

| Method | Description |
|---|--|
| void init(ServletConfig config) | This method is automatically called once during a servlet's execution cycle to initialize the servlet. The ServletConfig argument is supplied automatically by the server that executes the servlet. |
| ServletConfig getServletConfig() | This method returns a reference to an object that implements interface ServletConfig . This object provides access to the servlet's configuration information, such as initialization parameters, and the servlet's ServletContext , which provides the servlet with access to its environment (i.e., the server in which the servlet is executing). |
| void service(ServletRequest request, ServletResponse response) | This is the first method called on every servlet to respond to a client request. |
| String getServletInfo() | This method is defined by a servlet programmer to return a String containing servlet information such as the servlet's author and version. |
| void destroy() | 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). |

Fig. 29.1 Methods of interface **Servlet**.

| Method | Description |
|------------------|--|
| doDelete | Called in response to an HTTP DELETE 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. |
| doOptions | Called in response to an HTTP OPTIONS request. This returns information to the client indicating the HTTP options supported by the server. |
| doPut | Called in response to an HTTP PUT 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. |
| doTrace | Called in response to an HTTP TRACE 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). |

Fig. 29.2 Other methods of class `HttpServlet`.

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

Fig. 29.3 Important methods of interface **HttpServletRequest**.

| Method | Description |
|--|--|
| void addCookie(Cookie cookie) | Used to add a Cookie to the header of the response to the client. The Cookie 's maximum age and whether the client allows Cookies to be saved determine whether or not Cookies will be stored on the client. |
| 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. 29.4 Important methods of **HttpServletResponse**.

```
1 // Fig. 29.5: HTTPGetServlet.java
2 // Creating and sending a page to the client
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6
7 public class HTTPGetServlet extends HttpServlet {
8     public void doGet( HttpServletRequest request,
9                       HttpServletResponse response )
10        throws ServletException, IOException
11    {
12        PrintWriter output;
13
14        response.setContentType( "text/html" ); // content type
15        output = response.getWriter();         // get writer
16
17        // create and send HTML page to client
18        StringBuffer buf = new StringBuffer();
19        buf.append( "<HTML><HEAD><TITLE>\n" );
20        buf.append( "A Simple Servlet Example\n" );
21        buf.append( "</TITLE></HEAD><BODY>\n" );
22
23        buf.append( "<H1>Welcome to Servlets!</H1>\n" );
24        buf.append( "</BODY></HTML>" );
25        output.println( buf.toString() );
26        output.close(); // close PrintWriter stream
27    }
28 }
```

Fig. 29.5 The `HTTPGetServlet`, which processes an HTTP `GET` request .

```

1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2  <HTML>
3  <!-- Fig. 29.6: HTTPGetServlet.html -->
4
5  <HEAD>
6  <TITLE>Servlet HTTP GET Example</TITLE>
7  </HEAD>
8
9  <BODY>
10 <FORM ACTION = "http://localhost:8080/servlet/HTTPGetServlet"
11     METHOD = "GET">
12     <P>Click the button to have the servlet send
13       an HTML document</P>
14     <INPUT TYPE = "submit" VALUE = "Get HTML Document">
15 </FORM>
16 </BODY>
17 </HTML>

