Ordering Information:

Advanced Java 2 Platform How to Program

- View the complete **Table of Contents**
- Read the **Preface**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ONLINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**.

To learn more about our **Java and Advanced Java programming courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com**.

Note from the Authors: This article is an excerpt from Chapter 9, Section 9.1 of *Advanced Java 2 Platform How to Program*. The article introduces session tracking with cookies and provides a servlet example. Readers should be familiar with Java and Advanced Java concepts. The code examples included in this article show readers programming examples using the DEITEL™ signature LIVE-CODE™ Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

9.1 Session Tracking

Many e-businesses can personalize users' browsing experiences, tailoring Web pages to their users' individual preferences and letting users bypass irrelevant content. This is done by tracking the consumer's movement through the Internet and combining that data with information provided by the consumer, which could include billing information, interests and hobbies, among other things. *Personalization* is making it easier and more pleasant for many people to surf the Internet and find what they want. Consumers and companies can benefit from the unique treatment resulting from personalization. Providing content of special interest to your visitor can help establish a relationship that you can build upon each time that person returns to your site. Targeting consumers with personal offers, advertisements, promotions and services may lead to more customer loyalty—many customers enjoy the individual attention that a customized site provides. Originally, the Internet lacked personal assistance when compared with the individual service often experienced in bricks-and-mortar stores. Sophisticated technology helps many Web sites offer a personal touch

Descriptor element	Value
servlet *element*	
`servlet-name`	`redirect`
`description`	`Redirecting to static Web pages and other servlets.`
`servlet-class`	`com.deitel.advjhtp1.servlets.RedirectServlet`
servlet-mapping *element*	
`servlet-name`	`redirect`
`url-pattern`	`/redirect`

Fig. 9.1 Deployment descriptor information for servlet **`RedirectServlet`**.

to their visitors. For example, Web sites such as MSN.com and CNN.com allow you to customize their home page to suit your needs. Online shopping sites often customize their Web pages to individuals, and such sites must distinguish between clients so the company can determine the proper items and charge the proper amount for each client. Personalization is important for Internet marketing and for managing customer relationships to increase customer loyalty.

Hand in hand with the promise of personalization, however, comes the problem of *privacy invasion*. What if the e-business to which you give your personal data sells or gives those data to another organization without your knowledge? What if you do not want your movements on the Internet to be tracked by unknown parties? What if an unauthorized party gains access to your private data, such as credit-card numbers or medical history? These are some of the many questions that must be addressed by consumers, e-businesses and lawmakers alike.

As we have discussed, the request/response mechanism of the Web is based on HTTP. Unfortunately, HTTP is a *stateless protocol*—it does not support persistent information

that could help a Web server determine that a request is from a particular client. As far as a Web server is concerned, every request could be from the same client or every request could be from a different client. Thus, sites like MSN.com and CNN.com need a mechanism to identify individual clients. To help the server distinguish between clients, each client must identify itself to the server. There are a number of popular techniques for distinguishing between clients.

9.1.1 Cookies

A popular way to customize Web pages is via *cookies*. Browsers can store cookies on the user's computer for retrieval later in the same browsing session or in future browsing sessions. For example, cookies could be used in a shopping application to store unique identifiers for the users. When users add items to their online shopping carts or perform other tasks resulting in a request to the Web server, the server receives cookies containing unique identifiers for each user. The server then uses the unique identifier to locate the shopping carts and perform the necessary processing. Cookies could also be used to indicate the client's shopping preferences. When the servlet receives the client's next communication, the servlet can examine the cookie(s) it sent to the client in a previous communication, identify the client's preferences and immediately display products of interest to the client.

Cookies are text-based data that are sent by servlets (or other similar server-side technologies) as part of responses to clients. Every HTTP-based interaction between a client and a server includes a *header* containing information about the request (when the communication is from the client to the server) or information about the response (when the communication is from the server to the client). When an **`HttpServlet`** receives a request, the header includes information such as the request type (e.g., **`get`** or **`post`**) and the cookies that are sent by the server to be stored on the client machine. When the server formulates its response, the header information includes any cookies the server wants to store on the client computer and other information such as the MIME type of the response.

Testing and Debugging Tip 9.1

Some clients do not accept cookies. When a client declines a cookie, the Web site or the browser application can inform the client that the site may not function correctly without cookies enabled.

Depending on the *maximum age* of a cookie, the Web browser either maintains the cookie for the duration of the browsing session (i.e., until the user closes the Web browser) or stores the cookie on the client computer for future use. When the browser requests a resource from a server, cookies previously sent to the client by that server are returned to the server as part of the request formulated by the browser. Cookies are deleted automatically when they *expire* (i.e., reach their maximum age).

