


---

# HANDLING COOKIES

## Topics in This Chapter

- 
- Understanding the benefits and drawbacks of cookies
  - Sending outgoing cookies
  - Receiving incoming cookies
  - Tracking repeat visitors
  - Specifying cookie attributes
  - Differentiating between session cookies and persistent cookies
  - Simplifying cookie usage with utility classes
  - Modifying cookie values
  - Remembering user preferences

### **Training courses from the book's author:**

<http://courses.coreservlets.com/>

- *Personally* developed and taught by Marty Hall
- Available onsite at *your* organization (any country)
- Topics and pace can be customized for your developers
- Also available periodically at public venues
- Topics include Java programming, beginning/intermediate servlets and JSP, advanced servlets and JSP, Struts, JSF/MyFaces, Ajax, GWT, Ruby/Rails and more. Ask for custom courses!

---

# Chapter

# 8

## Training courses from the book's author:

<http://courses.coreservlets.com/>

- *Personally* developed and taught by Marty Hall
- Available onsite at *your* organization (any country)
- Topics and pace can be customized for your developers
- Also available periodically at public venues
- Topics include Java programming, beginning/intermediate servlets and JSP, advanced servlets and JSP, Struts, JSF/MyFaces, Ajax, GWT, Ruby/Rails and more. Ask for custom courses!

Cookies are small bits of textual information that a Web server sends to a browser and that the browser later returns unchanged when visiting the same Web site or domain. By letting the server read information it sent the client previously, the site can provide visitors with a number of conveniences such as presenting the site the way the visitor previously customized it or letting identifiable visitors in without their having to reenter a password.

This chapter discusses how to explicitly set and read cookies from within servlets, and the next chapter shows how to use the servlet session tracking API (which can use cookies behind the scenes) to keep track of users as they move around to different pages within your site.

## 8.1 Benefits of Cookies

There are four typical ways in which cookies can add value to your site. We summarize these benefits below, then give details in the rest of the section.

- **Identifying a user during an e-commerce session.** This type of short-term tracking is so important that another API is layered on top of cookies for this purpose. See the next chapter for details.

- **Remembering usernames and passwords.** Cookies let a user log in to a site automatically, providing a significant convenience for users of unshared computers.
- **Customizing sites.** Sites can use cookies to remember user preferences.
- **Focusing advertising.** Cookies let the site remember which topics interest certain users and show advertisements relevant to those interests.

## Identifying a User During an E-commerce Session

Many online stores use a “shopping cart” metaphor in which users select items, add them to their shopping carts, then continue shopping. Since the HTTP connection is usually closed after each page is sent, when a user selects a new item to add to the cart, how does the store know that it is the same user who put the previous item in the cart? Persistent (keep-alive) HTTP connections do *not* solve this problem, since persistent connections generally apply only to requests made very close together in time, as when a browser asks for the images associated with a Web page. Besides, many older servers and browsers lack support for persistent connections. Cookies, however, *can* solve this problem. In fact, this capability is so useful that servlets have an API specifically for session tracking, and servlet and JSP authors don’t need to manipulate cookies directly to take advantage of it. Session tracking is discussed in Chapter 9.

## Remembering Usernames and Passwords

Many large sites require you to register to use their services, but it is inconvenient to remember and enter the username and password each time you visit. Cookies are a good alternative for low-security sites. When a user registers, a cookie containing a unique user ID is sent to him. When the client reconnects at a later date, the user ID is returned automatically, the server looks it up, determines it belongs to a registered user that chose autologin, and permits access without an explicit username and password. The site might also store the user’s address, credit card number, and so forth in a database and use the user ID from the cookie as the key to retrieve the data. This approach prevents the user from having to reenter the data each time.

For example, when Marty travels to companies to give onsite JSP and servlet training courses, he typically checks both [travelocity.com](http://travelocity.com) and [expedia.com](http://expedia.com) for flight information. These both require usernames and passwords to search flight schedules, but have different rules about which characters are legal in usernames and how many characters are required for passwords. So, Marty has a difficult time remembering

how to log in. Fortunately, both sites use the cookie scheme described in the preceding paragraph, simplifying Marty's access from his personal desktop or laptop machine.

## Customizing Sites

Many "portal" sites let you customize the look of the main page. They might let you pick which weather report you want to see (yes, it is still raining in Seattle), what stock symbols should be displayed (yes, your stock is still way down), what sports results you care about (yes, the Orioles are still losing), how search results should be displayed (yes, you want to see more than one result per page), and so forth. Since it would be inconvenient for you to have to set up your page each time you visit their site, they use cookies to remember what you wanted. For simple settings, the site could accomplish this customization by storing the page settings directly in the cookies. For more complex customization, however, the site just sends the client a unique identifier and keeps a server-side database that associates identifiers with page settings.

## Focusing Advertising

Most advertiser-funded Web sites charge their advertisers much more for displaying "directed" (or "focused") ads than for displaying "random" ads. Advertisers are generally willing to pay much more to have their ads shown to people that are known to have some interest in the general product category. Sites reportedly charge advertisers as much as 30 times more for directed ads than for random ads. For example, if you go to a search engine and do a search on "Java Servlets," the search site can charge an advertiser much more for showing you an ad for a servlet development environment than for an ad for an online travel agent specializing in Indonesia. On the other hand, if the search had been for "Java Hotels," the situation would be reversed.

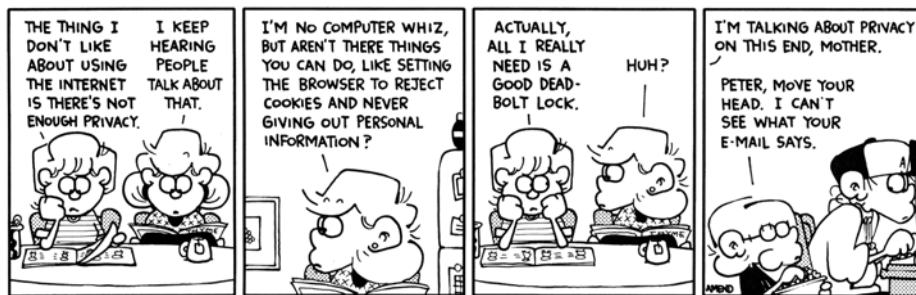
Without cookies, sites have to show a random ad when you first arrive and haven't yet performed a search, as well as when you search on something that doesn't match any ad categories. With cookies, they can identify your interests by remembering your previous searches. Since this approach enables them to show directed ads on visits to their home page as well as for their results page, it nearly doubles their advertising revenue.

## 8.2 Some Problems with Cookies

Providing convenience to the user and added value to the site owner is the purpose behind cookies. And despite much misinformation, cookies are not a serious security threat. Cookies are *never* interpreted or executed in any way and thus cannot be used

to insert viruses or attack your system. Furthermore, since browsers generally only accept 20 cookies per site and 300 cookies total, and since browsers can limit each cookie to 4 kilobytes, cookies cannot be used to fill up someone's disk or launch other denial-of-service attacks.

However, even though cookies don't present a serious *security* threat, they can present a significant threat to *privacy*.



FOXTROT © 1998 Bill Amend. Reprinted with permission of UNIVERSAL PRESS SYNDICATE. All rights reserved.

First, some people don't like the fact that search engines can remember what they previously searched for. For example, they might search for job openings or sensitive health data and don't want some banner ad tipping off their coworkers or boss next time they do a search. Besides, a search engine need not use a banner ad: a poorly designed one could display a textarea listing your most recent queries ("Jobs anywhere except at this stupid company!"; "Will my SARS infection kill my coworkers?"; etc.). A coworker could see this information if they visited the search engine for your computer or if they looked over your shoulder when you visited it.

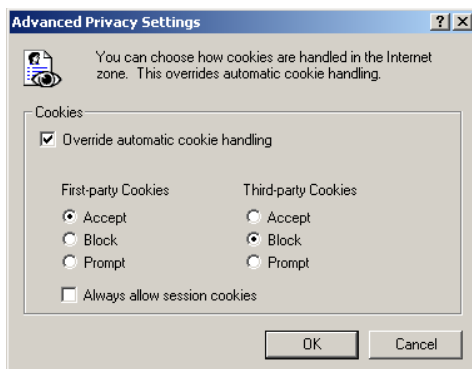
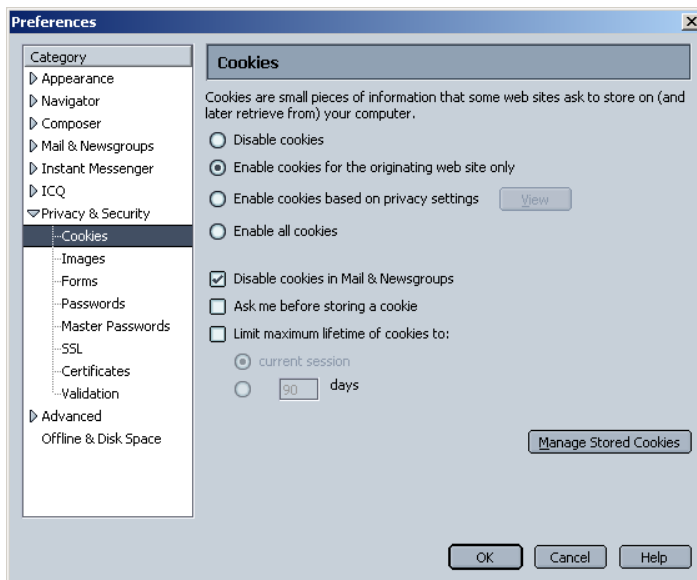
Even worse, two sites can share data on a user by each loading small images off the same third-party site, where that third party uses cookies and shares the data with both original sites. For example, suppose that both `some-search-site.com` and `some-random-site.com` wanted to display directed ads from `some-ad-site.com` based on what the user searched for at `some-search-site.com`. If the user searched for "Java Servlets," the search engine at `some-search-site.com` could return a page with the following image link:

```
<IMG SRC="http://some-ad-site.com/banner?data=Java+Servlets" ...>
```

Since the browser will make an HTTP connection to `some-ad-site.com`, `some-ad-site.com` can return a persistent cookie to the browser. Next, `some-random-site.com` could return an image link like this:

```
<IMG SRC="http://some-ad-site.com/banner" ...>
```

Since the browser will reconnect to `some-ad-site.com`—a site from which it got cookies earlier—it will return the cookie it previously received. Assuming that `some-ad-site.com` sent a unique cookie value and, in its database, associated that cookie value with the “Java Servlets” search, `some-ad-site` can return a directed banner ad even though it is the user’s first visit to `some-random-site`. The `doubleclick.net` service was the most famous early example of this technique. (Recent versions of Netscape and Internet Explorer, however, have a nice feature that lets you refuse cookies from sites other than that to which you connected, but without disabling cookies altogether. See Figure 8–1.)



**Figure 8–1** Cookie customization settings for Netscape (top) and Internet Explorer (bottom).

This trick of associating cookies with images can even be exploited through email if you use an HTML-enabled email reader that “supports” cookies and is associated with a browser. Thus, people could send you email that loads images, attach cookies to those images, and then identify you (email address and all) if you subsequently visit their Web site. Boo.

A second privacy problem occurs when sites rely on cookies for overly sensitive data. For example, some of the big online bookstores use cookies to remember your registration information and let you order without reentering much of your personal information. This is not a particular problem since they don’t actually display your complete credit card number and only let you send books to an address that was specified when you *did* enter the credit card in full or use the username and password. As a result, someone using your computer (or stealing your cookie file) could do no more harm than sending a big book order to your address, where the order could be refused. However, other companies might not be so careful, and an attacker who gained access to someone’s computer or cookie file could get online access to valuable personal information. Even worse, incompetent sites might embed credit card or other sensitive information directly in the cookies themselves, rather than using innocuous identifiers that are linked to real users only on the server. This embedding is dangerous, since most users don’t view leaving their computer unattended in their office as being tantamount to leaving their credit card sitting on their desk.

The point of this discussion is twofold:

1. Due to real and perceived privacy problems, some users turn off cookies. So, even when you use cookies to give added value to a site, whenever possible your site shouldn’t *depend* on them. Dependence on cookies is difficult to avoid in some situations, but if you can provide reasonable functionality for users without cookies enabled, so much the better.
2. As the author of servlets or JSP pages that use cookies, you should be careful not to use cookies for particularly sensitive information, since this would open users up to risks if somebody accessed the user’s computer or cookie files.