```

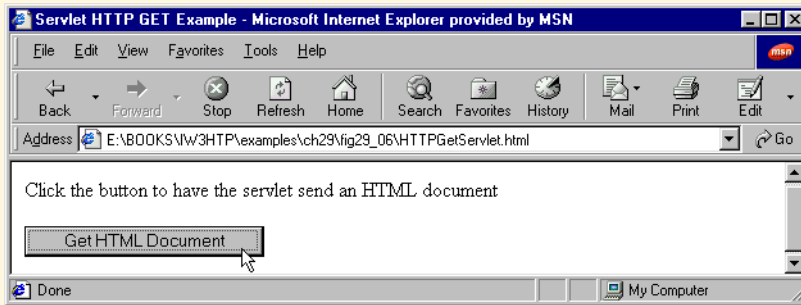


Fig. 29.6 HTML document to issue a **GET** request to **HTTPGetServlet** .

```

1 // Fig. 29.7: HTTPPostServlet.java
2 // A simple survey servlet
3 // This servlet writes data to a file on the Web server.
4 import javax.servlet.*;
5 import javax.servlet.http.*;
6 import java.text.*;
7 import java.io.*;
8
9 public class HTTPPostServlet extends HttpServlet {
10     private String animalNames[] =
11         { "dog", "cat", "bird", "snake", "none" };
12
13     public void doPost( HttpServletRequest request,
14                       HttpServletResponse response )
15         throws ServletException, IOException
16     {
17         int animals[] = null, total = 0;
18         File f = new File( "survey.dat" );
19
20         if ( f.exists() ) {
21             // Determine # of survey responses so far
22             try {
23                 ObjectInputStream input = new ObjectInputStream(
24                     new FileInputStream( f ) );
25
26                 animals = (int []) input.readObject();
27                 input.close(); // close stream
28
29                 for ( int i = 0; i < animals.length; ++i )
30                     total += animals[ i ];
31             }
32             catch( ClassNotFoundException cnfe ) {
33                 cnfe.printStackTrace();
34             }
35         }
36         else
37             animals = new int[ 5 ];
38
39         // read current survey response
40         String value =
41             request.getParameter( "animal" );
42         ++total; // update total of all responses
43
44         // determine which was selected and update its total
45         for ( int i = 0; i < animalNames.length; ++i )
46             if ( value.equals( animalNames[ i ] ) )
47                 ++animals[ i ];
48
49         // write updated totals out to disk

```

Fig. 29.7 HTTPPostServlet that processes an HTTP POST request (part 1 of 2).

```

50     ObjectOutputStream output = new ObjectOutputStream(
51         new FileOutputStream( f ) );
52
53     output.writeObject( animals );
54     output.flush();
55     output.close();
56
57     // Calculate percentages
58     double percentages[] = new double[ animals.length ];
59
60     for ( int i = 0; i < percentages.length; ++i )
61         percentages[ i ] = 100.0 * animals[ i ] / total;
62
63     // send a thank you message to client
64     response.setContentType( "text/html" ); // content type
65
66     PrintWriter responseOutput = response.getWriter();
67     StringBuffer buf = new StringBuffer();
68     buf.append( "<html>\n" );
69     buf.append( "<title>Thank you!</title>\n" );
70     buf.append( "Thank you for participating.\n" );
71     buf.append( "<BR>Results:\n<PRE>" );
72
73     DecimalFormat twoDigits = new DecimalFormat( "#0.00" );
74     for ( int i = 0; i < percentages.length; ++i ) {
75         buf.append( "<BR>" );
76         buf.append( animalNames[ i ] );
77         buf.append( ": " );
78         buf.append( twoDigits.format( percentages[ i ] ) );
79         buf.append( "% responses: " );
80         buf.append( animals[ i ] );
81         buf.append( "\n" );
82     }
83
84     buf.append( "\n<BR><BR>Total responses: " );
85     buf.append( total );
86     buf.append( "</PRE>\n</html>" );
87
88     responseOutput.println( buf.toString() );
89     responseOutput.close();
90 }
91 }

```

Fig. 29.7 HTTPPostServlet that processes an HTTP POST request (part 2 of 2).


```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2 <HTML>
3 <!-- Fig. 29.8: HTTPPostServlet.html -->
4
5 <HEAD>
6 <TITLE>Servlet HTTP Post Example</TITLE>
7 </HEAD>
8
9 <BODY>
10 <FORM ACTION = "http://localhost:8080/servlet/HTTPPostServlet"
11     METHOD = "POST">
12     What is your favorite pet?<BR><BR>
13     <INPUT TYPE = "radio" NAME = "animal" VALUE = "dog">Dog<BR>
14     <INPUT TYPE = "radio" NAME = "animal" VALUE = "cat">Cat<BR>
15     <INPUT TYPE = "radio" NAME = "animal" VALUE = "bird">Bird<BR>
16     <INPUT TYPE = "radio" NAME = "animal" VALUE = "snake">Snake<BR>
17     <INPUT TYPE = "radio" NAME = "animal" VALUE = "none"
18     CHECKED>None
19     <BR><BR><INPUT TYPE = "submit" VALUE = "Submit">
20     <INPUT TYPE = "reset">
21 </FORM>
22 </BODY>
23 </HTML>

```

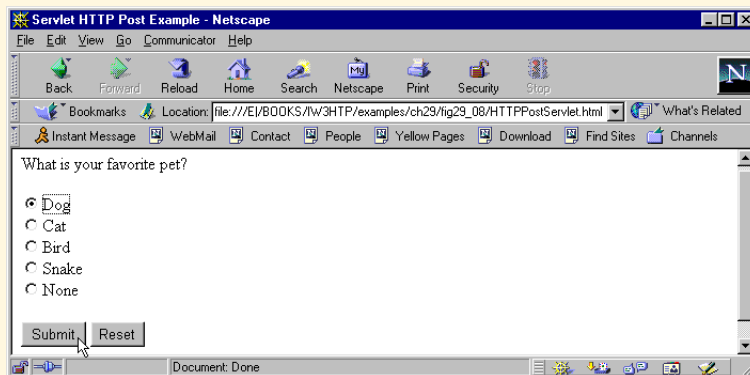


Fig. 29.8 Issuing a **POST** request to **HTTPPostServlet** (part 1 of 2).

