get its graphics context using the `getGraphics` method (line 28):

```
Graphics g = image.getGraphics();
```

You can use various drawing methods in the `Graphics` class to draw simple shapes, or you can use Java 2D to draw more sophisticated graphics. This example uses simple drawing methods to draw a clock that displays the current time.

39.9.3 Sending Images and Text Together

The servlets in the preceding example return images. Often images are mixed with other contents. In this case, you have to set the content type to "image/gif" before sending images, and set the content type to "text/html" before sending the text. However, the content type cannot be changed in one request. To circumvent this restriction, you may embed a GET request for displaying the image in a `<img>` tag in the HTML content. When the HTML content is displayed, a separate GET request for retrieving the image is then sent to the server. Thus text and image are obtained through two separate GET requests.

To demonstrate mixing images and texts, let us create a servlet in Listing 39.15 that mixes the clock image created in the preceding example with some text, as shown in Figure 39.27.

LISTING 39.15 MixedContent.java

```java
package chapter39;

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

public class MixedContent extends HttpServlet {
  /** Process the HTTP Get request */
  public void doGet(HttpServletRequest request, HttpServletResponse
      response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

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

    out.println("<img src = \"" +
      "ImageContentWithDrawing\" align=right>");

    out.println("This is a clock created using a Java program " +
      "and sent to the browser by a servlet.");

    out.close();
  }
}
```

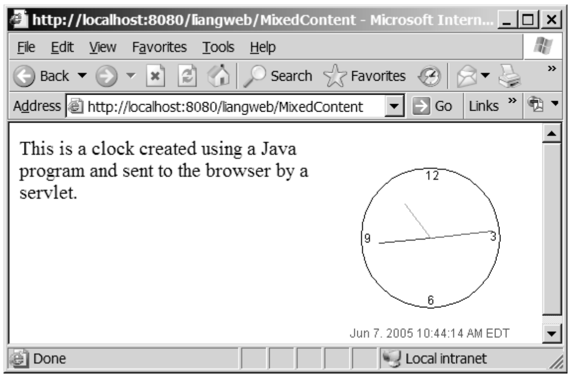
process GET

content type

get parameter

image tag

close stream


http://localhost:8080/liangweb/MixedContent - Microsoft Intern...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/liangweb/MixedContent Go Links
12
9
3
6
Done
Local intranet

This is a clock created using a Java program and sent to the browser by a servlet.

Jun 7. 2005 10:44:14 AM EDT

FIGURE 39.27 The servlet returns an image along with the text.

The servlet generates an HTML file with the image tag

```
<img src = "ImageContentWithDrawing" align = "right">
```

The HTML file is rendered by the browser. When the browser sees the image tag, it sends the request to the server. `ImageContentWithDrawing`, created in Listing 39.14, is invoked to send the image to the browser.

Key Terms

Common Gateway Interface 39–3	life-cycle methods 39–9
CGI programs 39–3	URL query string 39–3
cookie 39–28	servlet 39–2
GET and POST methods 39–4	servlet container (servlet engine) 39–4
GlassFish 39–4	Tomcat 39–4
HTML form 39–14	

Chapter Summary

- A servlet is a special kind of program that runs from a Web server that supports servlets. Tomcat and GlassFish are Web servers that can run servlets.

- A servlet URL is specified by the host name, port, and request string (e.g., http://localhost:8080/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 reference 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 explicitly specify the POST method.

- 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.

- 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.

- 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.

- 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.

- `HttpServletResponse` extends the `ServletResponse` interface to provide HTTP-specific functionality in sending a response. You can use the `getWriter` method to get an instance of `PrintWriter` to send a response to the client and the `setContentType` method to set the type of the content, such as html.

- 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.

- 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.

- 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.

- Java servlets are not limited to sending text to a browser. They can return images in GIF, JPEG, or PNG format.

Review Questions

Sections 39.1–39.2

39.1 What is the common gateway interface?

39.2 What are the differences between the GET and POST methods in an HTML form?

39.3 Can you submit a GET request directly from a URL? Can you submit a POST request directly from a URL?

39.4 What is wrong in the following URL for submitting a GET request to the servlet FindScore on host liang at port 8080 with parameter `name`?

http://liang:8080/findScore?name=“P Yates”

39.5 What are the differences between CGI and servlets?

Section 39.3 Creating and Running Servlets

39.6 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?