### 8.3 Deleting Cookies

You will probably find it easier to experiment with the examples in this chapter if you periodically delete your cookies (or at least the cookies that are associated with `localhost` or whatever host your server is running on).

To delete your cookies in Internet Explorer, start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies. To selectively delete cookies, press Settings, then View Files (cookie files have names that begin with Cookie:, but it is easier to find them if you choose Delete Files before View Files). See Figure 8–2.

To delete your cookies in Netscape, start at the Edit menu, then choose Preferences, Privacy and Security, and Cookies. Press the Manage Stored Cookies button to view or delete any or all of your cookies. Again, see Figure 8–2.

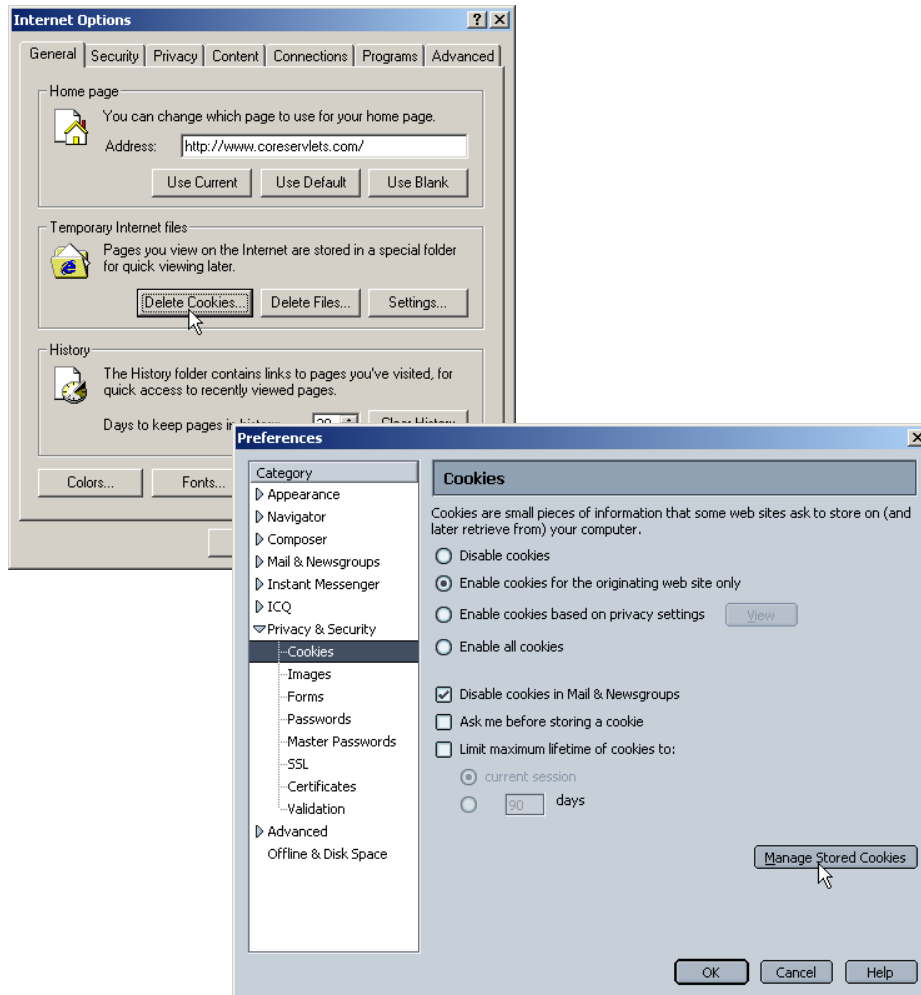


Figure 8–2 Deleting cookies in Internet Explorer and Netscape.



## 8.4 Sending and Receiving Cookies

To send cookies to the client, a servlet should use the `Cookie` constructor to create one or more cookies with designated names and values, set any optional attributes with `cookie.setXxx` (readable later by `cookie.getXxx`), and insert the cookies into the HTTP response headers with `response.addCookie`.

To read incoming cookies, a servlet should call `request.getCookies`, which returns an array of `Cookie` objects corresponding to the cookies the browser has associated with your site (`null` if there are no cookies in the request). In most cases, the servlet should then loop down this array calling `getName` on each cookie until it finds the one whose name matches the name it was searching for, then call `getValue` on that `Cookie` to see the value associated with the name. Each of these topics is discussed in more detail in the following subsections.

### Sending Cookies to the Client

Sending cookies to the client involves three steps (summarized below with details in the following subsections).

1. **Creating a `Cookie` object.** You call the `Cookie` constructor with a cookie name and a cookie value, both of which are strings.
2. **Setting the maximum age.** If you want the browser to store the cookie on disk instead of just keeping it in memory, you use `setMaxAge` to specify how long (in seconds) the cookie should be valid.
3. **Placing the `Cookie` into the HTTP response headers.** You use `response.addCookie` to accomplish this. If you forget this step, no cookie is sent to the browser!

### Creating a `Cookie` Object

You create a cookie by calling the `Cookie` constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain white space or any of the following characters:

```
[ ] ( ) = , " / ? @ : ;
```

For example, to create a cookie named `userID` with a value `a1234`, you would use the following.

```
Cookie c = new Cookie("userID", "a1234");
```

## Setting the Maximum Age

If you create a cookie and send it to the browser, by default it is a session-level cookie: a cookie that is stored in the browser's memory and deleted when the user quits the browser. If you want the browser to store the cookie on disk, use `setMaxAge` with a time in seconds, as below.

```
c.setMaxAge(60*60*24*7); // One week
```

Since you could use the session-tracking API (Chapter 9) to simplify most tasks for which you use session-level cookies, you almost always use the `setMaxAge` method when using the `Cookie` API.

Setting the maximum age to 0 instructs the browser to delete the cookie.

### Core Approach

---

*When you create a `Cookie` object, you should normally call `setMaxAge` before sending the cookie to the client.*

---



Note that `setMaxAge` is not the only `Cookie` characteristic that you can modify. The other, less frequently used characteristics are discussed in Section 8.6 (Using `Cookie` Attributes).

## Placing the Cookie in the Response Headers

By creating a `Cookie` object and calling `setMaxAge`, all you have done is manipulate a data structure in the server's memory. You haven't actually sent anything to the browser. If you don't send the cookie to the client, it has no effect. This may seem obvious, but a common mistake by beginning developers is to create and manipulate `Cookie` objects but fail to send them to the client.

### Core Warning

---

*Creating and manipulating a `Cookie` object has no effect on the client. You must explicitly send the cookie to the client with `response.addCookie`.*

---



To send the cookie, insert it into a `Set-Cookie` HTTP response header by means of the `addCookie` method of `HttpServletResponse`. The method is called `addCookie`, not `setCookie`, because any previously specified `Set-Cookie` headers

are left alone and a new header is set. Also, remember that the response headers must be set before any document content is sent to the client.

Here is an example:

```
Cookie userCookie = new Cookie("user", "uid1234");
userCookie.setMaxAge(60*60*24*365); // Store cookie for 1 year
response.addCookie(userCookie);
```

## Reading Cookies from the Client

To send a cookie *to* the client, you create a `Cookie`, set its maximum age (usually), then use `addCookie` to send a `Set-Cookie` HTTP response header. To read the cookies that come back *from* the client, you should perform the following two tasks, which are summarized below and then described in more detail in the following subsections.

1. Call `request.getCookies`. This yields an array of `Cookie` objects.
2. Loop down the array, calling `getName` on each one until you find the cookie of interest. You then typically call `getValue` and use the value in some application-specific way.

### Call `request.getCookies`

To obtain the cookies that were sent by the browser, you call `getCookies` on the `HttpServletRequest`. This call returns an array of `Cookie` objects corresponding to the values that came in on the `Cookie` HTTP request headers. If the request contains no cookies, `getCookies` should return `null`. Note, however, that an old version of Apache Tomcat (version 3.x) had a bug whereby it returned a zero-length array instead of `null`.

### Loop Down the Cookie Array

Once you have the array of cookies, you typically loop down it, calling `getName` on each `Cookie` until you find one matching the name you have in mind. Remember that cookies are specific to your host (or domain), not your servlet (or JSP page). So, although your servlet might send a single cookie, you could get many irrelevant cookies back. Once you find the cookie of interest, you typically call `getValue` on it and finish with some processing specific to the resultant value. For example:

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie cookie = cookies[i];
        if (cookieName.equals(cookie.getName())) {
```

```
        doSomethingWith(cookie.getValue());
    }
}
}
```

This is such a common process that, in Section 8.8, we present two utilities that simplify retrieving a cookie or cookie value that matches a designated cookie name.

## 8.5 Using Cookies to Detect First-Time Visitors

Suppose that, at your site, you want to display a prominent banner to first-time visitors, telling them to register. But, you don't want to clutter up the display showing a useless banner to return visitors.

A cookie is the perfect way to differentiate first-timers from repeat visitors. Check for the existence of a uniquely named cookie; if it is there, the client is a repeat visitor. If the cookie is not there, the visitor is a newcomer, and you should set an outgoing "this user has been here before" cookie.

Although this is a straightforward idea, there is one important point to note: you cannot determine if the user is a newcomer by the mere existence of entries in the cookie array. Many beginning servlet programmers erroneously use the following approach.

```
Cookie[] cookies = request.getCookies();
if (cookies == null) {
    doStuffForNewbie();           // Correct.
} else {
    doStuffForReturnVisitor(); // Incorrect.
}
```

Wrong! Sure, if the cookie array is `null`, the client is a newcomer (at least as far as you can tell—he could also have deleted or disabled cookies). But, if the array is non-`null`, it merely shows that the client has been to your *site* (or domain—see `setDomain` in the next section), not that they have been to your *servlet*. Other servlets, JSP pages, and non-Java Web applications can set cookies, and any of those cookies could get returned to your browser, depending on the path settings (see `setPath` in the next section).

Listing 8.1 illustrates the correct approach: checking for a specific cookie. Figures 8-3 and 8-4 show the results of initial and subsequent visits.

**Listing 8.1** RepeatVisitor.java

```
package coreservlets;

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

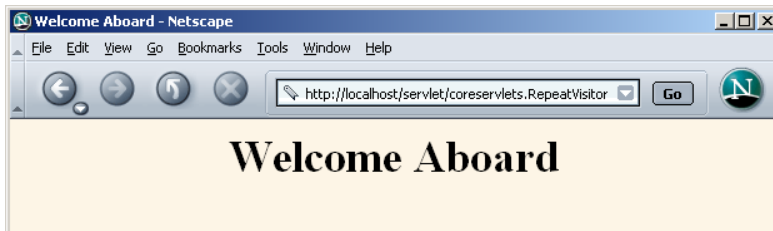
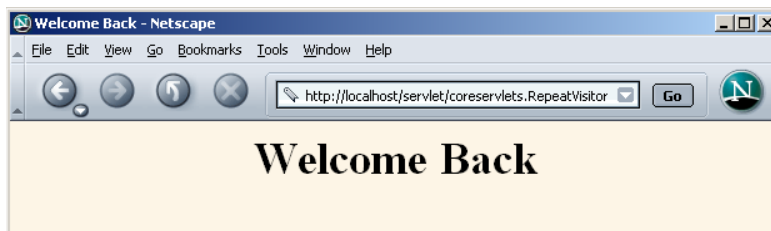
/** Servlet that says "Welcome aboard" to first-time
 * visitors and "Welcome back" to repeat visitors.
 * Also see RepeatVisitor2 for variation that uses
 * cookie utilities from later in this chapter.
 */

public class RepeatVisitor extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        boolean newbie = true;
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for(int i=0; i<cookies.length; i++) {
                Cookie c = cookies[i];
                if ((c.getName().equals("repeatVisitor")) &&
                    // Could omit test and treat cookie name as a flag
                    (c.getValue().equals("yes"))) {
                    newbie = false;
                    break;
                }
            }
        }
        String title;
        if (newbie) {
            Cookie returnVisitorCookie =
                new Cookie("repeatVisitor", "yes");
            returnVisitorCookie.setMaxAge(60*60*24*365); // 1 year
            response.addCookie(returnVisitorCookie);
            title = "Welcome Aboard";
        } else {
            title = "Welcome Back";
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC \"/>";

```

**Listing 8.1** RepeatVisitor.java (*continued*)

