### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void init(ServletConfig config)</code></td>
<td>This method is automatically called once during a servlet’s execution cycle to initialize the servlet. The <code>ServletConfig</code> argument is supplied automatically by the server that executes the servlet.</td>
</tr>
<tr>
<td><code>ServletConfig getServletConfig()</code></td>
<td>This method returns a reference to an object that implements interface <code>ServletConfig</code>. This object provides access to the servlet’s configuration information such as initialization parameters and the servlet’s <code>ServletContext</code>, which provides the servlet with access to its environment (i.e., the server in which the servlet is executing).</td>
</tr>
<tr>
<td><code>void service(ServletRequest request, ServletResponse response)</code></td>
<td>This is the first method called on every servlet to respond to a client request.</td>
</tr>
<tr>
<td><code>String getServletInfo()</code></td>
<td>This method is defined by a servlet programmer to return a <code>String</code> containing servlet information such as the servlet’s author and version.</td>
</tr>
<tr>
<td><code>void destroy()</code></td>
<td>This “cleanup” method is called when a servlet is terminated by the server on which it is executing. This is a good method to use to deallocate a resource used by the servlet (such as an open file or an open database connection).</td>
</tr>
</tbody>
</table>

**Fig. 19.1** Methods of interface `Servlet`.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doDelete</strong></td>
<td>Called in response to an HTTP <code>DELETE</code> request. Such a request is normally used to delete a file from the server. This may not be available on some servers because of its inherent security risks.</td>
</tr>
<tr>
<td><strong>doOptions</strong></td>
<td>Called in response to an HTTP <code>OPTIONS</code> request. This returns information to the client indicating the HTTP options supported by the server.</td>
</tr>
<tr>
<td><strong>doPut</strong></td>
<td>Called in response to an HTTP <code>PUT</code> request. Such a request is normally used to store a file on the server. This may not be available on some servers because of its inherent security risks.</td>
</tr>
<tr>
<td><strong>doTrace</strong></td>
<td>Called in response to an HTTP <code>TRACE</code> request. Such a request is normally used for debugging. The implementation of this method automatically returns an HTML document to the client containing the request header information (data sent by the browser as part of the request).</td>
</tr>
</tbody>
</table>

Fig. 19.2  Important methods of class `HttpServlet`. 
## Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String getParameter(String name)</code></td>
<td>Returns the value associated with a parameter sent to the servlet as part of a GET or POST request. The <code>name</code> argument represents the parameter name.</td>
</tr>
<tr>
<td><code>Enumeration getParameterNames()</code></td>
<td>Returns the names of all the parameters sent to the servlet as part of a POST request.</td>
</tr>
<tr>
<td><code>String[] getParameterValues(String name)</code></td>
<td>Returns an array of <code>String</code> s containing the values for a specified servlet parameter.</td>
</tr>
<tr>
<td><code>Cookie[] get_cookies()</code></td>
<td>Returns an array of <code>Cookie</code> objects stored on the client by the server. <code>Cookies</code> can be used to uniquely identify clients to the servlet.</td>
</tr>
<tr>
<td><code>HttpSession getSession(boolean create)</code></td>
<td>Returns an <code>HttpSession</code> object associated with the client’s current browsing session. An <code>HttpSession</code> object can be created by this method (true argument) if an <code>HttpSession</code> object does not already exist for the client. <code>HttpSession</code> objects can be used in similar ways to <code>Cookie</code> s for uniquely identifying clients.</td>
</tr>
</tbody>
</table>

---

**Fig. 19.3** Important methods of interface `HttpServletRequest`.
### ServletOutputStream getOutputStream()

Obtains a byte-based output stream that enables binary data to be sent to the client.

### PrintWriter getWriter()

Obtains a character-based output stream that enables text data to be sent to the client.

### void setContentType( String type )

Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type "text/html" indicates that the response is an HTML document, so the browser displays the HTML page.