39.7 How do you create a Web project in NetBeans?

39.8 How do you create a servlet in NetBeans?

39.9 How do you run a servlet in NetBeans?

39.10 When you run a servlet from NetBeans, what is the port number by default? What happens if the port number is already in use?

39.11 What is the .war file? How do you obtain a .war file for a Web project in NetBeans?

Section 39.4 The Servlet API

39.12 Describe the life cycle of a servlet.

39.13 Suppose that you started a Web server, ran the following servlet twice by issuing an appropriate URL from a Web browser, and finally stopped Web server. 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, HttpServletRe
    sponse
      response) throws ServletException, IOException {
    System.out.println("doGet called");
  }

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

Sections 39.5–39.7

39.14 What would be displayed if you changed the content type to `"html/plain"` in Listing 39.2, CurrentTime.java?

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

39.16 What happens when you invoke `request.getParameter(paramName)` if `paramName` does not exist?

39.17 How do you write a text field, combo box, check box, and text area in an HTML form?

39.18 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?

39.19 If the servlet uses a database driver other than the JDBC-ODBC driver, where should the driver be placed in NetBeans?

Section 39.8 Session Tracking

39.20 What is session tracking? What are three techniques for session tracking?

39.21 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?

39.22 Do you have to create five `Cookie` objects in the servlet in order to send five cookies to the browser?

39.23 How do you get a session, set object value for the session, and get object value from the session?

39.24 Suppose you inserted the following code in line 53 in Listing 39.11.

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

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

39.25 Suppose you inserted the following code in line 53 in Listing 39.11.

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

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

Section 39.9 Sending Images from Servlets

39.26 What output stream should you use to send images to the browser? What content type do you have to set for the response?

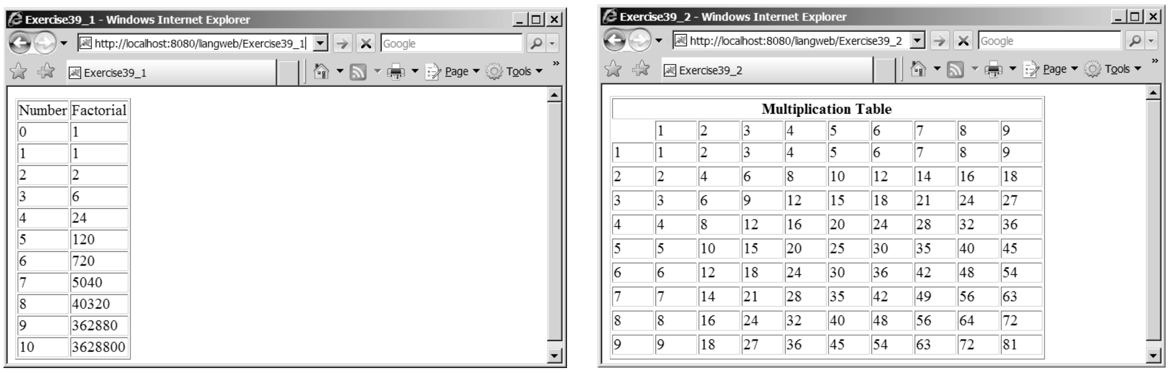
39.27 How do you deal with dynamic contents with images and text?

Programming Exercises

Note
Solutions to even-numbered exercises in this chapter are in `c:\book\liangweb\src\-java\chapter39` in book.zip, which can be downloaded from the Companion Web site.

Section 39.5 Creating Servlets

39.1* (*Factorial table*) Write a servlet to display a table that contains factorials for the numbers from `0` to `10`, as shown in Figure 39.28(a).


Exercise39_1 - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_1
Google
Exercise39_1
Page
Tools
Exercise39_2 - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_2
Google
Exercise39_2
Page
Tools

Number	Factorial
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Multiplication Table

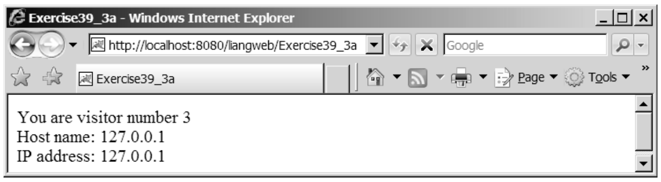
	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

(a) (b)

FIGURE 39.28 (a) The servlet displays factorials for the numbers from `0` to `10` in a table. (b) The servlet displays the multiplication table.