```
out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
    "</BODY></HTML>");
}
}
```

**Figure 8-3** First visit by a client to the RepeatVisitor servlet.**Figure 8-4** Subsequent visits by a client to the RepeatVisitor servlet.

## 8.6 Using Cookie Attributes

Before adding the cookie to the outgoing headers, you can set various characteristics of the cookie by using the following `setXxx` methods, where `Xxx` is the name of the attribute you want to specify.

Although each `setXxx` method has a corresponding `getXxx` method to retrieve the attribute value, note that the attributes are part of the header sent from the server to the browser; they are *not* part of the header returned by the browser to the

server. Thus, except for name and value, the cookie attributes apply only to *outgoing* cookies from the server to the client; they aren't set on cookies that come *from* the browser to the server. So, don't expect these attributes to be available in the cookies you get by means of `request.getCookies`. This means that you can't implement continually changing cookie values merely by setting a maximum age on a cookie once, sending it out, finding the appropriate cookie in the incoming array on the next request, reading the value, modifying it, and storing it back in the `Cookie`. You have to call `setMaxAge` again each time (and, of course, pass the `Cookie` to `response.addCookie`).

Here are the methods that set the cookie attributes.

**public void setComment(String comment)****public String getComment()**

These methods specify or look up a comment associated with the cookie. With version 0 cookies (see the upcoming entry on `setVersion` and `getVersion`), the comment is used purely for informational purposes on the server; it is not sent to the client.

**public void setDomain(String domainPattern)****public String getDomain()**

These methods set or retrieve the domain to which the cookie applies. Normally, the browser returns cookies only to the exact same hostname that sent the cookies. For instance, cookies sent from a servlet at `bali.vacations.com` would not normally get returned by the browser to pages at `queensland.vacations.com`. If the site wanted this to happen, the servlets could specify `cookie.setDomain(".vacations.com")`. To prevent servers from setting cookies that apply to hosts outside their domain, the specified domain must meet the following requirements: it must start with a dot (e.g., `.coreservlets.com`); it must contain two dots for noncountry domains like `.com`, `.edu`, and `.gov`; and it must contain three dots for country domains like `.co.uk` and `.edu.es`.

**public void setMaxAge(int lifetime)****public int getMaxAge()**

These methods tell how much time (in seconds) should elapse before the cookie expires. A negative value, which is the default, indicates that the cookie will last only for the current browsing session (i.e., until the user quits the browser) and will not be stored on disk. See the `LongLivedCookie` class (Listing 8.4), which defines a subclass of `Cookie` with a maximum age automatically set one year in the future. Specifying a value of 0 instructs the browser to delete the cookie.

**public String getName()**

The `getName` method retrieves the name of the cookie. The name and the value are the two pieces you virtually *always* care about. However, since the name is supplied to the `Cookie` constructor, there is *no* `setName` method; you cannot change the name once the cookie is created. On the other hand, `getName` is used on almost every cookie received by the server. Since the `getCookies` method of `HttpServletRequest` returns an array of `Cookie` objects, a common practice is to loop down the array, calling `getName` until you have a particular name, then to check the value with `getValue`. For an encapsulation of this process, see the `getCookieValue` method shown in Listing 8.3.

**public void setPath(String path)  
public String getPath()**

These methods set or get the path to which the cookie applies. If you don't specify a path, the browser returns the cookie only to URLs in or below the directory containing the page that sent the cookie. For example, if the server sent the cookie from `http://ecommerce.site.com/toys/specials.html`, the browser would send the cookie back when connecting to `http://ecommerce.site.com/toys/bikes/beginners.html`, but not to `http://ecommerce.site.com/cds/classical.html`. The `setPath` method can specify something more general. For example, `cookie.setPath("/")` specifies that *all* pages on the server should receive the cookie. The path specified must include the current page; that is, you may specify a more general path than the default, but not a more specific one. So, for example, a servlet at `http://host/store/cust-service/request` could specify a path of `/store/` (since `/store/` includes `/store/cust-service/`) but not a path of `/store/cust-service/returns/` (since this directory does not include `/store/cust-service/`).

**Core Approach**

---

*To specify that a cookie apply to all URLs on your site, use `cookie.setPath("/")`.*

---

**public void setSecure(boolean secureFlag)  
public boolean getSecure()**

This pair of methods sets or gets the boolean value indicating whether the cookie should only be sent over encrypted (i.e., SSL) connections. The default is `false`; the cookie should apply to all connections.



```
public void setValue(String cookieValue)  
public String getValue()
```

The `setValue` method specifies the value associated with the cookie; `getValue` looks it up. Again, the name and the value are the two parts of a cookie that you *almost always* care about, although in a few cases, a name is used as a boolean flag and its value is ignored (i.e., the existence of a cookie with the designated name is all that matters). However, since the cookie value is supplied to the `Cookie` constructor, `setValue` is typically reserved for cases when you change the values of incoming cookies and then send them back out. For an example, see Section 8.10 (Modifying Cookie Values: Tracking User Access Counts).

```
public void setVersion(int version)  
public int getVersion()
```

These methods set and get the cookie protocol version with which the cookie complies. Version 0, the default, follows the original Netscape specification ([http://wp.netscape.com/newsref/std/cookie\\_spec.html](http://wp.netscape.com/newsref/std/cookie_spec.html)). Version 1, not yet widely supported, adheres to RFC 2109 (retrieve RFCs from the archive sites listed at <http://www.rfc-editor.org/>).

## 8.7 Differentiating Session Cookies from Persistent Cookies

This section illustrates the use of the cookie attributes by contrasting the behavior of cookies with and without a maximum age. Listing 8.2 shows the `CookieTest` servlet, a servlet that performs two tasks:

1. First, the servlet sets six outgoing cookies. Three have no explicit age (i.e., have a negative value by default), meaning that they should apply only in the current browsing session—until the user restarts the browser. The other three use `setMaxAge` to stipulate that the browser should write them to disk and that they should persist for the next hour, regardless of whether the user restarts the browser or reboots the computer to initiate a new browsing session.
2. Second, the servlet uses `request.getCookies` to find all the incoming cookies and display their names and values in an HTML table.

Figure 8–5 shows the result of the initial visit, Figure 8–6 shows a visit immediately after that, and Figure 8–7 shows the result of a visit after the user restarts the browser.

**Listing 8.2** CookieTest.java

```
package coreservlets;

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

/** Creates a table of the cookies associated with
 * the current page. Also sets six cookies: three
 * that apply only to the current session
 * (regardless of how long that session lasts)
 * and three that persist for an hour (regardless
 * of whether the browser is restarted).
 */

public class CookieTest extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        for(int i=0; i<3; i++) {
            // Default maxAge is -1, indicating cookie
            // applies only to current browsing session.
            Cookie cookie = new Cookie("Session-Cookie-" + i,
                "Cookie-Value-S" + i);
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-" + i,
                "Cookie-Value-P" + i);
            // Cookie is valid for an hour, regardless of whether
            // user quits browser, reboots computer, or whatever.
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN">\n";
        String title = "Active Cookies";
```

## Listing 8.2 CookieTest.java (continued)

```

out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
    "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
    "<TR BGCOLOR=\"#FFAD00\">\n" +
    " <TH>Cookie Name\n" +
    " <TH>Cookie Value");
Cookie[] cookies = request.getCookies();
if (cookies == null) {
    out.println("<TR><TH COLSPAN=2>No cookies");
} else {
    Cookie cookie;
    for(int i=0; i<cookies.length; i++) {
        cookie = cookies[i];
        out.println("<TR>\n" +
            " <TD>" + cookie.getName() + "\n" +
            " <TD>" + cookie.getValue());
    }
}
out.println("</TABLE></BODY></HTML>");
}
}

```



**Figure 8-5** Result of initial visit to the `CookieTest` servlet. This is the same result as when visiting the servlet, quitting the browser, waiting an hour, and revisiting the servlet.



**Figure 8-6** Result of revisiting the `CookieTest` servlet within an hour of the original visit, in the same browser session (browser stayed open between the original visit and the visit shown here).



**Figure 8-7** Result of revisiting the `CookieTest` servlet within an hour of the original visit, in a different browser session (browser was restarted between the original visit and the visit shown here).

## 8.8 Basic Cookie Utilities

This section presents some simple but useful utilities for dealing with cookies.

### Finding Cookies with Specified Names

Listing 8.3 shows two static methods in the `CookieUtilities` class that simplify the retrieval of a cookie or cookie value, given a cookie name. The `getCookieValue` method loops through the array of available `Cookie` objects, returning the value of any `Cookie` whose name matches the input. If there is no match, the designated default value is returned. So, for example, our typical approach for dealing with cookies is as follows:

```
String color =  
    CookieUtilities.getCookieValue(request, "color", "black");  
  
String font =  
    CookieUtilities.getCookieValue(request, "font", "Arial");
```

The `getCookie` method also loops through the array comparing names but returns the actual `Cookie` object instead of just the value. That method is for cases when you want to do something with the `Cookie` other than just read its value. The `getCookieValue` method is more commonly used, but, for example, you might use `getCookie` in lieu of `getCookieValue` if you wanted to put a new value into the cookie and send it back out again. Just don't forget that if you use this approach, you have to respecify any cookie attributes such as date, path, and domain: these attributes are not set for incoming cookies.

#### Listing 8.3 `CookieUtilities.java`

```
package coreservlets;  
  
import javax.servlet.*;  
import javax.servlet.http.*;  
  
/** Two static methods for use in cookie handling. */  
  
public class CookieUtilities {
```

**Listing 8.3** CookieUtilities.java (continued)

```
/** Given the request object, a name, and a default value,
 * this method tries to find the value of the cookie with
 * the given name. If no cookie matches the name,
 * the default value is returned.
 */

public static String getCookieValue
    (HttpServletRequest request,
     String cookieName,
     String defaultValue) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName())) {
                return(cookie.getValue());
            }
        }
    }
    return(defaultValue);
}

/** Given the request object and a name, this method tries
 * to find and return the cookie that has the given name.
 * If no cookie matches the name, null is returned.
 */

public static Cookie getCookie(HttpServletRequest request,
    String cookieName) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName())) {
                return(cookie);
            }
        }
    }
    return(null);
}
}
```

## Creating Long-Lived Cookies

Listing 8.4 shows a small class that you can use instead of `Cookie` if you want your cookie to automatically persist for a year when the client quits the browser. This class (`LongLivedCookie`) merely extends `Cookie` and calls `setMaxAge` automatically.

### Listing 8.4 LongLivedCookie.java

```
package coreservlets;

import javax.servlet.http.*;

/** Cookie that persists 1 year. Default Cookie doesn't
 *  * persist past current browsing session.
 *  */

public class LongLivedCookie extends Cookie {
    public static final int SECONDS_PER_YEAR = 60*60*24*365;

    public LongLivedCookie(String name, String value) {
        super(name, value);
        setMaxAge(SECONDS_PER_YEAR);
    }
}
```

## 8.9 Putting the Cookie Utilities into Practice

Listing 8.5 redoes the `RepeatVisitor` servlet of Listing 8.1. The new version (`RepeatVisitor2`) has the same functionality as the old version: it says “Welcome Aboard” to first-time visitors and “Welcome Back” to repeat visitors. However, it uses the cookie utilities of Section 8.8 to simplify the code in two ways:

1. Instead of calling `request.getCookies` and looping down that array examining each name, it merely calls `CookieUtilities.getCookieValue`.
2. Instead of creating a `Cookie` object, calculating the number of seconds in a year, and then calling `setMaxAge`, it merely creates a `LongLivedCookie` object.

Figures 8–8 and 8–9 show the results.

**Listing 8.5** RepeatVisitor2.java

```
package coreservlets;