Figure 9.2 demonstrates cookies. The example allows the user to select a favorite programming language and **`post`** the choice to the server. The response is a Web page in which the user can select another favorite language or click a link to view a list of book recommendations. When the user selects the list of book recommendations, a **`get`** request is sent to the server. The cookies previously stored on the client are read by the servlet and used to form a Web page containing the book recommendations.

`CookieServlet` (Fig. 9.2) handles both the **`get`** and the **`post`** requests. The **`CookieSelectLanguage.html`** document of Fig. 9.3 contains four radio buttons (**C**,

C++, **Java** and **VB 6**) and a **Submit** button. When the user presses **Submit**, the **CookieServlet** is invoked with a **post** request. The servlet adds a cookie containing the selected language to the response header and sends an XHTML document to the client. Each time the user clicks **Submit**, a cookie is sent to the client.

Line 11 defines **Map books** as a **HashMap** (package **java.util**) in which we store key/value pairs that use the programming language as the key and the ISBN number of the recommended book as the value. The **CookieServlet init** method (line 14–20) populates books with four key/value pairs of books. Method **doPost** (lines 24–69) is invoked in response to the **post** request from the XHTML document of Fig. 9.3. Line 28 uses method **getParameter** to obtain the user's **language** selection (the value of the selected radio button on the Web page). Line 29 obtains the ISBN number for the selected language from **books**.

```
1  // Fig. 9.20: CookieServlet.java
2  // Using cookies to store data on the client computer.
3  package com.deitel.advjhtp1.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8  import java.util.*;
9
10 public class CookieServlet extends HttpServlet {
11    private final Map books = new HashMap();
12
13    // initialize Map books
14    public void init()
15    {
16       books.put( "C", "0130895725" );
17       books.put( "C++", "0130895717" );
18       books.put( "Java", "0130125075" );
19       books.put( "VB6", "0134569555" );
20    }
21
22    // receive language selection and send cookie containing
23    // recommended book to the client
24    protected void doPost( HttpServletRequest request,
25       HttpServletResponse response )
26          throws ServletException, IOException
27    {
28       String language = request.getParameter( "language" );
29       String isbn = books.get( language ).toString();
30       Cookie cookie = new Cookie( language, isbn );
31
32       response.addCookie( cookie );  // must precede getWriter
33       response.setContentType( "text/html" );
34       PrintWriter out = response.getWriter();
35
```

Fig. 9.2 Storing user data on the client computer with cookies (part 1 of 3).

```
36          // send XHTML page to client
37
38          // start XHTML document
39          out.println( "<?xml version = \"1.0\"?>" );
40
41          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
42             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
43             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
44
45          out.println(
46             "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
47
48          // head section of document
49          out.println( "<head>" );
50          out.println( "<title>Welcome to Cookies</title>" );
51          out.println( "</head>" );
52
53          // body section of document
54          out.println( "<body>" );
55          out.println( "<p>Welcome to Cookies! You selected " +
56             language + "</p>" );
57
58          out.println( "<p><a href = " +
59             "\"/advjhtp1/servlets/CookieSelectLanguage.html\">" +
60             "Click here to choose another language</a></p>" );
61
62          out.println( "<p><a href = \"/advjhtp1/cookies\">" +
63             "Click here to get book recommendations</a></p>" );
64          out.println( "</body>" );
65
66          // end XHTML document
67          out.println( "</html>" );
68          out.close();    // close stream
69       }
70
71       // read cookies from client and create XHTML document
72       // containing recommended books
73       protected void doGet( HttpServletRequest request,
74          HttpServletResponse response )
75             throws ServletException, IOException
76       {
77          Cookie cookies[] = request.getCookies();  // get cookies
78
79          response.setContentType( "text/html" );
80          PrintWriter out = response.getWriter();
81
82          // start XHTML document
83          out.println( "<?xml version = \"1.0\"?>" );
84
85          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
86             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
87             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
```

Fig. 9.2 Storing user data on the client computer with cookies (part 2 of 3).

```
88
89          out.println(
90             "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
91
92          // head section of document
93          out.println( "<head>" );
94          out.println( "<title>Recommendations</title>" );
95          out.println( "</head>" );
96
97          // body section of document
98          out.println( "<body>" );
99
100         // if there are any cookies, recommend a book for each ISBN
101         if ( cookies != null && cookies.length != 0 ) {
102            out.println( "<h1>Recommendations</h1>" );
103            out.println( "<p>" );
104
105            // get the name of each cookie
106            for ( int i = 0; i < cookies.length; i++ )
107               out.println( cookies[ i ].getName() +
108                  " How to Program. ISBN#: " +
109                  cookies[ i ].getValue() + "<br />" );
110
111            out.println( "</p>" );
112         }
113         else {   // there were no cookies
114            out.println( "<h1>No Recommendations</h1>" );

115            out.println( "<p>You did not select a language.</p>" );
116         }
117
118         out.println( "</body>" );
119
120         // end XHTML document
121         out.println( "</html>" );
122         out.close();    // close stream
123      }
124   }
```

Fig. 9.2 Storing user data on the client computer with cookies (part 3 of 3).