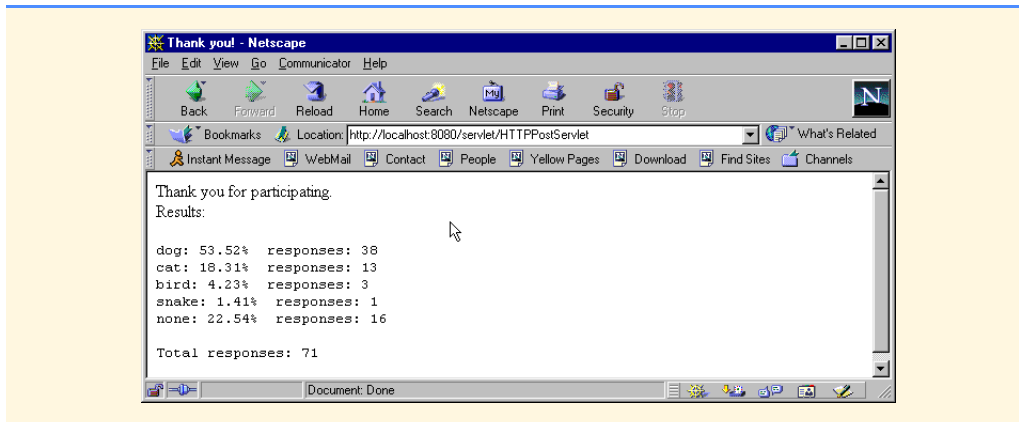


Fig. 29.8 Issuing a **POST** request to **HTTPPostServlet** (part 2 of 2).

```

1 // Fig. 29.9: CookieExample.java
2 // Using cookies.
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6
7 public class CookieExample extends HttpServlet {
8     private final static String names[] = { "C", "C++", "Java",
9                                             "Visual Basic 6" };
10    private final static String isbn[] = {
11        "0-13-226119-7", "0-13-528910-6",
12        "0-13-012507-5", "0-13-456955-5" };
13
14    public void doPost( HttpServletRequest request,
15                      HttpServletResponse response )
16        throws ServletException, IOException
17    {
18        PrintWriter output;
19        String language = request.getParameter( "lang" );
20
21        Cookie c = new Cookie( language, getISBN( language ) );
22        c.setMaxAge( 120 ); // seconds until cookie removed
23        response.addCookie( c ); // must precede getWriter
24
25        response.setContentType( "text/html" );
26        output = response.getWriter();
27
28        // send HTML page to client
29        output.println( "<HTML><HEAD><TITLE>" );
30        output.println( "Cookies" );
31        output.println( "</TITLE></HEAD><BODY>" );
32        output.println( "<P>Welcome to Cookies!<BR>" );
33        output.println( "<P>" );
34        output.println( language );
35        output.println( " is a great language." );
36        output.println( "</BODY></HTML>" );
37
38        output.close(); // close stream
39    }
40
41    public void doGet( HttpServletRequest request,
42                     HttpServletResponse response )
43        throws ServletException, IOException
44    {
45        PrintWriter output;
46        Cookie cookies[];
47
48        cookies = request.getCookies(); // get client's cookies
49
50        response.setContentType( "text/html" );
51        output = response.getWriter();

```

Fig. 29.9 Demonstrating Cookies (part 1 of 2).

```
52
53     output.println( "<HTML><HEAD><TITLE>" );
54     output.println( "Cookies II" );
55     output.println( "</TITLE></HEAD><BODY>" );
56
57     if ( cookies != null && cookies.length != 0 ) {
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     else {
67         output.println( "<H1>No Recommendations</H1>" );
68         output.println( "You did not select a language or" );
69         output.println( "the cookies have expired." );
70     }
71
72     output.println( "</BODY></HTML>" );
73     output.close();    // close stream
74 }
75
76 private String getISBN( String lang )
77 {
78     for ( int i = 0; i < names.length; ++i )
79         if ( lang.equals( names[ i ] ) )
80             return isbn[ i ];
81
82     return "";    // no matching string found
83 }
84 }
```

Fig. 29.9 Demonstrating Cookies (part 2 of 2).