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

/** A variation of the RepeatVisitor servlet that uses
 *  CookieUtilities.getCookieValue and LongLivedCookie
 *  to simplify the code.
 */

public class RepeatVisitor2 extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {
        boolean newbie = true;
        String value =
            CookieUtilities.getCookieValue(request, "repeatVisitor2",
                                         "no");
        if (value.equals("yes")) {
            newbie = false;
        }
        String title;
        if (newbie) {
            LongLivedCookie returnVisitorCookie =
                new LongLivedCookie("repeatVisitor2", "yes");
            response.addCookie(returnVisitorCookie);
            title = "Welcome Aboard";
        } else {
            title = "Welcome Back";
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"Transitional//EN\">\n";
        out.println(docType +
                   "<HTML>\n" +
                   "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                   "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                   "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                   "</BODY></HTML>");
    }
}
```



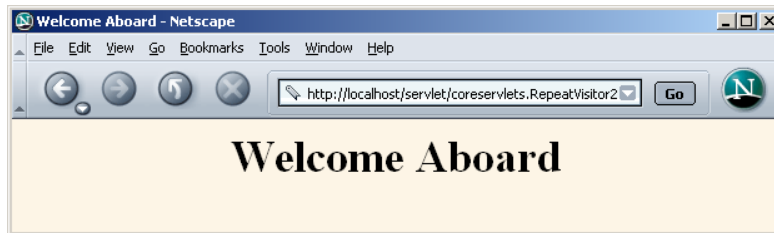


Figure 8-8 First visit by a client to the RepeatVisitor2 servlet.

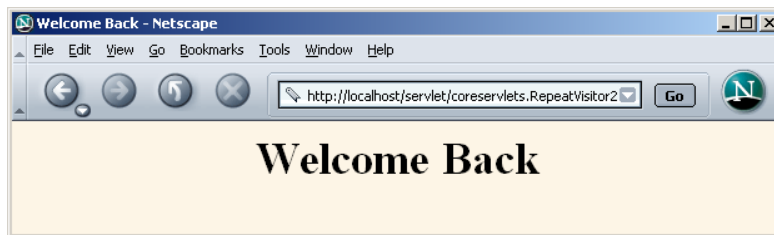


Figure 8-9 Subsequent visit by a client to the RepeatVisitor2 servlet.

## 8.10 Modifying Cookie Values: Tracking User Access Counts

In the previous examples, we sent a cookie to the user only on the first visit. Once the cookie had a value, we never changed it. This approach of a single cookie value is surprisingly common since cookies frequently contain nothing but unique user identifiers: all the real user data is stored in a database—the user identifier is merely the database key.

But what if you want to periodically change the value of a cookie? How do you do so?

- To *replace* a previous cookie value, send the same cookie name with a different cookie value. If you actually use the incoming `Cookie` objects, don't forget to do `response.addCookie`; merely calling `setValue` is not sufficient. You also need to reapply any relevant cookie attributes by calling `setMaxAge`, `setPath`, etc.—cookie attributes are not specified for incoming cookies. Reapplying these attributes means that reusing the incoming `Cookie` objects saves you little, so many developers don't bother.

- To instruct the browser to *delete* a cookie, use `setMaxAge` to assign a maximum age of 0.

Listing 8.6 presents a servlet that keeps track of how many times each client has visited the page. It does this by making a cookie whose name is `accessCount` and whose value is the actual count. To accomplish this task, the servlet needs to repeatedly replace the cookie value by resending a cookie with the identical name.

Figure 8–10 shows some typical results.

**Listing 8.6** ClientAccessCounts.java

```
package coreservlets;

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

/** Servlet that prints per-client access counts. */

public class ClientAccessCounts extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String countString =
            CookieUtilities.getCookieValue(request,
                "accessCount",
                "1");

        int count = 1;
        try {
            count = Integer.parseInt(countString);
        } catch (NumberFormatException nfe) { }
        LongLivedCookie c =
            new LongLivedCookie("accessCount",
                String.valueOf(count+1));
        response.addCookie(c);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Access Count Servlet";
        String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"Transitional//EN\">\n";
    }
}
```

**Listing 8.6** ClientAccessCounts.java (continued)

```

out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<CENTER>\n" +
    "<H1>" + title + "</H1>\n" +
    "<H2>This is visit number " +
    count + " by this browser.</H2>\n" +
    "</CENTER></BODY></HTML>");
    }
}

```



**Figure 8-10** Users each see their own access count. Also, Internet Explorer and Netscape maintain cookies separately, so the same user sees independent access counts with the two browsers.

## 8.11 Using Cookies to Remember User Preferences

One of the most common applications of cookies is to use them to “remember” user preferences. For simple user settings, as here, the preferences can be stored directly in the cookies. For more complex applications, the cookie typically contains a unique user identifier and the preferences are stored in a database.

Listing 8.7 presents a servlet that creates an input form with the following characteristics.

- **The form is redisplayed if it is incomplete when submitted.** The form sends data to a second servlet (Listing 8.8) that checks whether any of the designated request parameters is missing, then stores the parameter values in cookies. If no parameter is missing, the second servlet displays the parameter values. If a parameter is missing, the second servlet redirects the user to the original servlet so that the form can be redisplayed. The original servlet maintains the user’s previously entered values by extracting them from the cookies.
- **The form remembers previous entries.** The fields are prepopulated with whatever values the user entered on the most recent request.

Figures 8–11 through 8–13 show some typical results.

### Listing 8.7 RegistrationForm.java

```
package coreservlets;

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

/** Servlet that displays an HTML form to collect user's
 * first name, last name, and email address. Uses cookies
 * to determine the initial values of each of those
 * form fields.
 */
```

**Listing 8.7** RegistrationForm.java (continued)

```

public class RegistrationForm extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String actionURL =
            "/servlet/coreservlets.RegistrationServlet";
        String firstName =
            CookieUtilities.getCookieValue(request, "firstName", "");
        String lastName =
            CookieUtilities.getCookieValue(request, "lastName", "");
        String emailAddress =
            CookieUtilities.getCookieValue(request, "emailAddress",
                "");

        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN">\n";
        String title = "Please Register";
        out.println
            (docType +
            "<HTML>\n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
            "<BODY BGCOLOR=\"#FDF5E6\"\>\n" +
            "<CENTER>\n" +
            "<H1>" + title + "</H1>\n" +
            "<FORM ACTION=\"" + actionURL + "\">\n" +
            "First Name:\n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"firstName\" " +
            "VALUE=\"" + firstName + "\"><BR>\n" +
            "Last Name:\n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"lastName\" " +
            "VALUE=\"" + lastName + "\"><BR>\n" +
            "Email Address: \n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"emailAddress\" " +
            "VALUE=\"" + emailAddress + "\"><P>\n" +
            "<INPUT TYPE=\"SUBMIT\" VALUE=\"Register\">\n" +
            "</FORM></CENTER></BODY></HTML>");
    }
}

```

**Listing 8.8** RegistrationServlet.java

```
package coreservlets;

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

/** Servlet that processes a registration form containing
 * a user's first name, last name, and email address.
 * If all the values are present, the servlet displays the
 * values. If any of the values are missing, the input
 * form is redisplayed. Either way, the values are put
 * into cookies so that the input form can use the
 * previous values.
 */

public class RegistrationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        boolean isMissingValue = false;
        String firstName = request.getParameter("firstName");
        if (isMissing(firstName)) {
            firstName = "Missing first name";
            isMissingValue = true;
        }
        String lastName = request.getParameter("lastName");
        if (isMissing(lastName)) {
            lastName = "Missing last name";
            isMissingValue = true;
        }
        String emailAddress = request.getParameter("emailAddress");
        if (isMissing(emailAddress)) {
            emailAddress = "Missing email address";
            isMissingValue = true;
        }
        Cookie c1 = new LongLivedCookie("firstName", firstName);
        response.addCookie(c1);
        Cookie c2 = new LongLivedCookie("lastName", lastName);
        response.addCookie(c2);
        Cookie c3 = new LongLivedCookie("emailAddress",
            emailAddress);
        response.addCookie(c3);
        String formAddress =
            "/servlet/coreservlets.RegistrationForm";
        if (isMissingValue) {
            response.sendRedirect(formAddress);
        }
    }
}
```

**Listing 8.8** RegistrationServlet.java (*continued*)

```
    } else {
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN\">\n";
        String title = "Thanks for Registering";
        out.println
            (docType +
             "<HTML>\n" +
             "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
             "<BODY BGCOLOR=\"#FDF5E6\">\n" +
             "<CENTER>\n" +
             "<H1 ALIGN>" + title + "</H1>\n" +
             "<UL>\n" +
             "  <LI><B>First Name</B>: " +
             "    firstName + "\n" +
             "  <LI><B>Last Name</B>: " +
             "    lastName + "\n" +
             "  <LI><B>Email address</B>: " +
             "    emailAddress + "\n" +
             "</UL>\n" +
             "</CENTER></BODY></HTML>");
    }
}