**Fig. 19.4** Important methods of `HttpServletResponse`.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addCookie( Cookie cookie )</code></td>
<td>Used to add a <code>Cookie</code> to the header of the response to the client. The <code>Cookie</code>’s maximum age and whether the client allows <code>Cookies</code> to be saved determine whether or not <code>Cookies</code> will be stored on the client.</td>
</tr>
<tr>
<td><code>ServletOutputStream getOutputStream()</code></td>
<td>Obtains a byte-based output stream that enables binary data to be sent to the client.</td>
</tr>
<tr>
<td><code>PrintWriter getWriter()</code></td>
<td>Obtains a character-based output stream that enables text data to be sent to the client.</td>
</tr>
<tr>
<td><code>void setContentType( String type )</code></td>
<td>Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type &quot;text/html&quot; indicates that the response is an HTML document, so the browser displays the HTML page.</td>
</tr>
</tbody>
</table>
// Fig. 19.5: HTTPGetServlet.java
// Creating and sending a page to the client
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HTTPGetServlet extends HttpServlet {
    public void doGet( HttpServletRequest request,
                       HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter output;
        response.setContentType( "text/html" );  // content type
        output = response.getWriter();           // get writer

        // create and send HTML page to client
        StringBuffer buf = new StringBuffer();
        buf.append( "<HTML><HEAD><TITLE>");
        buf.append( "A Simple Servlet Example
" );
        buf.append( "</TITLE></HEAD><BODY>");
        buf.append( "<H1>Welcome to Servlets!\n" );
        buf.append( "</H1></BODY></HTML>" );
        output.println( buf.toString() );
        output.close();    // close PrintWriter stream
    }
}

Fig. 19.5  The HTTPGetServlet, which processes an HTTP GET request.
```html
<!-- Fig. 19.6: HTTPGetServlet.html -->

<HTML>
<HEAD>
<TITLE>
Servlet HTTP GET Example
</TITLE>
</HEAD>
<BODY>
<br />
<FORM
ACTION="http://localhost:8080/servlet/HTTPGetServlet"
METHOD="GET">
<br />
<P>Click the button to have the servlet send an HTML document</P>
<br />
<INPUT TYPE="submit" VALUE="Get HTML Document">
</FORM>
</BODY>
</HTML>
```

Fig. 19.6  HTML document to issue a GET request to HTTPGetServlet.
// Fig. 19.7: HTTPPostServlet.java
// A simple survey servlet
import javax.servlet;
import javax.servlet.http.*;
import java.text.*;
import java.io.*;
import java.util.*;

public class HTTPPostServlet extends HttpServlet {
    private String animalNames[] = {
        "dog", "cat", "bird", "snake", "none" ];

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                     throws ServletException, IOException {
        int animals[] = null, total = 0;
        File f = new File( "survey.txt" );
        if ( f.exists() ) {
            try {
                ObjectInputStream input = new ObjectInputStream( 
                        new FileInputStream( f ) );
                animals = (int []) input.readObject();
                input.close();   // close stream
                for ( int i = 0; i < animals.length; ++i )
                    total += animals[ i ];
            } catch( ClassNotFoundException cnfe ) {
                cnfe.printStackTrace();
            }
        } else
            animals = new int[ 5 ];

        // read current survey response
        String value = request.getParameter( "animal" );
        ++total;   // update total of all responses

        // determine which was selected and update its total
        for ( int i = 0; i < animalNames.length; ++i )
            if ( value.equals( animalNames[ i ] ) )
            ++animals[ i ];

        // write updated totals out to disk
        ObjectOutputStream output = new ObjectOutputStream( 
                      new FileOutputStream( f ) );
    }
}
output.writeObject(animals);
output.flush();
output.close();

// Calculate percentages
double percentages[] = new double[animals.length];
for (int i = 0; i < percentages.length; ++i)
    percentages[i] = 100.0 * animals[i] / total;

// send a thank you message to client
response.setContentType("text/html"); // content type
PrintWriter responseOutput = response.getWriter();
StringBuffer buf = new StringBuffer();
buf.append("<html>
");
buf.append("<title>Thank you!</title>
");
buf.append("Thank you for participating.
");
buf.append("<BR>Results:
<PRE>
");
DecimalFormat twoDigits = new DecimalFormat("#0.00");
for (int i = 0; i < percentages.length; ++i) {
    buf.append("<BR>
");
    buf.append(animalNames[i]);
    buf.append("
");
    buf.append(twoDigits.format(percentages[i]));
    buf.append("% responses: ");
    buf.append(animals[i]);
    buf.append("\n");
}
buf.append("\n<BR>Total responses: ");
buf.append(total);
buf.append("</PRE>\n</html> ");
responseOutput.println(buf.toString());
responseOutput.close();
}

