Agenda

- Format of the HTTP response
- **Usage of** `HttpServletResponse`
- How to set status codes
- Shortcut methods for redirection and error pages
- Setting response headers
- Summary of HTTP headers
- Example: store a persistent state and asks reload
- Cookies
  - Session and persistent cookies
  - Tracking user access
  - Remember user preferences

Most of the slides were made available by www.coreservlets.com
HTTP Request/Response

- **Request**
  
  ```plaintext
  GET /servlet/SomeName HTTP/1.1
  Host: ...
  Header2: ...
  ...
  HeaderN: 
  (Blank Line)
  ```

- **Response**
  
  ```plaintext
  HTTP/1.1 200 OK
  Content-Type: text/html
  Header2: ...
  ...
  HeaderN: ...
  (Blank Line)
  <!DOCTYPE ...>
  <HTML>
  <HEAD>...
  <BODY>
  ...
  </BODY></HTML>
  ```

**Status code**
The client read it and can perform an action based on that code
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                        HttpServletResponse response)
                    throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
Setting Status Codes

- `response.setStatus(int statusCode)`
  - Use a **constant** for the code, not an explicit `int` (see specifications of `HTTPServletResponse`)
  - Names derived from standard message, e.g.: `SC_OK`, `SC_NOT_FOUND`, etc.

- `response.sendError(int code, String message)`
  - 404 (Not Found) is `SC_NOT_FOUND`
  - Wraps message inside small HTML document

- `response.sendRedirect(String url)`
  1. Sets status code to 302
  2. Sets `Location` response header and gives the URL of the new document.
Common HTTP 1.1 Status Codes

- **200 (OK)**
  - Everything is fine; document follows
  - **Default** for servlets

- **204 (No Content)**
  - Browser should keep displaying previous document

- **301 (Moved Permanently)**
  - Requested document permanently moved elsewhere *(indicated in Location header)*
  - Browsers go to new location automatically
  - Browsers are technically supposed to **follow** 301 and 302 (next page) requests only when the incoming request is GET, but do it for POST with 303.
Common HTTP 1.1 Status Codes

- **302 (Found)**
  - Requested document *temporarily* moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use `sendRedirect`, *not* `setStatus`, when setting this header (see example)

- **401 (Unauthorized)**
  - Browser tried to access password-protected page without proper Authorization header

- **404 (Not Found)**
  - No such page. Servlets should use `sendError` to set this.
Redirect Users to Browser-Specific Pages

```java
public class WrongDestination extends HttpServlet {
    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        String userAgent = request.getHeader("User-Agent");
        if ((userAgent != null) &&
            (userAgent.indexOf("MSIE") != -1)) {
            response.sendRedirect("http://home.netscape.com");
        } else {
            response.sendRedirect("http://www.microsoft.com");
        }
    }
}
```

It is TRUE if "MSIE" is found in the User-Agent header
Redirects Users to Browser-Specific Pages
Front End to Search Engines: HTML Form

One-Stop Web Search!

Search keywords: pizza
testa

- Google
- AllTheWeb
- Yahoo
- AltaVista
- Lycos
- HotBot
- MSN

Submit Query

Link to SearchEngineForm

SearchEngineForm servelet calls SearchEngines servelet
public class SearchEngines extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String searchString = request.getParameter("searchString"); // provided by the form
        if ((searchString == null) ||
            (searchString.length() == 0)) {
            reportProblem(response, "Missing search string");
            return;
        }
        searchString = URLEncoder.encode(searchString); // in java.net
        String searchEngineName =
            request.getParameter("searchEngine"); // provided by the form
        if ((searchEngineName == null) ||
            (searchEngineName.length() == 0)) {
            reportProblem(response, "Missing search engine name");
            return;
        }
    }
}
A Front End to Various Search Engines

String searchURL =
    SearchUtilities.makeURL(searchEngineName,
     searchString);  //combine the query string
if (searchURL != null) {
    //and the search engine name
    response.sendRedirect(searchURL); //set a header ...
} else {
    reportProblem(response,
        "Unrecognized search engine");
}

private void reportProblem(HttpServletResponse response,
     String message)
    throws IOException {
    response.sendError(response.SC_NOT_FOUND, message);
}
}
Result for Invalid Data

HTTP Status 404 - Missing search engine name

- **type**: Status report
- **message**: Missing search engine name
- **description**: The requested resource (Missing search engine name) is not available.