/** Determines if value is null or empty. */
private boolean isMissing(String param) {
    return((param == null) ||
           (param.trim().equals("")));
}
}
```

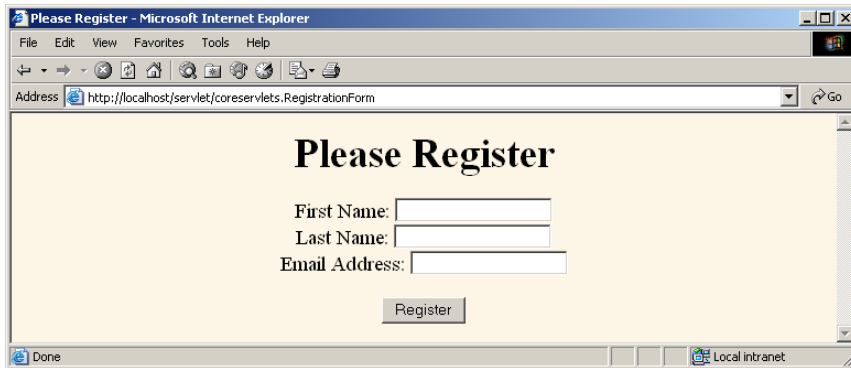


Figure 8–11 Initial result of RegistrationForm servlet.

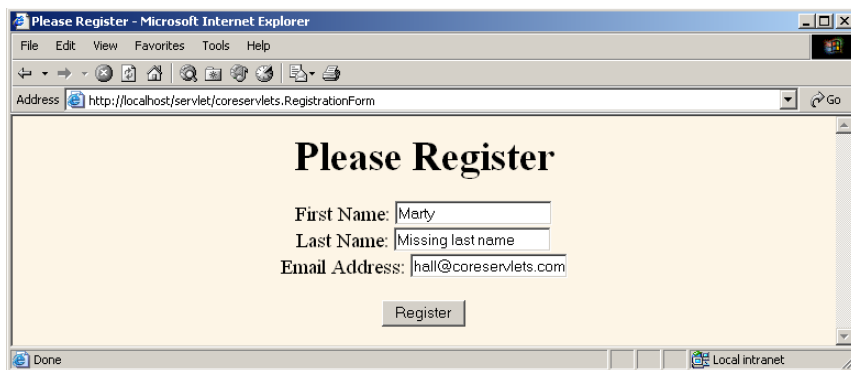
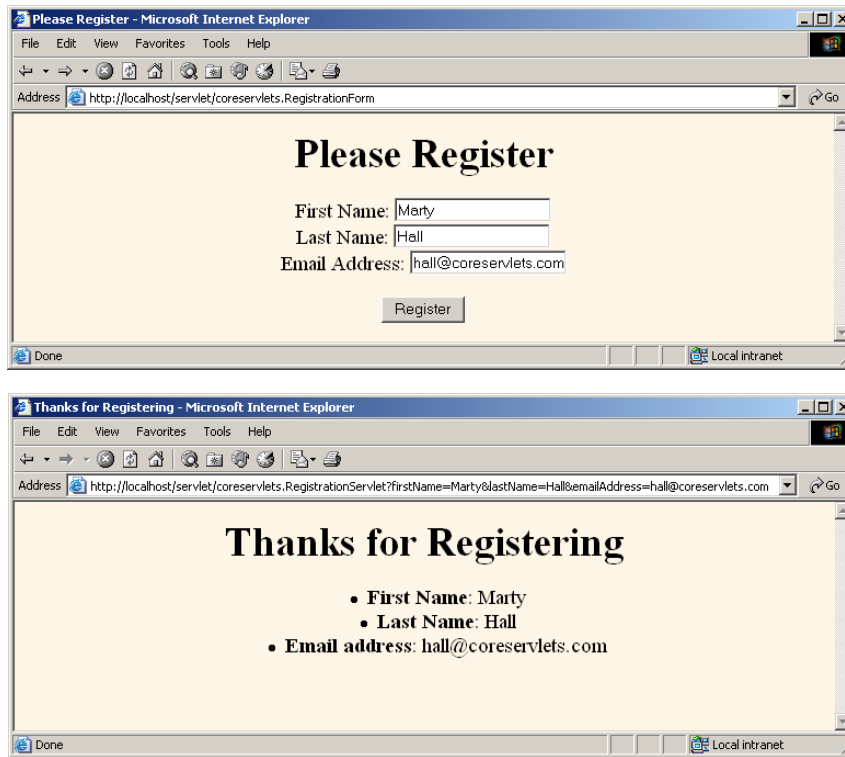


Figure 8–12 When the input form is incompletely filled in (top), the RegistrationServlet redirects the user to the RegistrationForm (bottom). The RegistrationForm uses cookies to determine the values of the form fields that were already filled in.






**Figure 8-13** When the input form is completely filled in (top), the `RegistrationServlet` (bottom) simply displays the request parameter values. The input form shown here (top) is also representative of how the form will look when the user revisits the input form at some later date: form is prepopulated with the most recently used values.



---

# HANDLING COOKIES

## Topics in This Chapter

- 
- Understanding the benefits and drawbacks of cookies
  - Sending outgoing cookies
  - Receiving incoming cookies
  - Tracking repeat visitors
  - Specifying cookie attributes
  - Differentiating between session cookies and persistent cookies
  - Simplifying cookie usage with utility classes
  - Modifying cookie values
  - Remembering user preferences

### **Training courses from the book's author:**

<http://courses.coreservlets.com/>

- *Personally* developed and taught by Marty Hall
- Available onsite at *your* organization (any country)
- Topics and pace can be customized for your developers
- Also available periodically at public venues
- Topics include Java programming, beginning/intermediate servlets and JSP, advanced servlets and JSP, Struts, JSF/MyFaces, Ajax, GWT, Ruby/Rails and more. Ask for custom courses!

---

# Chapter

# 8

## Training courses from the book's author:

<http://courses.coreservlets.com/>

- *Personally* developed and taught by Marty Hall
- Available onsite at *your* organization (any country)
- Topics and pace can be customized for your developers
- Also available periodically at public venues
- Topics include Java programming, beginning/intermediate servlets and JSP, advanced servlets and JSP, Struts, JSF/MyFaces, Ajax, GWT, Ruby/Rails and more. Ask for custom courses!

Cookies are small bits of textual information that a Web server sends to a browser and that the browser later returns unchanged when visiting the same Web site or domain. By letting the server read information it sent the client previously, the site can provide visitors with a number of conveniences such as presenting the site the way the visitor previously customized it or letting identifiable visitors in without their having to reenter a password.

This chapter discusses how to explicitly set and read cookies from within servlets, and the next chapter shows how to use the servlet session tracking API (which can use cookies behind the scenes) to keep track of users as they move around to different pages within your site.

## 8.1 Benefits of Cookies

There are four typical ways in which cookies can add value to your site. We summarize these benefits below, then give details in the rest of the section.

- **Identifying a user during an e-commerce session.** This type of short-term tracking is so important that another API is layered on top of cookies for this purpose. See the next chapter for details.

- **Remembering usernames and passwords.** Cookies let a user log in to a site automatically, providing a significant convenience for users of unshared computers.
- **Customizing sites.** Sites can use cookies to remember user preferences.
- **Focusing advertising.** Cookies let the site remember which topics interest certain users and show advertisements relevant to those interests.

## Identifying a User During an E-commerce Session

Many online stores use a “shopping cart” metaphor in which users select items, add them to their shopping carts, then continue shopping. Since the HTTP connection is usually closed after each page is sent, when a user selects a new item to add to the cart, how does the store know that it is the same user who put the previous item in the cart? Persistent (keep-alive) HTTP connections do *not* solve this problem, since persistent connections generally apply only to requests made very close together in time, as when a browser asks for the images associated with a Web page. Besides, many older servers and browsers lack support for persistent connections. Cookies, however, *can* solve this problem. In fact, this capability is so useful that servlets have an API specifically for session tracking, and servlet and JSP authors don’t need to manipulate cookies directly to take advantage of it. Session tracking is discussed in Chapter 9.

## Remembering Usernames and Passwords

Many large sites require you to register to use their services, but it is inconvenient to remember and enter the username and password each time you visit. Cookies are a good alternative for low-security sites. When a user registers, a cookie containing a unique user ID is sent to him. When the client reconnects at a later date, the user ID is returned automatically, the server looks it up, determines it belongs to a registered user that chose autologin, and permits access without an explicit username and password. The site might also store the user’s address, credit card number, and so forth in a database and use the user ID from the cookie as the key to retrieve the data. This approach prevents the user from having to reenter the data each time.

For example, when Marty travels to companies to give onsite JSP and servlet training courses, he typically checks both [travelocity.com](http://travelocity.com) and [expedia.com](http://expedia.com) for flight information. These both require usernames and passwords to search flight schedules, but have different rules about which characters are legal in usernames and how many characters are required for passwords. So, Marty has a difficult time remembering

how to log in. Fortunately, both sites use the cookie scheme described in the preceding paragraph, simplifying Marty's access from his personal desktop or laptop machine.

## Customizing Sites

Many "portal" sites let you customize the look of the main page. They might let you pick which weather report you want to see (yes, it is still raining in Seattle), what stock symbols should be displayed (yes, your stock is still way down), what sports results you care about (yes, the Orioles are still losing), how search results should be displayed (yes, you want to see more than one result per page), and so forth. Since it would be inconvenient for you to have to set up your page each time you visit their site, they use cookies to remember what you wanted. For simple settings, the site could accomplish this customization by storing the page settings directly in the cookies. For more complex customization, however, the site just sends the client a unique identifier and keeps a server-side database that associates identifiers with page settings.

## Focusing Advertising

Most advertiser-funded Web sites charge their advertisers much more for displaying "directed" (or "focused") ads than for displaying "random" ads. Advertisers are generally willing to pay much more to have their ads shown to people that are known to have some interest in the general product category. Sites reportedly charge advertisers as much as 30 times more for directed ads than for random ads. For example, if you go to a search engine and do a search on "Java Servlets," the search site can charge an advertiser much more for showing you an ad for a servlet development environment than for an ad for an online travel agent specializing in Indonesia. On the other hand, if the search had been for "Java Hotels," the situation would be reversed.

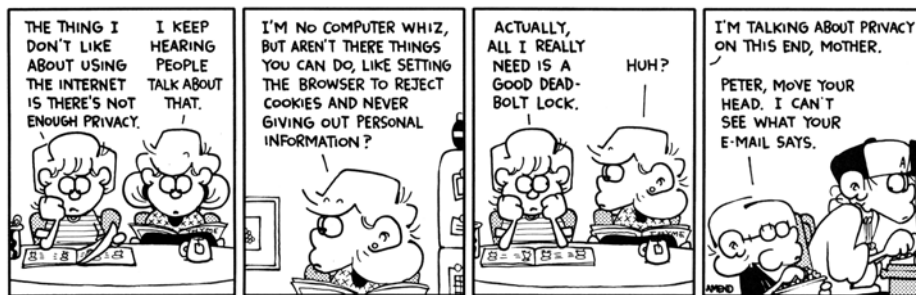
Without cookies, sites have to show a random ad when you first arrive and haven't yet performed a search, as well as when you search on something that doesn't match any ad categories. With cookies, they can identify your interests by remembering your previous searches. Since this approach enables them to show directed ads on visits to their home page as well as for their results page, it nearly doubles their advertising revenue.

## 8.2 Some Problems with Cookies

Providing convenience to the user and added value to the site owner is the purpose behind cookies. And despite much misinformation, cookies are not a serious security threat. Cookies are *never* interpreted or executed in any way and thus cannot be used

to insert viruses or attack your system. Furthermore, since browsers generally only accept 20 cookies per site and 300 cookies total, and since browsers can limit each cookie to 4 kilobytes, cookies cannot be used to fill up someone's disk or launch other denial-of-service attacks.

However, even though cookies don't present a serious *security* threat, they can present a significant threat to *privacy*.



FOXTROT © 1998 Bill Amend. Reprinted with permission of UNIVERSAL PRESS SYNDICATE. All rights reserved.

First, some people don't like the fact that search engines can remember what they previously searched for. For example, they might search for job openings or sensitive health data and don't want some banner ad tipping off their coworkers or boss next time they do a search. Besides, a search engine need not use a banner ad: a poorly designed one could display a textarea listing your most recent queries ("Jobs anywhere except at this stupid company!"; "Will my SARS infection kill my coworkers?"; etc.). A coworker could see this information if they visited the search engine for your computer or if they looked over your shoulder when you visited it.

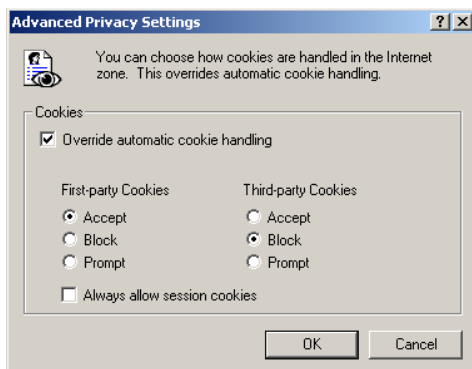
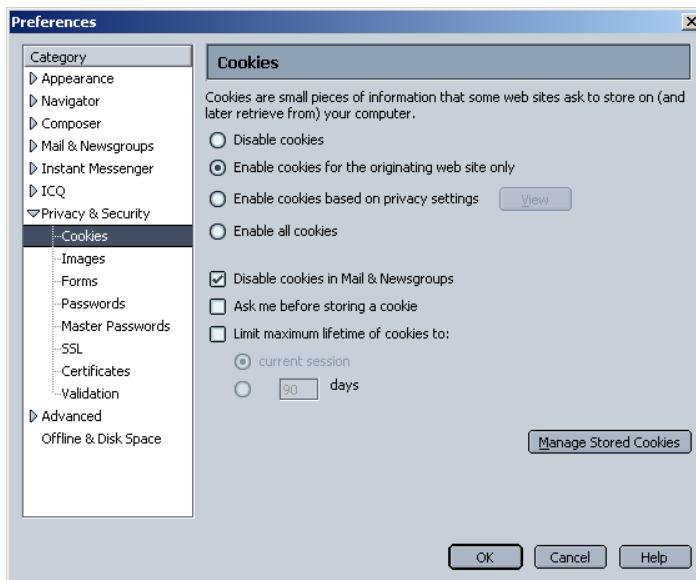
Even worse, two sites can share data on a user by each loading small images off the same third-party site, where that third party uses cookies and shares the data with both original sites. For example, suppose that both `some-search-site.com` and `some-random-site.com` wanted to display directed ads from `some-ad-site.com` based on what the user searched for at `some-search-site.com`. If the user searched for "Java Servlets," the search engine at `some-search-site.com` could return a page with the following image link:

```
<IMG SRC="http://some-ad-site.com/banner?data=Java+Servlets" ...>
```

Since the browser will make an HTTP connection to `some-ad-site.com`, `some-ad-site.com` can return a persistent cookie to the browser. Next, `some-random-site.com` could return an image link like this:

```
<IMG SRC="http://some-ad-site.com/banner" ...>
```

Since the browser will reconnect to `some-ad-site.com`—a site from which it got cookies earlier—it will return the cookie it previously received. Assuming that `some-ad-site.com` sent a unique cookie value and, in its database, associated that cookie value with the “Java Servlets” search, `some-ad-site` can return a directed banner ad even though it is the user’s first visit to `some-random-site`. The `doubleclick.net` service was the most famous early example of this technique. (Recent versions of Netscape and Internet Explorer, however, have a nice feature that lets you refuse cookies from sites other than that to which you connected, but without disabling cookies altogether. See Figure 8–1.)



**Figure 8–1** Cookie customization settings for Netscape (top) and Internet Explorer (bottom).



This trick of associating cookies with images can even be exploited through email if you use an HTML-enabled email reader that “supports” cookies and is associated with a browser. Thus, people could send you email that loads images, attach cookies to those images, and then identify you (email address and all) if you subsequently visit their Web site. Boo.

A second privacy problem occurs when sites rely on cookies for overly sensitive data. For example, some of the big online bookstores use cookies to remember your registration information and let you order without reentering much of your personal information. This is not a particular problem since they don’t actually display your complete credit card number and only let you send books to an address that was specified when you *did* enter the credit card in full or use the username and password. As a result, someone using your computer (or stealing your cookie file) could do no more harm than sending a big book order to your address, where the order could be refused. However, other companies might not be so careful, and an attacker who gained access to someone’s computer or cookie file could get online access to valuable personal information. Even worse, incompetent sites might embed credit card or other sensitive information directly in the cookies themselves, rather than using innocuous identifiers that are linked to real users only on the server. This embedding is dangerous, since most users don’t view leaving their computer unattended in their office as being tantamount to leaving their credit card sitting on their desk.

The point of this discussion is twofold:

1. Due to real and perceived privacy problems, some users turn off cookies. So, even when you use cookies to give added value to a site, whenever possible your site shouldn’t *depend* on them. Dependence on cookies is difficult to avoid in some situations, but if you can provide reasonable functionality for users without cookies enabled, so much the better.
2. As the author of servlets or JSP pages that use cookies, you should be careful not to use cookies for particularly sensitive information, since this would open users up to risks if somebody accessed the user’s computer or cookie files.

### 8.3 Deleting Cookies

You will probably find it easier to experiment with the examples in this chapter if you periodically delete your cookies (or at least the cookies that are associated with `localhost` or whatever host your server is running on).

To delete your cookies in Internet Explorer, start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies. To selectively delete cookies, press Settings, then View Files (cookie files have names that begin with Cookie:, but it is easier to find them if you choose Delete Files before View Files). See Figure 8–2.

To delete your cookies in Netscape, start at the Edit menu, then choose Preferences, Privacy and Security, and Cookies. Press the Manage Stored Cookies button to view or delete any or all of your cookies. Again, see Figure 8–2.

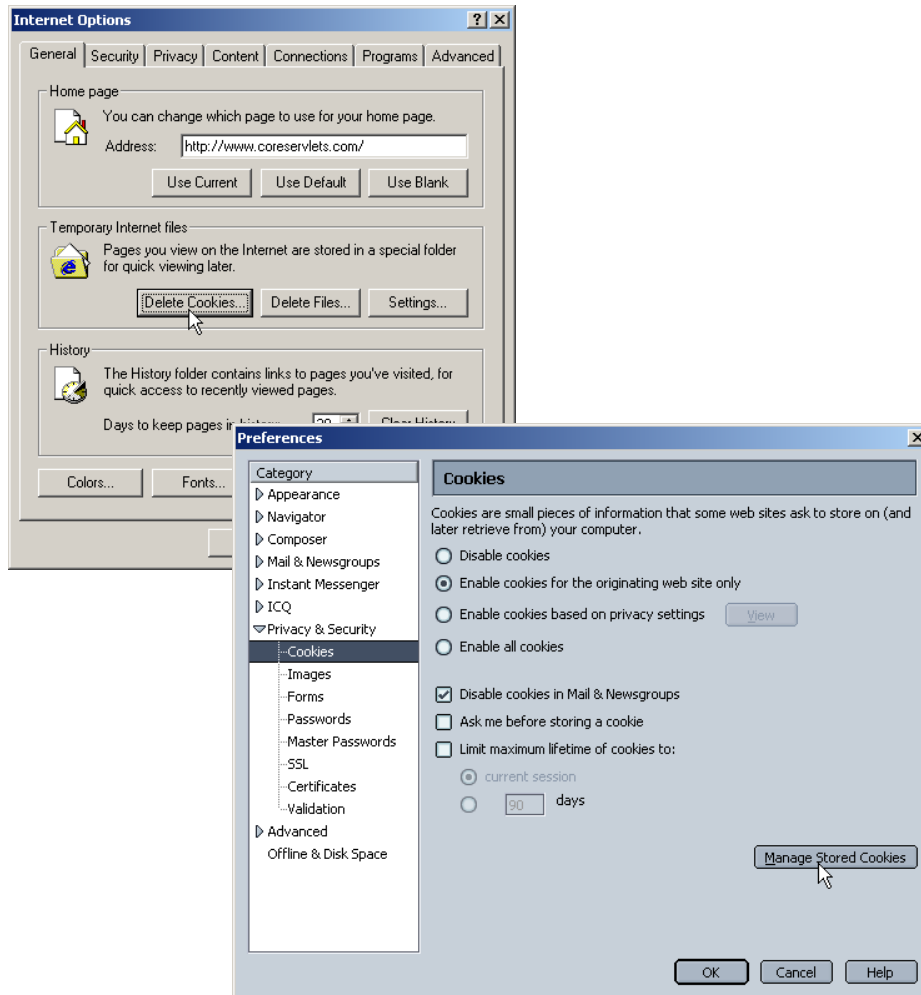


Figure 8–2 Deleting cookies in Internet Explorer and Netscape.

## 8.4 Sending and Receiving Cookies

To send cookies to the client, a servlet should use the `Cookie` constructor to create one or more cookies with designated names and values, set any optional attributes with `cookie.setXxx` (readable later by `cookie.getXxx`), and insert the cookies into the HTTP response headers with `response.addCookie`.

To read incoming cookies, a servlet should call `request.getCookies`, which returns an array of `Cookie` objects corresponding to the cookies the browser has associated with your site (`null` if there are no cookies in the request). In most cases, the servlet should then loop down this array calling `getName` on each cookie until it finds the one whose name matches the name it was searching for, then call `getValue` on that `Cookie` to see the value associated with the name. Each of these topics is discussed in more detail in the following subsections.

### Sending Cookies to the Client

Sending cookies to the client involves three steps (summarized below with details in the following subsections).

1. **Creating a `Cookie` object.** You call the `Cookie` constructor with a cookie name and a cookie value, both of which are strings.
2. **Setting the maximum age.** If you want the browser to store the cookie on disk instead of just keeping it in memory, you use `setMaxAge` to specify how long (in seconds) the cookie should be valid.
3. **Placing the `Cookie` into the HTTP response headers.** You use `response.addCookie` to accomplish this. If you forget this step, no cookie is sent to the browser!

### Creating a `Cookie` Object

You create a cookie by calling the `Cookie` constructor, which takes two strings: the cookie name and the cookie value. Neither the name nor the value should contain white space or any of the following characters:

```
[ ] ( ) = , " / ? @ : ;
```

For example, to create a cookie named `userID` with a value `a1234`, you would use the following.

```
Cookie c = new Cookie("userID", "a1234");
```

## Setting the Maximum Age

If you create a cookie and send it to the browser, by default it is a session-level cookie: a cookie that is stored in the browser's memory and deleted when the user quits the browser. If you want the browser to store the cookie on disk, use `setMaxAge` with a time in seconds, as below.

```
c.setMaxAge(60*60*24*7); // One week
```

Since you could use the session-tracking API (Chapter 9) to simplify most tasks for which you use session-level cookies, you almost always use the `setMaxAge` method when using the `Cookie` API.

Setting the maximum age to 0 instructs the browser to delete the cookie.

### Core Approach

---

*When you create a `Cookie` object, you should normally call `setMaxAge` before sending the cookie to the client.*

---



Note that `setMaxAge` is not the only `Cookie` characteristic that you can modify. The other, less frequently used characteristics are discussed in Section 8.6 (Using `Cookie` Attributes).

## Placing the Cookie in the Response Headers

By creating a `Cookie` object and calling `setMaxAge`, all you have done is manipulate a data structure in the server's memory. You haven't actually sent anything to the browser. If you don't send the cookie to the client, it has no effect. This may seem obvious, but a common mistake by beginning developers is to create and manipulate `Cookie` objects but fail to send them to the client.

### Core Warning

---

*Creating and manipulating a `Cookie` object has no effect on the client. You must explicitly send the cookie to the client with `response.addCookie`.*

---



To send the cookie, insert it into a `Set-Cookie` HTTP response header by means of the `addCookie` method of `HttpServletResponse`. The method is called `addCookie`, not `setCookie`, because any previously specified `Set-Cookie` headers

are left alone and a new header is set. Also, remember that the response headers must be set before any document content is sent to the client.

Here is an example:

```
Cookie userCookie = new Cookie("user", "uid1234");
userCookie.setMaxAge(60*60*24*365); // Store cookie for 1 year
response.addCookie(userCookie);
```

## Reading Cookies from the Client

To send a cookie *to* the client, you create a `Cookie`, set its maximum age (usually), then use `addCookie` to send a `Set-Cookie` HTTP response header. To read the cookies that come back *from* the client, you should perform the following two tasks, which are summarized below and then described in more detail in the following subsections.

1. Call `request.getCookies`. This yields an array of `Cookie` objects.
2. Loop down the array, calling `getName` on each one until you find the cookie of interest. You then typically call `getValue` and use the value in some application-specific way.

### Call `request.getCookies`

To obtain the cookies that were sent by the browser, you call `getCookies` on the `HttpServletRequest`. This call returns an array of `Cookie` objects corresponding to the values that came in on the `Cookie` HTTP request headers. If the request contains no cookies, `getCookies` should return `null`. Note, however, that an old version of Apache Tomcat (version 3.x) had a bug whereby it returned a zero-length array instead of `null`.

### Loop Down the Cookie Array

Once you have the array of cookies, you typically loop down it, calling `getName` on each `Cookie` until you find one matching the name you have in mind. Remember that cookies are specific to your host (or domain), not your servlet (or JSP page). So, although your servlet might send a single cookie, you could get many irrelevant cookies back. Once you find the cookie of interest, you typically call `getValue` on it and finish with some processing specific to the resultant value. For example:

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        Cookie cookie = cookies[i];
        if (cookieName.equals(cookie.getName())) {
```

```
        doSomethingWith(cookie.getValue());
    }
}
}
```

This is such a common process that, in Section 8.8, we present two utilities that simplify retrieving a cookie or cookie value that matches a designated cookie name.

## 8.5 Using Cookies to Detect First-Time Visitors

Suppose that, at your site, you want to display a prominent banner to first-time visitors, telling them to register. But, you don't want to clutter up the display showing a useless banner to return visitors.

A cookie is the perfect way to differentiate first-timers from repeat visitors. Check for the existence of a uniquely named cookie; if it is there, the client is a repeat visitor. If the cookie is not there, the visitor is a newcomer, and you should set an outgoing "this user has been here before" cookie.

Although this is a straightforward idea, there is one important point to note: you cannot determine if the user is a newcomer by the mere existence of entries in the cookie array. Many beginning servlet programmers erroneously use the following approach.

```
Cookie[] cookies = request.getCookies();
if (cookies == null) {
    doStuffForNewbie();           // Correct.
} else {
    doStuffForReturnVisitor(); // Incorrect.
}
```

Wrong! Sure, if the cookie array is `null`, the client is a newcomer (at least as far as you can tell—he could also have deleted or disabled cookies). But, if the array is non-`null`, it merely shows that the client has been to your *site* (or domain—see `setDomain` in the next section), not that they have been to your *servlet*. Other servlets, JSP pages, and non-Java Web applications can set cookies, and any of those cookies could get returned to your browser, depending on the path settings (see `setPath` in the next section).

Listing 8.1 illustrates the correct approach: checking for a specific cookie. Figures 8-3 and 8-4 show the results of initial and subsequent visits.

**Listing 8.1** RepeatVisitor.java

```
package coreservlets;

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

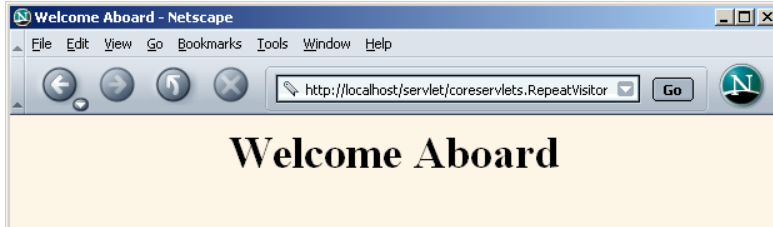
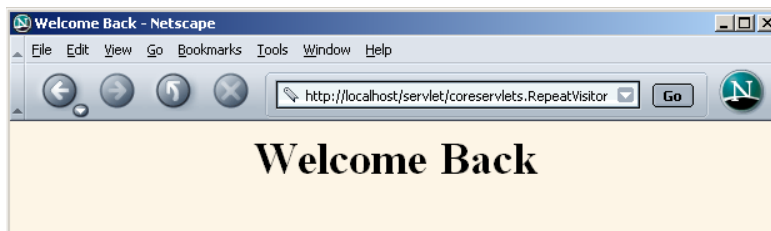
/** Servlet that says "Welcome aboard" to first-time
 * visitors and "Welcome back" to repeat visitors.
 * Also see RepeatVisitor2 for variation that uses
 * cookie utilities from later in this chapter.
 */

public class RepeatVisitor extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        boolean newbie = true;
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for(int i=0; i<cookies.length; i++) {
                Cookie c = cookies[i];
                if ((c.getName().equals("repeatVisitor")) &&
                    // Could omit test and treat cookie name as a flag
                    (c.getValue().equals("yes")) {
                    newbie = false;
                    break;
                }
            }
        }
        String title;
        if (newbie) {
            Cookie returnVisitorCookie =
                new Cookie("repeatVisitor", "yes");
            returnVisitorCookie.setMaxAge(60*60*24*365); // 1 year
            response.addCookie(returnVisitorCookie);
            title = "Welcome Aboard";
        } else {
            title = "Welcome Back";
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC \"/>";

```

**Listing 8.1** RepeatVisitor.java (*continued*)

```
out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
    "</BODY></HTML>");
}
```

**Figure 8-3** First visit by a client to the RepeatVisitor servlet.**Figure 8-4** Subsequent visits by a client to the RepeatVisitor servlet.