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2 <HTML>
3 <!-- Fig. 29.10: SelectLanguage.html -->
4
5 <HEAD>
6 <TITLE>Cookies</TITLE>
7 </HEAD>
8
9 <BODY>
10 <FORM ACTION = "http://localhost:8080/servlet/CookieExample"
11     METHOD = "POST">
12     <STRONG>Select a programming language:</STRONG><BR>
13     <INPUT TYPE = "radio" NAME = "lang" VALUE = "C">C<BR>
14     <INPUT TYPE = "radio" NAME = "lang" VALUE = "C++">C++<BR>
15     <INPUT TYPE = "radio" NAME = "lang" VALUE = "Java" CHECKED>
16     Java<BR>
17     <INPUT TYPE = "radio" NAME = "lang" VALUE = "Visual Basic 6">
18     Visual Basic 6
19     <P><INPUT TYPE = "submit" VALUE = "Submit">
20     <INPUT TYPE = "reset" > </P>
21 </FORM>
22 </BODY>
23 </HTML>

```

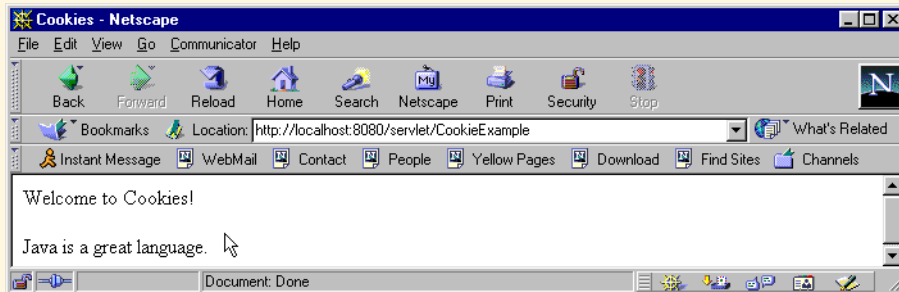
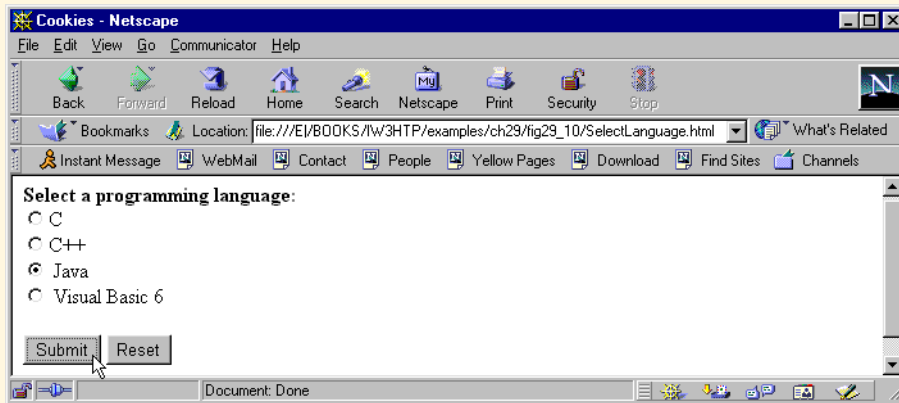


Fig. 29.10 HTML document that invokes the cookie servlet with a **POST** request and passes the user's language selection as an argument .

© Copyright 2000 by Prentice Hall. All Rights Reserved.

For use only by instructors in classes for which *Java How to Program, Third Edition* is the required textbook.