39.2* (*Multiplication table*) Write a servlet to display a multiplication table, as shown in Figure 39.28(b).

39.3* (*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 39.29.


Exercise39_3a - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_3a
Exercise39_3a

```
You are visitor number 3
Host name: 127.0.0.1
IP address: 127.0.0.1
```

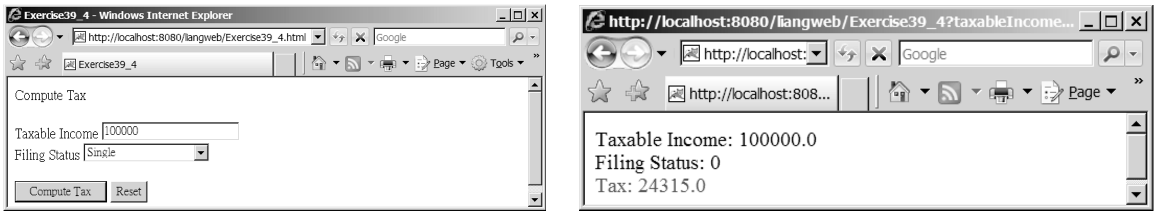
Figure 39.29 The servlet displays the number of visits and the client's host name, IP address, and request URL.

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 `RandomAccessFile` 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 39.6 HTML Forms

39.4* (*Calculating tax*) Write an HTML form to prompt the user to enter taxable income and filing status, as shown in Figure 39.30(a). Clicking the *Compute Tax* button invokes a servlet to compute and display the tax, as shown in Figure 39.30(b). Use the `computeTax` method introduced in Listing 3.7, ComputingTax.java, to compute tax.

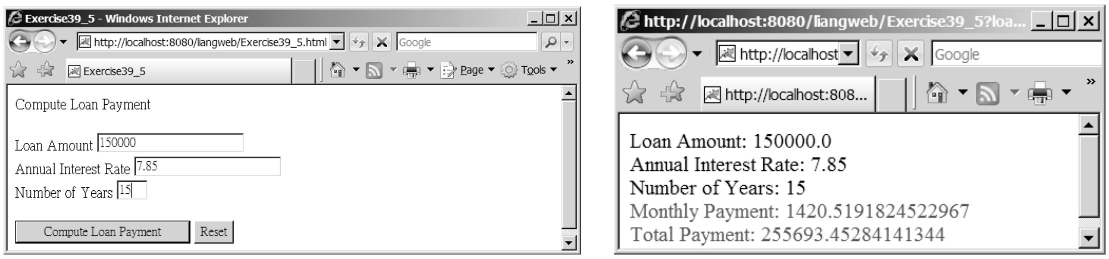

Exercise39_4 - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_4.html
Compute Tax
Taxable Income 100000
Filing Status Single
Compute Tax
Reset
http://localhost:8080/liangweb/Exercise39_4?taxableIncome...

```
Taxable Income: 100000.0
Filing Status: 0
Tax: 24315.0
```

(a) (b)

Figure 39.30 The servlet computes the tax.

39.5* (*Calculating loan*) Write an HTML form that prompts the user to enter loan amount, interest rate, and number of years, as shown in Figure 39.31(a). Clicking the *Compute Loan Payment* button invokes a servlet to compute and display the monthly and total loan payments, as shown in Figure 39.31(b). Use the `Loan` class given in Listing 9.2, Loan.java, to compute the monthly and total payments.


Exercise39_5 - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_5.html
Compute Loan Payment
Reset
http://localhost:8080/liangweb/Exercise39_5?loa...