HTTP Status 404 - Unrecognized search engine

- **type**: Status report
- **message**: Unrecognized search engine
- **description**: The requested resource (Unrecognized search engine) is not available.
HTTP Request/Response

- Request
  
  GET /servlet/SomeName HTTP/1.1
  Host: ...
  Header2: ...
  ...
  HeaderN:
  (Blank Line)

- Response
  
  HTTP/1.1 200 OK
  Content-Type: text/html
  Header2: ...
  ...
  HeaderN: ...
  (Blank Line)
  <!DOCTYPE ...>
  <HTML>
  <HEAD>...</HEAD>
  <BODY>
  ...
  </BODY></HTML>
Setting Arbitrary Response Headers

- `response.setHeader(String headerName, String headerValue)`
  - Sets an arbitrary header – **specify it before returning the document**

- `response.setDateHeader(String name, long millisecs)`
  - The name of the date could be a header that contains a date, such as: `Date`, or `Last-Modified`
  - Converts milliseconds since 1970 to a date string in GMT format!!

- `response.setIntHeader(String name, int headerValue)`
  - Prevents need to convert int to String before calling `setHeader`

- `addHeader, addDateHeader, addIntHeader`
  - Adds new occurrence of header instead of replacing.
Setting Common Response Headers

- `setContentType`
  - Sets the Content-Type header
  - Servlets almost always use this
  - See table of common MIME types

- `setContentLength`
  - Sets the Content-Length header
  - Used for persistent HTTP connections

- `addCookie`
  - Adds a value to the Set-Cookie header
  - See separate section on cookies

- `sendRedirect`
  - Sets the Location header (plus changes status code).
Common HTTP 1.1 Response Headers

- **Cache-Control:** (1.1) and **Pragma:** (1.0)
  - A **no-cache** value prevents browsers from caching page
  - **public** - document can be cached
  - **private** - document is for a single user, can be stored only in private (nonshared) caches
  - **no-store** - not cache and not even store in a temporary location
  - **must-revalidate** - client must revalidate document with original server (not the proxy)
  - **proxy-revalidate** - as must-revalidate but it applies to shared caches
  - **max-age=xxx** - document should be considered stale after xxx seconds

**Ex cache-control: private**
Common HTTP 1.1 Response Headers

- **Content-Disposition**
  - Lets you request that the browser ask the user to save the response to disk in a file of the given name: `Content-Disposition: attachment; filename=file-name`

- **Content-Encoding**
  - The way document is encoded

- **Content-Length**
  - The number of bytes in the response
  - **Use** `ByteArrayOutputStream` **to buffer document** before sending it, so that you can determine size.
Common HTTP 1.1 Response Headers

- **Content-Type**
  - The MIME type of the document being returned
  - Use `setContentType` to set this header

- **Expires**
  - The time at which document should be considered out-of-date and thus should no longer be cached
  - Use `setDateHeader` to set this header (with `Expires` as header name)

- **Last-Modified**
  - The time document was last changed
  - Don’t set this header explicitly; provide a `getLastModified` method instead (see lottery number example in previous lecture).

**Example:** `Expires: and max-age`
Common HTTP 1.1 Response Headers

- **Location**
  - The URL to which browser should reconnect
  - *Use* `sendRedirect` *instead of setting this directly*

- **Refresh**
  - The number of seconds until browser should reload page:
    - `response.setIntHeader("Refresh", 5)`
  - Can also include URL to connect to:
    - `response.setHeader("Refresh", "5; URL=http://host/path")`

- **Set-Cookie**
  - The cookies that browser should remember. Don’t set this header directly; use `addCookie` instead (see next slides)

- **WWW-Authenticate**
  - The authorization type and realm needed in Authorization header.
Handling Long-Running Servlets

- A way to store data useful for different requests
  - For data that is not specific to any one client, store it in a field (instance variable) of the servlet
  - For data that is specific to a user, store it in the HttpSession object (next lecture on session tracking)
  - For data that needs to be available to other servlets or JSP pages (regardless of user), store it in theServletContext

- A way to keep computations running after the response is sent to the user
  - Start a Thread - set the thread priority to a low value so that you do not slow down the server

- A way to get the updated results to the browser when they are ready
  - Use Refresh header to tell browser to ask for updates.
Persistent State and Auto-Reloading Pages

- Idea: generate list of large (e.g., 150-digit) prime numbers
  - Show partial results until completed
  - Let new clients make use of results previously computed for other requests
- Demonstrates use of the Refresh header
- Shows how easy it is for servlets to maintain state between requests
  - Very difficult in traditional CGI
- Also illustrates that servlets can handle multiple simultaneous connections
  - Each request is in a separate thread.
Finding Large Prime Numbers

Number of primes to calculate: [25]
Number of digits: [15]
Start Calculating

form
Finding Prime Numbers

Some 85-Digit Prime Numbers

Primes found with 85 or more digits: 11.