```

1 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2 <HTML>
3 <!-- Fig. 29.11: BookRecommendation.html -->
4
5 <HEAD>
6 <TITLE>Cookies</TITLE>
7 </HEAD>
8
9 <BODY>
10 <FORM ACTION = "http://localhost:8080/servlet/CookieExample"
11     METHOD = "GET">
12     Press "Recommend books" for a list of books.
13     <INPUT TYPE = submit VALUE = "Recommend books">
14 </FORM>
15 </BODY>
16 </HTML>

```

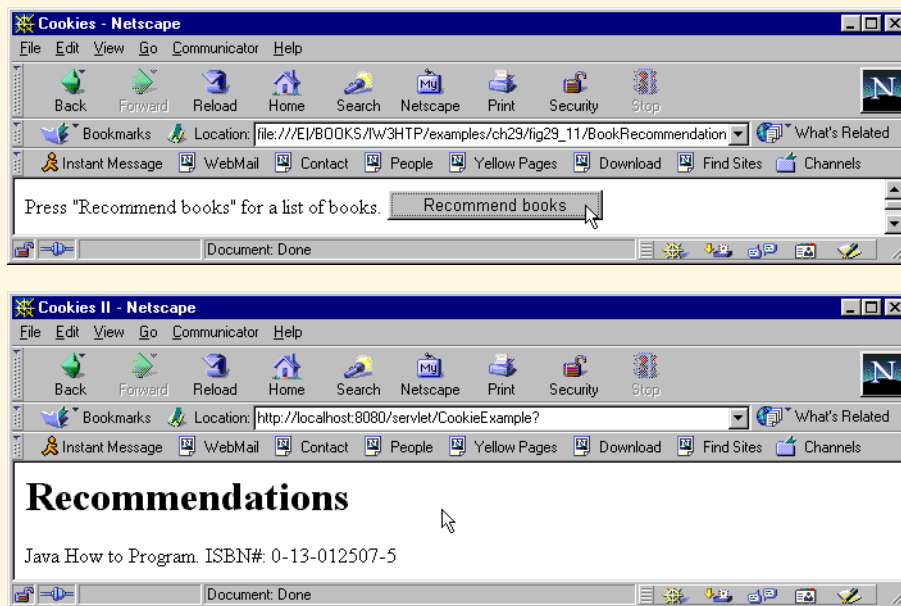


Fig. 29.11 HTML document for a servlet that reads a client's cookies (part 1 of 2).

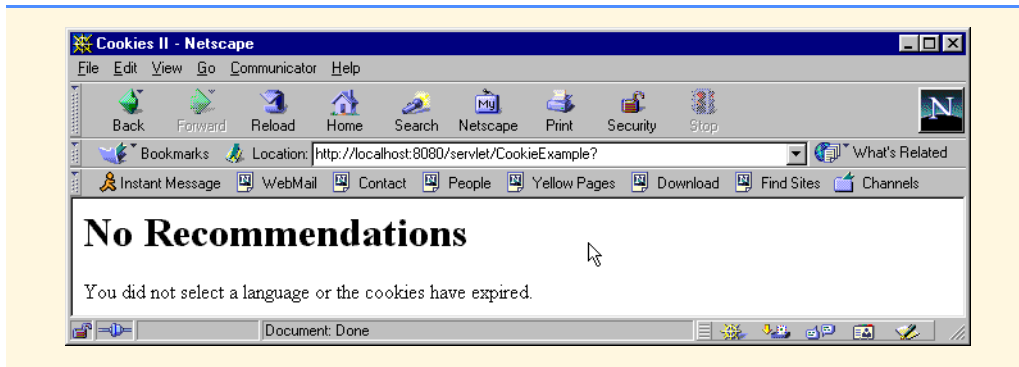


Fig. 29.11 HTML document for a servlet that reads a client's cookies (part 2 of 2).

| Method | Description |
|-----------------------------------|---|
| <code>getComment()</code> | Returns a String describing the purpose of the cookie (null if no comment has been set with <code>setComment</code>). |
| <code>getDomain()</code> | Returns a String 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. |
| <code>getMaxAge()</code> | Returns an int representing the maximum age of the cookie in seconds. |
| <code>getName()</code> | Returns a String containing the name of the cookie as set by the constructor. |
| <code>getPath()</code> | Returns a String 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. |
| <code>getSecure()</code> | Returns a boolean value indicating if the cookie should be transmitted using a secure protocol (true if so). |
| <code>getValue()</code> | Returns a String containing the value of the cookie as set with <code>setValue</code> or the constructor. |
| <code>getVersion()</code> | Returns an int 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. |
| <code>setComment(String)</code> | 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). |
| <code>setDomain(String)</code> | 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 <code>".deitel.com"</code> , indicating that all servers ending with <code>.deitel.com</code> can receive this cookie. |
| <code>setMaxAge(int)</code> | Sets the maximum age of the cookie in seconds. |
| <code>setPath(String)</code> | Sets the "target" URL indicating the directories on the server that lead to the services that can receive this cookie. |
| <code>setSecure(boolean)</code> | A true value indicates that the cookie should only be sent using a secure protocol. |
| <code>setValue(String)</code> | Sets the value of a cookie. |
| <code>setVersion(int)</code> | Sets the cookie protocol for this cookie. |

Fig. 29.12 Important methods of class `Cookie`.