Fig. 19.7 The HTTPPostServlet that processes an HTTP POST request (part 2 of 2).
<!-- Fig. 19.8: HTTPPostServlet.html -->

<HTML>
<HEAD>
  <TITLE>Servlet HTTP Post Example</TITLE>
</HEAD>

<BODY>
  <FORM METHOD="POST" ACTION="http://localhost:8080/servlet/HTTPPostServlet">
    What is your favorite pet?<BR><BR>
    <INPUT TYPE=radio NAME=animal VALUE=dog>Dog<BR>
    <INPUT TYPE=radio NAME=animal VALUE=cat>Cat<BR>
    <INPUT TYPE=radio NAME=animal VALUE=bird>Bird<BR>
    <INPUT TYPE=radio NAME=animal VALUE=snake>Snake<BR>
    <INPUT TYPE=radio NAME=animal VALUE=none CHECKED>None<BR><BR>
    <INPUT TYPE=submit VALUE="Submit"> <INPUT TYPE=reset>
  </FORM>
</BODY>
</HTML>

Fig. 19.8 Issuing a POST request to HTTPPostServlet (part 1 of 2).
Fig. 19.8  Issuing a POST request to HTTPPostServlet (part 2 of 2).
// Fig. 19.9: CookieExample.java
// Using cookies.
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class CookieExample extends HttpServlet {
    private String names[] = { "C", "C++", "Java", "Visual Basic 6" };
    private String isbn[] = {
        "0-13-226119-7", "0-13-528910-6",
        "0-13-012507-5", "0-13-528910-6" }
    public void doPost( HttpServletRequest request, 
                       HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter output;
        String language = request.getParameter( "lang" );
        Cookie c = new Cookie( language, getISBN( language ) );
        c.setMaxAge( 120 );  // seconds until cookie removed
        response.addCookie( c );  // must precede getWriter
        response.setContentType( "text/html" );
        output = response.getWriter();
        // send HTML page to client
        output.println( "<HTML><HEAD><TITLE>");
        output.println( "Cookies" );
        output.println( "</TITLE></HEAD><BODY>" );
        output.println( "<P>Welcome to Cookies!<BR>" );
        output.println( language );
        output.println( " is a great language." );
        output.println( "</BODY></HTML>" );
        output.close();    // close stream
    }
    public void doGet( HttpServletRequest request, 
                      HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter output;
        Cookie cookies[];
        cookies = request.getCookies(); // get client's cookies
        response.setContentType( "text/html" );
        output = response.getWriter();
    }
}

Fig. 19.9  Demonstrating Cookies (part 1 of 2).
53    output.println( "<HTML><HEAD><TITLE>" );
54    output.println( "Cookies II" );
55    output.println( "</TITLE></HEAD><BODY>" );
56
57    if ( cookies != null ) {
58        output.println( "<H1>Recommendations</H1>" );
59
60        // get the name of each cookie
61        for ( int i = 0; i < cookies.length; i++ )
62            output.println(
63                cookies[ i ].getName() + " How to Program. " +
64                "ISBN#: " + cookies[ i ].getValue() + "<BR>" );
65    }
66
67    else {
68        output.println( "<H1>No Recommendations</H1>" );
69        output.println( "You did not select a language or" );
70        output.println( "the cookies have expired." );
71    }
72
73    output.println( "</BODY></HTML>" );
74    output.close();    // close stream
75}
76
77    private String getISBN( String lang )
78    {
79        for ( int i = 0; i < names.length; ++i )
80            if ( lang.equals( names[ i ] ) )
81                return isbn[ i ];
82        return "";    // no matching string found
83    }
84}