Line 30 creates a new ***Cookie*** object (package ***javax.servlet.http***), using the **language** and **isbn** values as the *cookie name* and *cookie value*, respectively. The cookie name identifies the cookie; the cookie value is the information associated with the cookie. Browsers that support cookies must be able to store a minimum of 20 cookies per Web site and 300 cookies per user. Browsers may limit the cookie size to 4K (4096 bytes). Each cookie stored on the client includes a domain. The browser sends a cookie only to the domain stored in the cookie.

Software Engineering Observation 9.1

Browser users can disable cookies, so Web applications that use cookies may not function properly for clients with cookies disabled.

```
4
5  <!-- Fig. 9.21: CookieSelectLanguage.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
```

MaxAge to indicate the number of seconds until the cookie expires.

Line 32 adds the cookie to the **response** with method ***addCookie*** of interface **HttpServletResponse**. Cookies are sent to the client as part of the HTTP header. The header information is always provided to the client first, so the cookies should be added to the **response** with **addCookie** before any data is written as part of the response. After the cookie is added, the servlet sends an XHTML document to the client (see the second screen capture of Fig. 9.3).

Common Programming Error 9.1

*Writing response data to the client before calling method **addCookie** to add a cookie to the response is a logic error. Cookies must be added to the header first.*

The XHTML document sent to the client in response to a **post** request includes a hyperlink that invokes method **doGet** (lines 73–123). The method reads any **Cookie**s that were written to the client in **doPost**. For each **Cookie** written, the servlet recommends a Deitel book on the subject. Up to four books are displayed on the Web page created by the servlet.

Line 77 retrieves the cookies from the client using **HttpServletRequest** method ***getCookies***, which returns an array of **Cookie** objects. When a **get** or **post** operation is performed to invoke a servlet, the cookies associated with that server's domain are automatically sent to the servlet.

If method **getCookies** does not return **null** (i.e., there were no cookies), lines 106–109 retrieve the name of each **Cookie** using **Cookie** method ***getName***, retrieve the value of each **Cookie** using **Cookie** method ***getValue*** and write a line to the client indicating the name of a recommended book and its ISBN number.

Software Engineering Observation 9.3

*Normally, each servlet class handles one request type (e.g., **get** or **post**, but not both).*

Figure 9.3 shows the XHTML document the user loads to select a language. When the user presses **Submit**, the value of the currently selected radio button is sent to the server as part of the **post** request to the **CookieServlet**, which we refer to as **cookies** in this example.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Fig. 9.3 **CookieSelectLanguage.html** document for selecting a programming language and posting the data to the **CookieServlet** (part 1 of 4).

```
4
5  <!-- Fig. 9.21: CookieSelectLanguage.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
```

Fig. 9.3 **`CookieSelectLanguage.html`** document for selecting a programming language and posting the data to the **`CookieServlet`** (part 1 of 4).