Compute Loan Payment

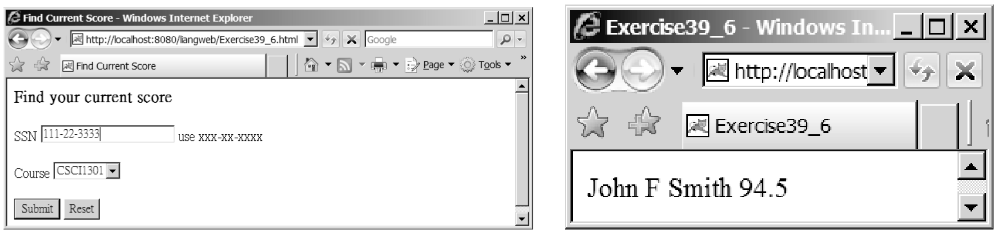
Loan Amount 150000
Annual Interest Rate 7.85
Number of Years 15

Loan Amount: 150000.0
Annual Interest Rate: 7.85
Number of Years: 15
Monthly Payment: 1420.5191824522967
Total Payment: 255693.45284141344

(a) (b)

Figure 39.31 The servlet computes the loan payment.

39.6** (*Finding 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 39.32(a). Upon clicking the *Submit* button, the result is displayed, as shown in Figure 39.32(b). If the SSN or the class ID does not match, report an error. Assume three courses are available: CSCI1301, CSCI1302, and CSCI3720.


Find Current Score - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_6.html
Submit
Reset
Exercise39_6 - Windows In...

Find your current score

SSN 111-22-3333 use xxx-xx-xxxx

Course CSCI1301

John F Smith 94.5

(a) (b)

Figure 39.32 The HTML form accepts the SSN and class ID from the user and sends them to the servlet to obtain the score.

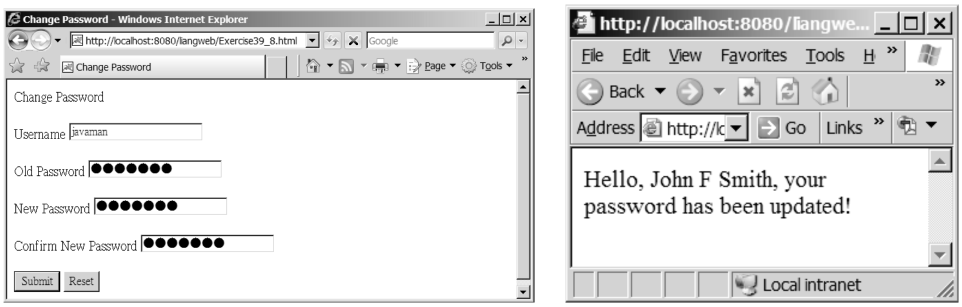
Section 39.7 Database Programming in Servlets

39.7** (*Finding scores from database tables*) Rewrite the preceding servlet. Assume that 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.

39.8* (*Changing the password*) Write a servlet that enables the user to change the password from an HTML form, as shown in Figure 39.33(a). Suppose that 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 39.33(b).


Change Password - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_8.html
Submit
Reset
http://localhost:8080/liangwe...
Local intranet

Change Password

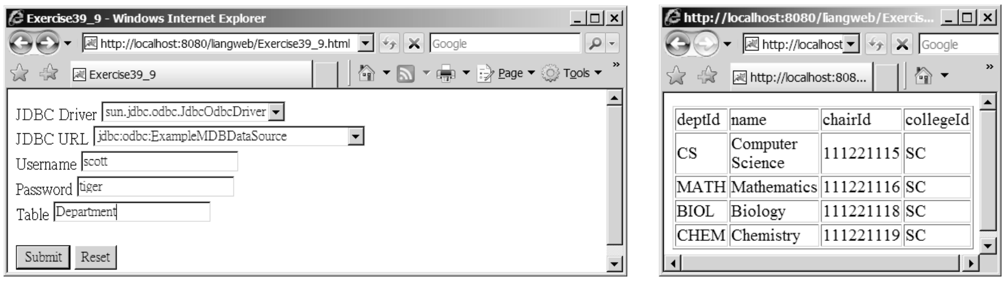
Username	javaman
Old Password	●●●●●●●
New Password	●●●●●●●
Confirm New Password	●●●●●●●

Hello, John F Smith, your password has been updated!

(a) (b)

Figure 39.33 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.

39.9** (*Displaying 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 39.34(a). Clicking the *Submit* button displays the table content, as shown in Figure 39.34(b).