---

Fig. 19.9  Demonstrating Cookies (part 2 of 2).
Fig. 19.10  HTML document that invokes the cookie servlet with a POST request and passes the user's language selection as an argument (part 1 of 2).

```html
<!-- Fig. 19.10: SelectLanguage.html -->
<html>
<head>
<title>Cookies</title>
</head>
<body>
<form action="http://localhost:8080/servlet/CookieExample"
method="POST">
  <strong>Select a programming language:<br>
  </strong><br>
  <pre>
  <input type="radio" name="lang" value="C">C<br>
  <input type="radio" name="lang" value="C++">C++<br>
  <input type="radio" name="lang" value="Java" checked>Java<br>
  <input type="radio" name="lang" value="Visual Basic 6">Visual Basic 6<br>
  </pre>
  <input type="submit" value="Submit">
  <input type="reset">
</form>
</body>
</html>
```
Fig. 19.10  HTML document that invokes the cookie servlet with a **POST** request and passes the user’s language selection as an argument (part 2 of 2).
Fig. 19.11 HTML document for a servlet that reads a client’s cookies (part 1 of 2).
Fig. 19.11  HTML document for a servlet that reads a client’s cookies (part 2 of 2).
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getComment()</code></td>
<td>Returns a <code>String</code> describing the purpose of the cookie (null if no comment has been set with <code>setComment</code>).</td>
</tr>
<tr>
<td><code>getDomain()</code></td>
<td>Returns a <code>String</code> containing the cookie’s domain. This determines which servers can receive the cookie. By default cookies are sent to the server that originally sent the cookie to the client.</td>
</tr>
<tr>
<td><code>getMaxAge()</code></td>
<td>Returns an <code>int</code> representing the maximum age of the cookie in seconds.</td>
</tr>
<tr>
<td><code>getName()</code></td>
<td>Returns a <code>String</code> containing the name of the cookie as set by the constructor.</td>
</tr>
<tr>
<td><code>getPath()</code></td>
<td>Returns a <code>String</code> containing the URL prefix for the cookie. Cookies can be “targeted” to specific URLs that include directories on the Web server. By default a cookie is returned to services operating in the same directory as the service that sent the cookie or a subdirectory of that directory.</td>
</tr>
<tr>
<td><code>getSecure()</code></td>
<td>Returns a <code>boolean</code> value indicating if the cookie should be transmitted using a secure protocol (true).</td>
</tr>
<tr>
<td><code>getValue()</code></td>
<td>Returns a <code>String</code> containing the value of the cookie as set with <code>setValue</code> or the constructor.</td>
</tr>
<tr>
<td><code>getVersion()</code></td>
<td>Returns an <code>int</code> containing the version of the cookie protocol used to create the cookie. Cookies are currently undergoing standardization. A value of 0 (the default) indicates the original cookie protocol as defined by Netscape. A value of 1 indicates the version currently undergoing standardization.</td>
</tr>
<tr>
<td><code>setComment( String )</code></td>
<td>The comment describing the purpose of the cookie that is presented by the browser to the user (some browsers allow the user to accept cookies on a per-cookie basis).</td>
</tr>
<tr>
<td><code>setDomain( String )</code></td>
<td>This determines which servers can receive the cookie. By default cookies are sent to the server that originally sent the cookie to the client. The domain is specified in the form &quot;.deitel.com&quot;, indicating that all servers ending with .deitel.com can receive this cookie.</td>
</tr>
<tr>
<td><code>setMaxAge( int )</code></td>
<td>Sets the maximum age of the cookie in seconds.</td>
</tr>
<tr>
<td><code>setPath( String )</code></td>
<td>Sets the “target” URL indicating the directories on the server that lead to the services that can receive this cookie.</td>
</tr>
<tr>
<td><code>setSecure( boolean )</code></td>
<td>A <code>true</code> value indicates that the cookie should only be sent using a secure protocol.</td>
</tr>
<tr>
<td><code>setValue( String )</code></td>
<td>Sets the value of a cookie.</td>
</tr>
<tr>
<td><code>setVersion( int )</code></td>
<td>Sets the cookie protocol for this cookie.</td>
</tr>
</tbody>
</table>

**Fig. 19.12 Important methods of class `Cookie`**
public class SessionExample extends HttpServlet {
    private final static String names[] = {
        "C", "C++", "Java", "Visual Basic 6"};
    private final static String isbn[] = {
        "0-13-226119-7", "0-13-528910-6",
        "0-13-012507-5", "0-13-528910-6"};

    public void doPost( HttpServletRequest request, HttpServletResponse response )
            throws ServletException, IOException {
        PrintWriter output;
        String language = request.getParameter( "lang" );