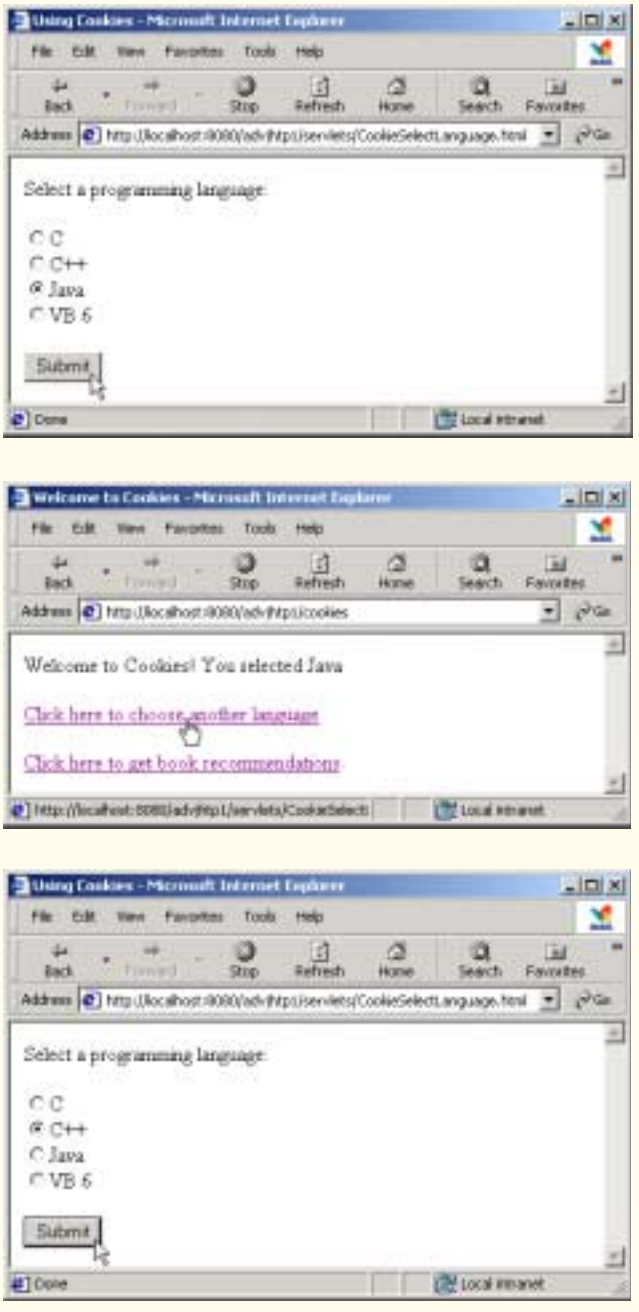

Fig. 9.3 **`CookieSelectLanguage.html`** document for selecting a programming language and posting the data to the **`CookieServlet`** (part 3 of 4).

Fig. 9.3 **`CookieSelectLanguage.html`** document for selecting a programming language and posting the data to the **`CookieServlet`** (part 4 of 4).

We use our **`advjhtp1`** context root to demonstrate the servlet of Fig. 9.2. Place **`CookieSelectLanguage.html`** in the **`servlets`** directory created previously. Place **`CookieServlet.class`** in the **`classes`** subdirectory of **`WEB-INF`** in the **`advjhtp1`** context root. Then, edit the **`web.xml`** deployment descriptor in the **`WEB-INF`** directory to include the information specified in Fig. 9.4. Restart Tomcat and type the following URL in your Web browser:

```
http://localhost:8080/advjhtp1/servlets/
CookieSelectLanguage.html
```

Select a language, and press the **Submit** button in the Web page to invoke the servlet.

Various **`Cookie`** methods are provided to manipulate the members of a **`Cookie`**. Some of these methods are listed in Fig. 9.5.

Descriptor element	Value
servlet *element*	
servlet-name	**cookies**
description	Using cookies to maintain state information.
servlet-class	**com.deitel.advjhtp1.servlets.CookieServlet**
servlet-mapping *element*	
servlet-name	**cookies**
url-pattern	**/cookies**

Fig. 9.4 Deployment descriptor information for servlet **CookieServlet**.

Method	Description
getComment()	Returns a **String** describing the purpose of the cookie (**null** if no comment has been set with **setComment**).
getDomain()	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.
getMaxAge()	Returns an **int** representing the maximum age of the cookie in seconds.
getName()	Returns a **String** containing the name of the cookie as set by the constructor.
getPath()	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.
getSecure()	Returns a **boolean** value indicating if the cookie should be transmitted using a secure protocol (**true**).
getValue()	Returns a **String** containing the value of the cookie as set with **setValue** or the constructor.
getVersion()	Returns an **int** containing the version of the cookie protocol used to create the cookie. A value of 0 (the default) indicates the original cookie protocol as defined by Netscape. A value of 1 indicates the current version, which is based on *Request for Comments (RFC) 2109*.
setComment(String)	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.)

Fig. 9.5 Important methods of class **Cookie** (part 1 of 2).

Method	Description
`setDomain( String )`	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 `".deitel.com"`, indicating that all servers ending with `.deitel.com` can receive this cookie.
`setMaxAge( int )`	Sets the maximum age of the cookie in seconds.
`setPath( String )`	Sets the "target" URL prefix indicating the directories on the server that lead to the services that can receive this cookie.
`setSecure( boolean )`	A `true` value indicates that the cookie should only be sent using a secure protocol.
`setValue( String )`	Sets the value of a cookie.
`setVersion( int )`	Sets the cookie protocol for this cookie.

Fig. 9.5 Important methods of class **`Cookie`** (part 2 of 2).