Exercise39_9 - Windows Internet Explorer
http://localhost:8080/liangweb/Exercise39_9.html
Submit
Reset
http://localhost:8080/liangweb/Exercis...

JDBC Driver	sun.jdbc.odbc.JdbcOdbcDriver
JDBC URL	jdbc:odbc:ExampleMDBDataSource
Username	scott
Password	tiger
Table	Department

deptId	name	chairId	collegeId
CS	Computer Science	111221115	SC
MATH	Mathematics	111221116	SC
BIOL	Biology	111221118	SC
CHEM	Chemistry	111221119	SC

(a) (b)

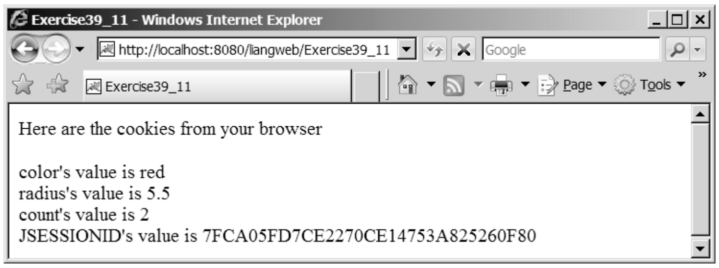
Figure 39.34 The user enters database information and specifies a table to display its content.

Section 39.8 Session Tracking

39.10* (*Storing 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.

39.11* (*Retrieving 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 39.35.

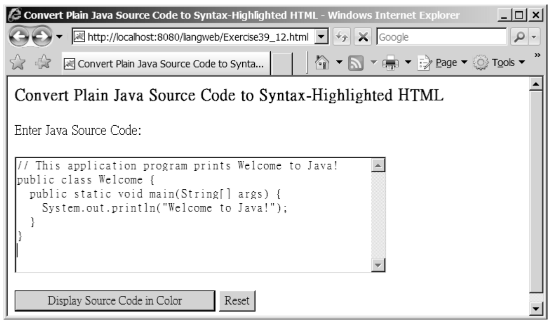


```
Here are the cookies from your browser

color's value is red
radius's value is 5.5
count's value is 2
JSESSIONID's value is 7FCA05FD7CE2270CE14753A825260F80
```

FIGURE 39.35 All the cookies on the client are displayed in the browser.

Comprehensive

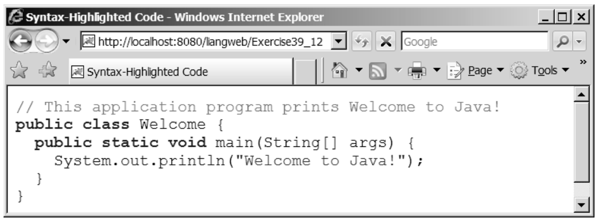
39.12*** (*Syntax highlighting*) Create an HTML form that prompts the user to enter a Java program in a text area, as shown in Figure 39.36(a). The form invokes a servlet that displays the Java source code in a syntax-highlighted HTML format, as shown in Figure 39.36(b). The keywords, comments, and literals are displayed in bold navy, green, and blue, respectively.



Convert Plain Java Source Code to Syntax-Highlighted HTML

Enter Java Source Code:

```
// This application program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

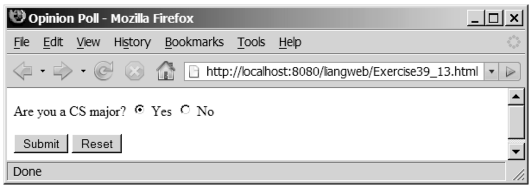


```
// This application program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

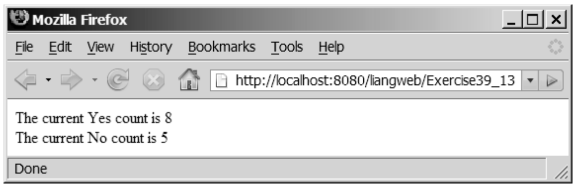
(a) (b)

FIGURE 39.36 The Java code in plain text in (a) is displayed in HTML with syntax highlighted in (b).

39.13*** (*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 39.37(a). 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 39.37(b).



Are you a CS major? [✓ Yes] No



The current Yes count is 8
The current No count is 5

(a) (b)

FIGURE 39.37 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 (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);
```