        // Get the user's session object.
        // Create a session (true) if one does not exist.
        HttpSession session = request.getSession( true );

        // add a value for user's choice to session
        session.putValue( language, getISBN( language ) );

        response.setContentType( "text/html" );
        output = response.getWriter();

        // send HTML page to client
        output.println( "<HTML><HEAD><TITLE>");
        output.println( "Sessions" );
        output.println( "</TITLE></HEAD><BODY>" );
        output.println( "<P>Welcome to Sessions!<BR>");
        output.println( language );
        output.println( " is a great language." );
        output.println( "</BODY></HTML>" );

        output.close();    // close stream
    }

    public void doGet( HttpServletRequest request, HttpServletResponse response )
            throws ServletException, IOException {
        PrintWriter output;

        // Get the user's session object.
        // Don't create a session (false) if one does not exist.
        HttpSession session = request.getSession( false );
// get names of session object's values
String valueNames[];

if ( session != null )
    valueNames = session.getValueNames();
else
    valueNames = null;

response.setContentType( "text/html" );
output = response.getWriter();

output.println( "<HTML><HEAD><TITLE>" );
output.println( "Sessions II" );
output.println( "</TITLE></HEAD><BODY>" );

if ( valueNames != null && valueNames.length != 0 ) {
    output.println( "<H1>Recommendations</H1>" );

    // get value for each name in valueNames
    for ( int i = 0; i < valueNames.length; i++ ) {
        String value =
                    (String) session.getValue( valueNames[ i ] );

        output.println( valueNames[ i ] + " How to Program. " +
                        "ISBN#: " + value + "<BR>" );
    }
}
else {
    output.println( "<H1>No Recommendations</H1>" );
    output.println( "You did not select a language or" );
    output.println( "the session has expired." );
}

output.println( "</BODY></HTML>" );
output.close(); // close stream

private String getISBN( String lang )
{
    for ( int i = 0; i < names.length; ++i )
        if ( lang.equals( names[ i ] ) )
            return isbn[ i ];

    return ""; // no matching string found
}
<!-- Fig. 19.14: SelectLanguage.html -->

```html
<HTML>
<HEAD>
<TITLE>Sessions</TITLE>
</HEAD>
<BODY>
<FORM ACTION="http://localhost:8080/servlet/SessionExample"
METHOD="POST">
<STRONG>Select a programming language:<br>
</STRONG><BR>
<pre>
<INPUT TYPE="radio" NAME="lang" VALUE="C">C<br>
<INPUT TYPE="radio" NAME="lang" VALUE="C++">C++<br>
<INPUT TYPE="radio" NAME="lang" VALUE="Java" CHECKED>Java<br>
<INPUT TYPE="radio" NAME="lang" VALUE="Visual Basic 6">Visual Basic 6<br>
</pre>
<INPUT TYPE="submit" VALUE="Submit">
<INPUT TYPE="reset"> </P>
</FORM>
</BODY>
</HTML>
```

Fig. 19.14  HTML document that invokes the session tracking servlet with a **POST** request and passes the language selection as an argument (part 1 of 2).
**Fig. 19.14** HTML document that invokes the session tracking servlet with a **POST** request and passes the language selection as an argument (part 2 of 2).
<!-- Fig. 19.15: BookRecommendation.html -->

<HTML>
<HEAD>
<TITLE>Sessions</TITLE>
</HEAD>
<BODY>
<FORM ACTION="http://localhost:8080/servlet/SessionExample"
METHOD="GET">
Press "Recommend books" for a list of books.