```
1 // Fig. 29.13: SessionExample.java
2 // Using sessions.
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import java.io.*;
6
7 public class SessionExample extends HttpServlet {
8     private final static String names[] =
9         { "C", "C++", "Java", "Visual Basic 6" };
10    private final static String isbn[] = {
11        "0-13-226119-7", "0-13-528910-6",
12        "0-13-012507-5", "0-13-456955-5" };
13
14    public void doPost( HttpServletRequest request,
15                      HttpServletResponse response )
16        throws ServletException, IOException
17    {
18        PrintWriter output;
19        String language = request.getParameter( "lang" );
20
21        // Get the user's session object.
22        // Create a session (true) if one does not exist.
23        HttpSession session = request.getSession( true );
24
25        // add a value for user's choice to session
26        session.putValue( language, getISBN( language ) );
27
28        response.setContentType( "text/html" );
29        output = response.getWriter();
30
31        // send HTML page to client
32        output.println( "<HTML><HEAD><TITLE>" );
33        output.println( "Sessions" );
34        output.println( "</TITLE></HEAD><BODY>" );
35        output.println( "<P>Welcome to Sessions!<BR>" );
36        output.println( "<P>" );
37        output.println( language );
38        output.println( " is a great language." );
39        output.println( "</BODY></HTML>" );
40
41        output.close(); // close stream
42    }
43
44    public void doGet( HttpServletRequest request,
45                     HttpServletResponse response )
46        throws ServletException, IOException
47    {
48        PrintWriter output;
49
50        // Get the user's session object.
51        // Don't create a session (false) if one does not exist.
52        HttpSession session = request.getSession( false );
53
```

Fig. 29.13 Session tracking example (part 1 of 2).

© Copyright 2000 by Prentice Hall. All Rights Reserved.

For use only by instructors in classes for which *Java How to Program, Third Edition* is the required textbook.

```

54     // get names of session object's values
55     String valueNames[];
56
57     if ( session != null )
58         valueNames = session.getValueNames();
59     else
60         valueNames = null;
61
62     response.setContentType( "text/html" );
63     output = response.getWriter();
64
65     output.println( "<HTML><HEAD><TITLE>" );
66     output.println( "Sessions II" );
67     output.println( "</TITLE></HEAD><BODY>" );
68
69     if ( valueNames != null && valueNames.length != 0 ) {
70         output.println( "<H1>Recommendations</H1>" );
71
72         // get value for each name in valueNames
73         for ( int i = 0; i < valueNames.length; i++ ) {
74             String value =
75                 (String) session.getValue( valueNames[ i ] );
76
77             output.println(
78                 valueNames[ i ] + " How to Program. " +
79                 "ISBN#: " + value + "<BR>" );
80         }
81     }
82     else {
83         output.println( "<H1>No Recommendations</H1>" );
84         output.println( "You did not select a language" );
85     }
86
87     output.println( "</BODY></HTML>" );
88     output.close();    // close stream
89 }
90
91 private String getISBN( String lang )
92 {
93     for ( int i = 0; i < names.length; ++i )
94         if ( lang.equals( names[ i ] ) )
95             return isbn[ i ];
96
97     return "";    // no matching string found
98 }
99 }

```

Fig. 29.13 Session tracking example (part 2 of 2).

```

1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2  <HTML>
3  <!-- Fig. 29.15: BookRecommendation.html -->
4
5  <HEAD>
6  <TITLE>Sessions</TITLE>
7  </HEAD>
8
9  <BODY>
10 <FORM ACTION = "http://localhost:8080/servlet/SessionExample"
11     METHOD = "GET">
12     Press "Recommend books" for a list of books.
13     <INPUT TYPE = submit VALUE = "Recommend books">
14 </FORM>
15 </BODY>
16 </HTML>

```

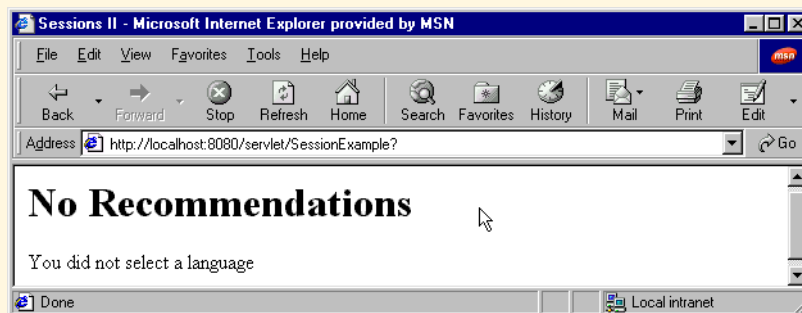
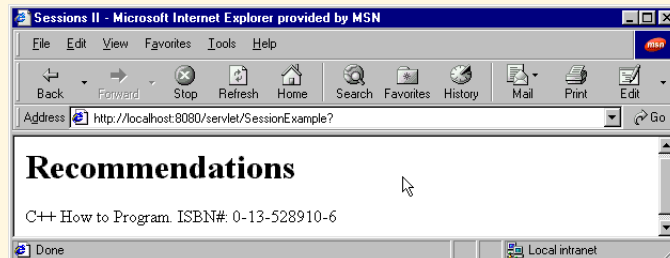
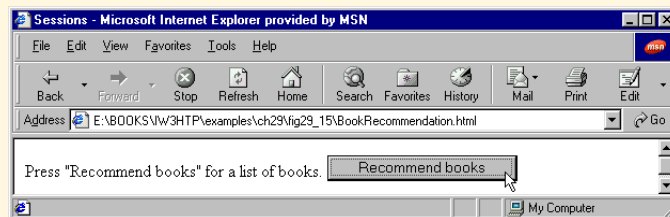


Fig. 29.14 HTML that interacts with the session tracking servlet to read the session information and return book recommendations to the user .