## 8.6 Using Cookie Attributes

Before adding the cookie to the outgoing headers, you can set various characteristics of the cookie by using the following `setXxx` methods, where `Xxx` is the name of the attribute you want to specify.

Although each `setXxx` method has a corresponding `getXxx` method to retrieve the attribute value, note that the attributes are part of the header sent from the server to the browser; they are *not* part of the header returned by the browser to the



server. Thus, except for name and value, the cookie attributes apply only to *outgoing* cookies from the server to the client; they aren't set on cookies that come *from* the browser to the server. So, don't expect these attributes to be available in the cookies you get by means of `request.getCookies`. This means that you can't implement continually changing cookie values merely by setting a maximum age on a cookie once, sending it out, finding the appropriate cookie in the incoming array on the next request, reading the value, modifying it, and storing it back in the `Cookie`. You have to call `setMaxAge` again each time (and, of course, pass the `Cookie` to `response.addCookie`).

Here are the methods that set the cookie attributes.

**public void setComment(String comment)****public String getComment()**

These methods specify or look up a comment associated with the cookie. With version 0 cookies (see the upcoming entry on `setVersion` and `getVersion`), the comment is used purely for informational purposes on the server; it is not sent to the client.

**public void setDomain(String domainPattern)****public String getDomain()**

These methods set or retrieve the domain to which the cookie applies. Normally, the browser returns cookies only to the exact same hostname that sent the cookies. For instance, cookies sent from a servlet at `bali.vacations.com` would not normally get returned by the browser to pages at `queensland.vacations.com`. If the site wanted this to happen, the servlets could specify `cookie.setDomain(".vacations.com")`. To prevent servers from setting cookies that apply to hosts outside their domain, the specified domain must meet the following requirements: it must start with a dot (e.g., `.coreservlets.com`); it must contain two dots for noncountry domains like `.com`, `.edu`, and `.gov`; and it must contain three dots for country domains like `.co.uk` and `.edu.es`.

**public void setMaxAge(int lifetime)****public int getMaxAge()**

These methods tell how much time (in seconds) should elapse before the cookie expires. A negative value, which is the default, indicates that the cookie will last only for the current browsing session (i.e., until the user quits the browser) and will not be stored on disk. See the `LongLivedCookie` class (Listing 8.4), which defines a subclass of `Cookie` with a maximum age automatically set one year in the future. Specifying a value of 0 instructs the browser to delete the cookie.

**public String getName()**

The `getName` method retrieves the name of the cookie. The name and the value are the two pieces you virtually *always* care about. However, since the name is supplied to the `Cookie` constructor, there is *no* `setName` method; you cannot change the name once the cookie is created. On the other hand, `getName` is used on almost every cookie received by the server. Since the `getCookies` method of `HttpServletRequest` returns an array of `Cookie` objects, a common practice is to loop down the array, calling `getName` until you have a particular name, then to check the value with `getValue`. For an encapsulation of this process, see the `getCookieValue` method shown in Listing 8.3.

**public void setPath(String path)  
public String getPath()**

These methods set or get the path to which the cookie applies. If you don't specify a path, the browser returns the cookie only to URLs in or below the directory containing the page that sent the cookie. For example, if the server sent the cookie from `http://ecommerce.site.com/toys/specials.html`, the browser would send the cookie back when connecting to `http://ecommerce.site.com/toys/bikes/beginners.html`, but not to `http://ecommerce.site.com/cds/classical.html`. The `setPath` method can specify something more general. For example, `cookie.setPath("/")` specifies that *all* pages on the server should receive the cookie. The path specified must include the current page; that is, you may specify a more general path than the default, but not a more specific one. So, for example, a servlet at `http://host/store/cust-service/request` could specify a path of `/store/` (since `/store/` includes `/store/cust-service/`) but not a path of `/store/cust-service/returns/` (since this directory does not include `/store/cust-service/`).

**Core Approach**

---

*To specify that a cookie apply to all URLs on your site, use `cookie.setPath("/")`.*

---

**public void setSecure(boolean secureFlag)  
public boolean getSecure()**

This pair of methods sets or gets the boolean value indicating whether the cookie should only be sent over encrypted (i.e., SSL) connections. The default is `false`; the cookie should apply to all connections.

```
public void setValue(String cookieValue)  
public String getValue()
```

The `setValue` method specifies the value associated with the cookie; `getValue` looks it up. Again, the name and the value are the two parts of a cookie that you *almost always* care about, although in a few cases, a name is used as a boolean flag and its value is ignored (i.e., the existence of a cookie with the designated name is all that matters). However, since the cookie value is supplied to the `Cookie` constructor, `setValue` is typically reserved for cases when you change the values of incoming cookies and then send them back out. For an example, see Section 8.10 (Modifying Cookie Values: Tracking User Access Counts).

```
public void setVersion(int version)  
public int getVersion()
```

These methods set and get the cookie protocol version with which the cookie complies. Version 0, the default, follows the original Netscape specification ([http://wp.netscape.com/newsref/std/cookie\\_spec.html](http://wp.netscape.com/newsref/std/cookie_spec.html)). Version 1, not yet widely supported, adheres to RFC 2109 (retrieve RFCs from the archive sites listed at <http://www.rfc-editor.org/>).

## 8.7 Differentiating Session Cookies from Persistent Cookies

This section illustrates the use of the cookie attributes by contrasting the behavior of cookies with and without a maximum age. Listing 8.2 shows the `CookieTest` servlet, a servlet that performs two tasks:

1. First, the servlet sets six outgoing cookies. Three have no explicit age (i.e., have a negative value by default), meaning that they should apply only in the current browsing session—until the user restarts the browser. The other three use `setMaxAge` to stipulate that the browser should write them to disk and that they should persist for the next hour, regardless of whether the user restarts the browser or reboots the computer to initiate a new browsing session.
2. Second, the servlet uses `request.getCookies` to find all the incoming cookies and display their names and values in an HTML table.

Figure 8–5 shows the result of the initial visit, Figure 8–6 shows a visit immediately after that, and Figure 8–7 shows the result of a visit after the user restarts the browser.

**Listing 8.2** CookieTest.java

```
package coreservlets;

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

/** Creates a table of the cookies associated with
 * the current page. Also sets six cookies: three
 * that apply only to the current session
 * (regardless of how long that session lasts)
 * and three that persist for an hour (regardless
 * of whether the browser is restarted).
 */

public class CookieTest extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        for(int i=0; i<3; i++) {
            // Default maxAge is -1, indicating cookie
            // applies only to current browsing session.
            Cookie cookie = new Cookie("Session-Cookie-" + i,
                "Cookie-Value-S" + i);
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-" + i,
                "Cookie-Value-P" + i);
            // Cookie is valid for an hour, regardless of whether
            // user quits browser, reboots computer, or whatever.
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN">\n";
        String title = "Active Cookies";
```

## Listing 8.2 CookieTest.java (continued)

```

out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
    "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
    "<TR BGCOLOR=\"#FFAD00\">\n" +
    "  <TH>Cookie Name\n" +
    "  <TH>Cookie Value");
Cookie[] cookies = request.getCookies();
if (cookies == null) {
    out.println("<TR><TH COLSPAN=2>No cookies");
} else {
    Cookie cookie;
    for(int i=0; i<cookies.length; i++) {
        cookie = cookies[i];
        out.println("<TR>\n" +
            "  <TD>" + cookie.getName() + "\n" +
            "  <TD>" + cookie.getValue());
    }
}
out.println("</TABLE></BODY></HTML>");
}
}

```



**Figure 8-5** Result of initial visit to the `CookieTest` servlet. This is the same result as when visiting the servlet, quitting the browser, waiting an hour, and revisiting the servlet.



**Figure 8-6** Result of revisiting the `CookieTest` servlet within an hour of the original visit, in the same browser session (browser stayed open between the original visit and the visit shown here).



**Figure 8-7** Result of revisiting the `CookieTest` servlet within an hour of the original visit, in a different browser session (browser was restarted between the original visit and the visit shown here).

## 8.8 Basic Cookie Utilities

This section presents some simple but useful utilities for dealing with cookies.

### Finding Cookies with Specified Names

Listing 8.3 shows two static methods in the `CookieUtilities` class that simplify the retrieval of a cookie or cookie value, given a cookie name. The `getCookieValue` method loops through the array of available `Cookie` objects, returning the value of any `Cookie` whose name matches the input. If there is no match, the designated default value is returned. So, for example, our typical approach for dealing with cookies is as follows:

```
String color =
    CookieUtilities.getCookieValue(request, "color", "black");
String font =
    CookieUtilities.getCookieValue(request, "font", "Arial");
```

The `getCookie` method also loops through the array comparing names but returns the actual `Cookie` object instead of just the value. That method is for cases when you want to do something with the `Cookie` other than just read its value. The `getCookieValue` method is more commonly used, but, for example, you might use `getCookie` in lieu of `getCookieValue` if you wanted to put a new value into the cookie and send it back out again. Just don't forget that if you use this approach, you have to respecify any cookie attributes such as date, path, and domain: these attributes are not set for incoming cookies.

#### Listing 8.3 `CookieUtilities.java`

```
package coreservlets;

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

/** Two static methods for use in cookie handling. */
public class CookieUtilities {
```

**Listing 8.3** CookieUtilities.java (continued)

```
/** Given the request object, a name, and a default value,
 * this method tries to find the value of the cookie with
 * the given name. If no cookie matches the name,
 * the default value is returned.
 */

public static String getCookieValue
    (HttpServletRequest request,
     String cookieName,
     String defaultValue) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName())) {
                return(cookie.getValue());
            }
        }
    }
    return(defaultValue);
}

/** Given the request object and a name, this method tries
 * to find and return the cookie that has the given name.
 * If no cookie matches the name, null is returned.
 */

public static Cookie getCookie(HttpServletRequest request,
    String cookieName) {
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for(int i=0; i<cookies.length; i++) {
            Cookie cookie = cookies[i];
            if (cookieName.equals(cookie.getName())) {
                return(cookie);
            }
        }
    }
    return(null);
}
}
```



## Creating Long-Lived Cookies

Listing 8.4 shows a small class that you can use instead of `Cookie` if you want your cookie to automatically persist for a year when the client quits the browser. This class (`LongLivedCookie`) merely extends `Cookie` and calls `setMaxAge` automatically.

### Listing 8.4 LongLivedCookie.java

```
package coreservlets;

import javax.servlet.http.*;

/** Cookie that persists 1 year. Default Cookie doesn't
 *  * persist past current browsing session.
 *  */

public class LongLivedCookie extends Cookie {
    public static final int SECONDS_PER_YEAR = 60*60*24*365;

    public LongLivedCookie(String name, String value) {
        super(name, value);
        setMaxAge(SECONDS_PER_YEAR);
    }
}
```

---

## 8.9 Putting the Cookie Utilities into Practice

Listing 8.5 redoes the `RepeatVisitor` servlet of Listing 8.1. The new version (`RepeatVisitor2`) has the same functionality as the old version: it says “Welcome Aboard” to first-time visitors and “Welcome Back” to repeat visitors. However, it uses the cookie utilities of Section 8.8 to simplify the code in two ways:

1. Instead of calling `request.getCookies` and looping down that array examining each name, it merely calls `CookieUtilities.getCookieValue`.
2. Instead of creating a `Cookie` object, calculating the number of seconds in a year, and then calling `setMaxAge`, it merely creates a `LongLivedCookie` object.

Figures 8–8 and 8–9 show the results.