<INPUT TYPE=submit VALUE="Recommend books">
</FORM>
</BODY>
</HTML>

Fig. 19.15  HTML that interacts with the session tracking servlet to read the session information and return book recommendations to the user (part 1 of 2).
Fig. 19.15 HTML that interacts with the session tracking servlet to read the session information and return book recommendations to the user (part 2 of 2).
public class GuestBookServlet extends HttpServlet {
    private Statement statement = null;
    private Connection connection = null;
    private String URL = "jdbc:odbc:GuestBook";

    public void init( ServletConfig config )
    throws ServletException
    {
        super.init( config );
        try {
            Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
            connection = DriverManager.getConnection( URL, "", "" );
        }
        catch ( Exception e ) {
            e.printStackTrace();
            connection = null;
        }
    }

    public void doPost( HttpServletRequest req, HttpServletResponse res )
    throws ServletException, IOException
    {
        String email, firstName, lastName, company, snailmailList, cppList, javaList, vbList, iwwwList;
        email = req.getParameter( "Email" );
        firstName = req.getParameter( "FirstName" );
        lastName = req.getParameter( "LastName" );
        company = req.getParameter( "Company" );
        snailmailList = req.getParameter( "mail" );
        cppList = req.getParameter( "c_cpp" );
        javaList = req.getParameter( "java" );
        vbList = req.getParameter( "vb" );
        iwwwList = req.getParameter( "iwww" );
        PrintWriter output = res.getWriter();
        res.setContentType( "text/html" );
if ( email.equals( "" ) ||
    firstName.equals( "" ) ||
    lastName.equals( "" ) ) {
    output.println( "<H3> Please click the back " +
                     "button and fill in all " +
                     "fields.</H3>" );
    output.close();
    return;
}

/* Note: The GuestBook database actually contains fields
   * Address1, Address2, City, State and Zip that are not
   * used in this example. However, the insert into the
   * database must still account for these fields. */
boolean success = insertIntoDB(
    "'" + email + ',' + firstName + ',' + lastName +
    ',' + company + '',' ',' ',' ',' ',' ',' ';
( snailmailList != null ? "yes" : "no" ) + ',' +
( cppList != null ? "yes" : "no" ) + ',' +
( javaList != null ? "yes" : "no" ) + ',' +
( iwwwList != null ? "yes" : "no" ) + ' ');

if ( success )
    output.print( "<H2>Thank you " + firstName +
                 " for registering.</H2>" );
else
    output.print( "<H2>An error occurred. " +
                  "Please try again later.</H2>" );

output.close();
private boolean insertIntoDB( String stringtoinsert )
{
    try {
         statement = connection.createStatement();
         statement.execute(
            "INSERT INTO GuestBook values (" +
            stringtoinsert + ");" );
         statement.close();
    }
}

Fig. 19.16  GuestBookServlet, which allows client to register for mailing lists
           (part 2 of 3).
101 }
102
103 public void destroy()
104 {
105     try {
106         connection.close();
107     }
108     catch( Exception e ) {
109         System.err.println( "Problem closing the database" );
110     }
111 }
112 }

**Fig. 19.16** GuestBookServlet, which allows client to register for mailing lists (part 3 of 3).
<!-- Fig. 19.17: GuestBookForm.html -->

<HTML>
<HEAD>
<TITLE>Deitel Guest Book Form</TITLE>
</HEAD>

<BODY>
<H1>Guest Book</H1>

<FORM ACTION=http://localhost:8080/servlet/GuestBookServlet
METHOD=POST>
<pre>
* Email address: <INPUT TYPE=text NAME=Email>
* First Name:    <INPUT TYPE=text NAME=FirstName>
* Last name:     <INPUT TYPE=text NAME=LastName>
Company:         <INPUT TYPE=text NAME=Company>

* fields are required
</PRE>

<P> Select mailing lists from which you want to receive information
<INPUT TYPE=CHECKBOX NAME=mail VALUE=mail>
Snail Mail<br>
<INPUT TYPE=CHECKBOX NAME=c_cpp VALUE=c_cpp>
<i>C++ How to Program & C How to Program</i><br>
<INPUT TYPE=CHECKBOX NAME=java VALUE=java>
<i>Java How to Program</i><br>
<INPUT TYPE=CHECKBOX NAME=vb VALUE=vb>
<i>Visual Basic How to Program</i><br>
<INPUT TYPE=CHECKBOX NAME=iwww VALUE=iwww>
<i>Internet and World Wide Web How to Program</i><br>
</P>

<INPUT TYPE=SUBMIT Value="Submit">
</FORM>
</BODY>
</HTML>
Fig. 19.17  HTML that invokes the GuestBookServlet (part 2 of 2).