```
1 // Fig. 29.16: GuestBookServlet.java
2 // Three-Tier Example
3 import java.io.*;
4 import javax.servlet.*;
5 import javax.servlet.http.*;
6 import java.util.*;
7 import java.sql.*;
8
9 public class GuestBookServlet extends HttpServlet {
10     private Statement statement = null;
11     private Connection connection = null;
12     private String URL = "jdbc:odbc:GuestBook";
13
14     public void init( ServletConfig config )
15         throws ServletException
16     {
17         super.init( config );
18
19         try {
20             Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
21             connection =
22                 DriverManager.getConnection( URL, "", "" );
23         }
24         catch ( Exception e ) {
25             e.printStackTrace();
26             connection = null;
27         }
28     }
29
30     public void doPost( HttpServletRequest req,
31                       HttpServletResponse res )
32         throws ServletException, IOException
33     {
34         String email, firstName, lastName, company,
35             snailmailList, cppList, javaList, vbList,
36             iwwwList;
37
38         email = req.getParameter( "Email" );
39         firstName = req.getParameter( "FirstName" );
40         lastName = req.getParameter( "LastName" );
41         company = req.getParameter( "Company" );
42         snailmailList = req.getParameter( "mail" );
43         cppList = req.getParameter( "c_cpp" );
44         javaList = req.getParameter( "java" );
45         vbList = req.getParameter( "vb" );
46         iwwwList = req.getParameter( "iwww" );
47
48         res.setContentType( "text/html" );
49         PrintWriter output = res.getWriter();
50
51         if ( email.equals( "" ) ||
52             firstName.equals( "" ) ||
53             lastName.equals( "" ) ) {
```

Fig. 29.15 GuestBookServlet for mailing list registration (part 1 of 3).

© Copyright 2000 by Prentice Hall. All Rights Reserved.

For use only by instructors in classes for which *Java How to Program, Third Edition* is the required textbook.

```

54         output.println( "<H3> Please click the back " +
55                         "button and fill in all " +
56                         "fields.</H3>" );
57         output.close();
58         return;
59     }
60
61     /* Note: The GuestBook database actually contains fields
62     * Address1, Address2, City, State and Zip that are not
63     * used in this example. However, the insert into the
64     * database must still account for these fields. */
65     boolean success = insertIntoDB(
66         "'" + email + "','" + firstName + "','" + lastName +
67         "','" + company + "','" + snailmailList + "','" +
68         ( snailmailList != null ? "yes" : "no" ) + "','" +
69         ( cppList != null ? "yes" : "no" ) + "','" +
70         ( javaList != null ? "yes" : "no" ) + "','" +
71         ( vbList != null ? "yes" : "no" ) + "','" +
72         ( iwwwList != null ? "yes" : "no" ) + "'" );
73
74     if ( success )
75         output.print( "<H2>Thank you " + firstName +
76                     " for registering.</H2>" );
77     else
78         output.print( "<H2>An error occurred. " +
79                     "Please try again later.</H2>" );
80
81     output.close();
82 }
83
84 private boolean insertIntoDB( String stringtoinsert )
85 {
86     try {
87         statement = connection.createStatement();
88         statement.execute(
89             "INSERT INTO GuestBook values (" +
90             stringtoinsert + ");" );
91         statement.close();
92     }
93     catch ( Exception e ) {
94         System.err.println(
95             "ERROR: Problems with adding new entry" );
96         e.printStackTrace();
97         return false;
98     }
99
100     return true;
101 }
102
103 public void destroy()
104 {
105     try {
106         connection.close();

```

Fig. 29.15 GuestBookServlet for mailing list registration (part 2 of 3).