**Listing 8.5** RepeatVisitor2.java

```
package coreservlets;

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

/** A variation of the RepeatVisitor servlet that uses
 *  CookieUtilities.getCookieValue and LongLivedCookie
 *  to simplify the code.
 */

public class RepeatVisitor2 extends HttpServlet {
    public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
        throws ServletException, IOException {
        boolean newbie = true;
        String value =
            CookieUtilities.getCookieValue(request, "repeatVisitor2",
                                         "no");
        if (value.equals("yes")) {
            newbie = false;
        }
        String title;
        if (newbie) {
            LongLivedCookie returnVisitorCookie =
                new LongLivedCookie("repeatVisitor2", "yes");
            response.addCookie(returnVisitorCookie);
            title = "Welcome Aboard";
        } else {
            title = "Welcome Back";
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN">\n";
        out.println(docType +
                   "<HTML>\n" +
                   "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                   "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                   "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                   "</BODY></HTML>");
    }
}
```

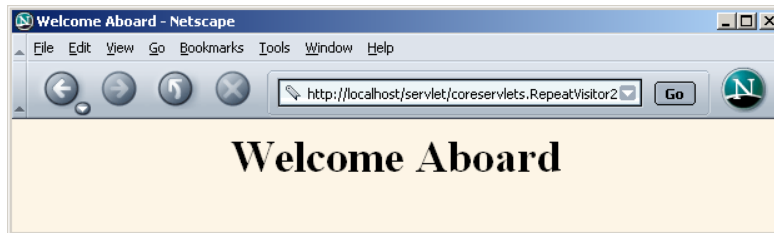


Figure 8-8 First visit by a client to the RepeatVisitor2 servlet.

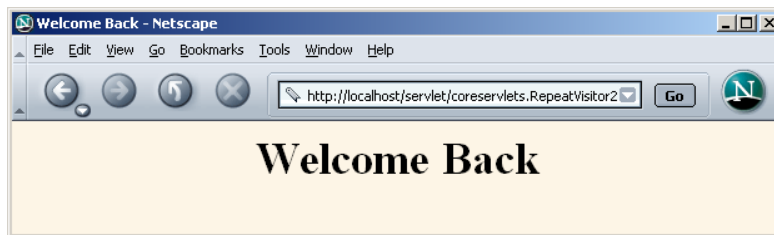


Figure 8-9 Subsequent visit by a client to the RepeatVisitor2 servlet.

## 8.10 Modifying Cookie Values: Tracking User Access Counts

In the previous examples, we sent a cookie to the user only on the first visit. Once the cookie had a value, we never changed it. This approach of a single cookie value is surprisingly common since cookies frequently contain nothing but unique user identifiers: all the real user data is stored in a database—the user identifier is merely the database key.

But what if you want to periodically change the value of a cookie? How do you do so?

- To *replace* a previous cookie value, send the same cookie name with a different cookie value. If you actually use the incoming `Cookie` objects, don't forget to do `response.addCookie`; merely calling `setValue` is not sufficient. You also need to reapply any relevant cookie attributes by calling `setMaxAge`, `setPath`, etc.—cookie attributes are not specified for incoming cookies. Reapplying these attributes means that reusing the incoming `Cookie` objects saves you little, so many developers don't bother.

- To instruct the browser to *delete* a cookie, use `setMaxAge` to assign a maximum age of 0.

Listing 8.6 presents a servlet that keeps track of how many times each client has visited the page. It does this by making a cookie whose name is `accessCount` and whose value is the actual count. To accomplish this task, the servlet needs to repeatedly replace the cookie value by resending a cookie with the identical name.

Figure 8–10 shows some typical results.

**Listing 8.6** ClientAccessCounts.java

```
package coreservlets;

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

/** Servlet that prints per-client access counts. */

public class ClientAccessCounts extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        String countString =
            CookieUtilities.getCookieValue(request,
                "accessCount",
                "1");

        int count = 1;
        try {
            count = Integer.parseInt(countString);
        } catch (NumberFormatException nfe) { }
        LongLivedCookie c =
            new LongLivedCookie("accessCount",
                String.valueOf(count+1));
        response.addCookie(c);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Access Count Servlet";
        String docType =
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"Transitional//EN\">\n";
    }
}
```

**Listing 8.6** ClientAccessCounts.java (continued)

```

out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
    "<BODY BGCOLOR=\"#FDF5E6\">\n" +
    "<CENTER>\n" +
    "<H1>" + title + "</H1>\n" +
    "<H2>This is visit number " +
    count + " by this browser.</H2>\n" +
    "</CENTER></BODY></HTML>");
    }
}

```



**Figure 8-10** Users each see their own access count. Also, Internet Explorer and Netscape maintain cookies separately, so the same user sees independent access counts with the two browsers.

## 8.11 Using Cookies to Remember User Preferences

One of the most common applications of cookies is to use them to “remember” user preferences. For simple user settings, as here, the preferences can be stored directly in the cookies. For more complex applications, the cookie typically contains a unique user identifier and the preferences are stored in a database.

Listing 8.7 presents a servlet that creates an input form with the following characteristics.

- **The form is redisplayed if it is incomplete when submitted.** The form sends data to a second servlet (Listing 8.8) that checks whether any of the designated request parameters is missing, then stores the parameter values in cookies. If no parameter is missing, the second servlet displays the parameter values. If a parameter is missing, the second servlet redirects the user to the original servlet so that the form can be redisplayed. The original servlet maintains the user’s previously entered values by extracting them from the cookies.
- **The form remembers previous entries.** The fields are prepopulated with whatever values the user entered on the most recent request.

Figures 8–11 through 8–13 show some typical results.

### Listing 8.7 RegistrationForm.java

```
package coreservlets;

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

/** Servlet that displays an HTML form to collect user's
 * first name, last name, and email address. Uses cookies
 * to determine the initial values of each of those
 * form fields.
 */
```

**Listing 8.7** RegistrationForm.java (continued)

```

public class RegistrationForm extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String actionURL =
            "/servlet/coreservlets.RegistrationServlet";
        String firstName =
            CookieUtilities.getCookieValue(request, "firstName", "");
        String lastName =
            CookieUtilities.getCookieValue(request, "lastName", "");
        String emailAddress =
            CookieUtilities.getCookieValue(request, "emailAddress",
                "");

        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN">\n";
        String title = "Please Register";
        out.println
            (docType +
            "<HTML>\n" +
            "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
            "<BODY BGCOLOR=\"#FDF5E6\"\>\n" +
            "<CENTER>\n" +
            "<H1>" + title + "</H1>\n" +
            "<FORM ACTION=\"" + actionURL + "\">\n" +
            "First Name:\n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"firstName\" " +
            "VALUE=\"" + firstName + "\"><BR>\n" +
            "Last Name:\n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"lastName\" " +
            "VALUE=\"" + lastName + "\"><BR>\n" +
            "Email Address: \n" +
            "  <INPUT TYPE=\"TEXT\" NAME=\"emailAddress\" " +
            "VALUE=\"" + emailAddress + "\"><P>\n" +
            "<INPUT TYPE=\"SUBMIT\" VALUE=\"Register\">\n" +
            "</FORM></CENTER></BODY></HTML>");
    }
}

```

**Listing 8.8** RegistrationServlet.java

```
package coreservlets;

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

/** Servlet that processes a registration form containing
 * a user's first name, last name, and email address.
 * If all the values are present, the servlet displays the
 * values. If any of the values are missing, the input
 * form is redisplayed. Either way, the values are put
 * into cookies so that the input form can use the
 * previous values.
 */

public class RegistrationServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        boolean isMissingValue = false;
        String firstName = request.getParameter("firstName");
        if (isMissing(firstName)) {
            firstName = "Missing first name";
            isMissingValue = true;
        }
        String lastName = request.getParameter("lastName");
        if (isMissing(lastName)) {
            lastName = "Missing last name";
            isMissingValue = true;
        }
        String emailAddress = request.getParameter("emailAddress");
        if (isMissing(emailAddress)) {
            emailAddress = "Missing email address";
            isMissingValue = true;
        }
        Cookie c1 = new LongLivedCookie("firstName", firstName);
        response.addCookie(c1);
        Cookie c2 = new LongLivedCookie("lastName", lastName);
        response.addCookie(c2);
        Cookie c3 = new LongLivedCookie("emailAddress",
            emailAddress);
        response.addCookie(c3);
        String formAddress =
            "/servlet/coreservlets.RegistrationForm";
        if (isMissingValue) {
            response.sendRedirect(formAddress);
        }
    }
}
```



**Listing 8.8** RegistrationServlet.java (*continued*)

```
    } else {
        PrintWriter out = response.getWriter();
        String docType =
            "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
            "Transitional//EN\">\n";
        String title = "Thanks for Registering";
        out.println
            (docType +
             "<HTML>\n" +
             "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
             "<BODY BGCOLOR=\"#FDF5E6\">\n" +
             "<CENTER>\n" +
             "<H1 ALIGN>" + title + "</H1>\n" +
             "<UL>\n" +
             "  <LI><B>First Name</B>: " +
             "    firstName + "\n" +
             "  <LI><B>Last Name</B>: " +
             "    lastName + "\n" +
             "  <LI><B>Email address</B>: " +
             "    emailAddress + "\n" +
             "</UL>\n" +
             "</CENTER></BODY></HTML>");
    }
}

/** Determines if value is null or empty. */
private boolean isMissing(String param) {
    return((param == null) ||
           (param.trim().equals("")));
}
}
```

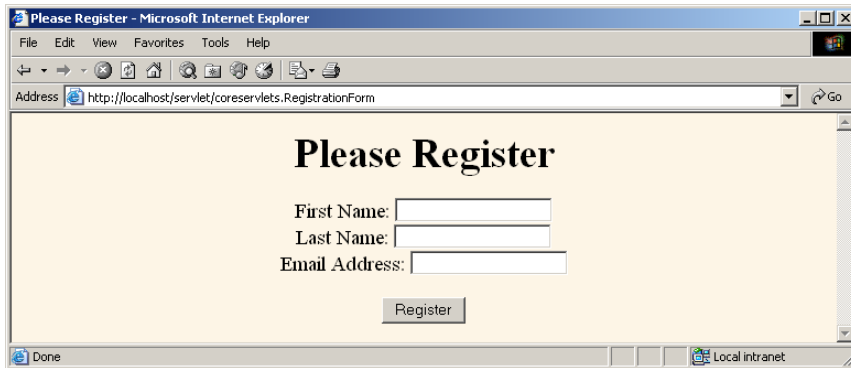
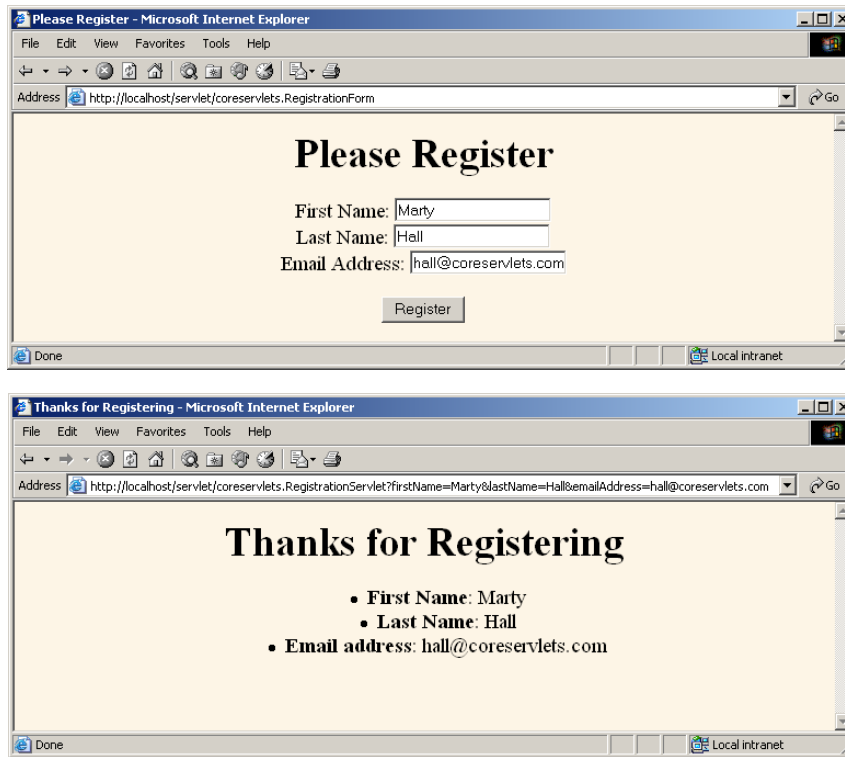


Figure 8–11 Initial result of RegistrationForm servlet.



Figure 8–12 When the input form is incompletely filled in (top), the RegistrationServlet redirects the user to the RegistrationForm (bottom). The RegistrationForm uses cookies to determine the values of the form fields that were already filled in.



**Figure 8-13** When the input form is completely filled in (top), the `RegistrationServlet` (bottom) simply displays the request parameter values. The input form shown here (top) is also representative of how the form will look when the user revisits the input form at some later date: form is prepopulated with the most recently used values.