```
107     }  
108     catch( Exception e ) {  
109         System.err.println( "Problem closing the database" );  
110     }  
111 }  
112 }
```

Fig. 29.15 `GuestBookServlet` for mailing list registration (part 3 of 3).

```

1  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
2  <HTML>
3  <!-- Fig. 29.17: GuestBookForm.html -->
4
5  <HEAD>
6  <TITLE>Deitel Guest Book Form</TITLE>
7  </HEAD>
8
9  <BODY>
10 <H1>Guest Book</H1>
11 <FORM ACTION = "http://localhost:8080/servlet/GuestBookServlet"
12     METHOD = "POST"><PRE>
13 Email address (*): <INPUT TYPE = text NAME = Email>
14     First Name (*): <INPUT TYPE = text NAME = FirstName>
15     Last name (*): <INPUT TYPE = text NAME = LastName>
16     Company: <INPUT TYPE = text NAME = Company>
17
18     NOTE: (*) fields are required</PRE>
19 <P>Select mailing lists from which you want
20     to receive information<BR>
21 <INPUT TYPE = "checkbox" NAME = "mail" VALUE = "mail">
22 Snail Mail<BR>
23 <INPUT TYPE = "checkbox" NAME = "c_cpp" VALUE = "c_cpp">
24 <I>C++ How to Program & C How to Program</I><BR>
25 <INPUT TYPE = "checkbox" NAME = "java" VALUE = "java">
26 <I>Java How to Program</I><BR>
27 <INPUT TYPE = "checkbox" NAME = "vb" VALUE = "vb">
28 <I>Visual Basic How to Program</I><BR>
29 <INPUT TYPE = "checkbox" NAME = "iwww" VALUE = "iwww">
30 <I>Internet and World Wide Web How to Program</I><BR></P>
31 <INPUT TYPE = "SUBMIT" VALUE = "Submit">
32 </FORM>
33 </BODY>
34 </HTML>

```

Fig. 29.16 Web page that invokes the `GuestBookServlet`.

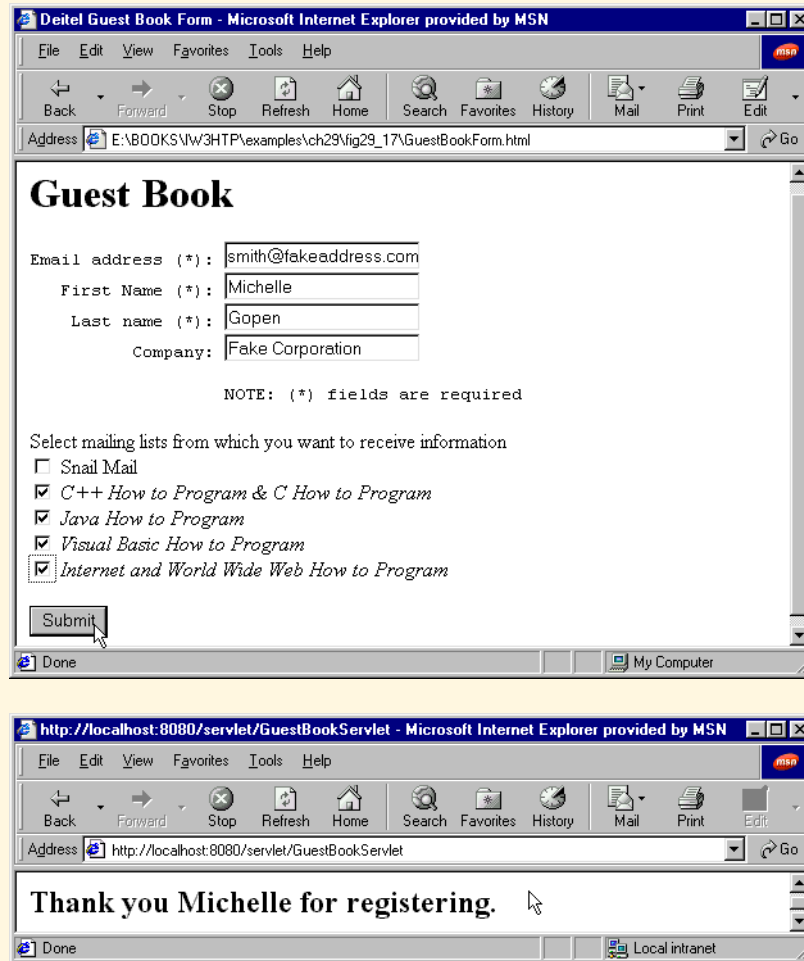


Fig. 29.16 Web page that invokes the `GuestBookServlet`.