Still looking for 14 more...

1. 277430128019416709090185630136235751760830502088340
2. 277430128019416709090185630136235751760830502088340
3. 277430128019416709090185630136235751760830502088340
4. 277430128019416709090185630136235751760830502088340
5. 277430128019416709090185630136235751760830502088340
6. 277430128019416709090185630136235751760830502088340
7. 277430128019416709090185630136235751760830502088340
8. 277430128019416709090185630136235751760830502088340
9. 277430128019416709090185630136235751760830502088340
10. 277430128019416709090185630136235751760830502088340
11. 277430128019416709090185630136235751760830502088340

Done searching.

Some 85-Digit Prime Numbers

Primes found with 85 or more digits: 25.

Done searching.
public class PrimeNumberServlet extends HttpServlet {
    private ArrayList primeListCollection = new ArrayList();
    private int maxPrimeLists = 30;

    public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
                      throws ServletException, IOException {
        int numPrimes =
            ServletUtilities.getIntParameter(request,
                                               "numPrimes", 50);

        int numDigits =
            ServletUtilities.getIntParameter(request,
                                               "numDigits", 120);

        PrimeList primeList =
            findPrimeList(primeListCollection, // Checks if has already
                          numPrimes, numDigits); // computed these primes
    }

PrimeNumberServlet uses ServletUtilities
if (primeList == null) {
    // generates the primes in a separate thread
    primeList = new PrimeList(numPrimes, numDigits, true);
    // Multiple servlet request threads share the instance
    // variables (fields) of PrimeNumbers. So
    // synchronize all access to servlet fields.
    synchronized(primeListCollection) {
        if (primeListCollection.size() >= maxPrimeLists)
            primeListCollection.remove(0);
        primeListCollection.add(primeList); // when this list is added
    } // is probably not completed
}

ArrayList currentPrimes = primeList.getPrimes(); // those found
int numCurrentPrimes = currentPrimes.size();
int numPrimesRemaining = (numPrimes - numCurrentPrimes);
boolean isLastResult = (numPrimesRemaining == 0);
if (!isLastResult) {
    response.setIntHeader("Refresh", 5);
}
Finding Prime Numbers (III)

response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Some " + numDigits + "-Digit Prime Numbers";
out.println(ServletUtilities.headWithTitle(title) +
   "<BODY BGCOLOR="#FDF5E6">\n" +
   "<H2 ALIGN=CENTER>" + title + "</H2>\n" +
   "<H3>Primes found with " + numDigits +
   " or more digits: " + numCurrentPrimes +
   ".</H3>" );
if (isLastResult)
   out.println("<B>Done searching.</B>" );
else
   out.println("<B>Still looking for " + numPrimesRemaining +
   " more<BLINK>...</BLINK></B>");
out.println("<OL>");
for(int i=0; i<numCurrentPrimes; i++) {
   out.println("  <LI>" + currentPrimes.get(i));
}
out.println("</OL>");
out.println("</BODY></HTML>" );
The Potential of Cookies

- **Idea**
  - Servlet sends a simple name and value to client
  - Client returns same name and value when it connects to same site (or same domain, depending on cookie settings)

- **Typical Uses of Cookies**
  - Identifying a user during an e-commerce session
    - Servlets have a higher-level API for this task
  - **Avoiding username and password**
  - Customizing a site
  - Focusing advertising.
Sending Cookies to the Client

- **Create a Cookie object**
  - Call the Cookie constructor with a cookie name and a cookie value, both of which are strings.
    
    ```java
    Cookie c = new Cookie("userID", "a1234");
    ```

- **Set the maximum age**
  - To tell browser to store cookie on disk instead of just in memory, use setMaxAge (argument is in seconds)
    
    ```java
    c.setMaxAge(60*60*24*7); // One week
    ```

- **Place the Cookie into the HTTP response**
  - Use `response.addCookie`
  - If you forget this step, no cookie is sent to the browser!
    
    ```java
    response.addCookie(c);
    ```
# Reading Cookies from the Client

- **Call** `request.getCookies`
  - This yields an array of `Cookie` objects
- **Loop down the array, calling `getName` on each entry until you find the cookie of interest**
  - Use the value (`getValue`) in application-specific way.

```java
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(int i=0; i<cookies.length; i++) {
        if (cookieName.equals(cookies[i].getName())) {
            doSomethingWith(cookies[i].getValue());
        }
    }
}
```
Using Cookies to Detect First-Time Visitors

Welcome Aboard

Welcome Back

Call servlet
Using Cookies to Detect First-Time Visitors

```java
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")) &&
                    (c.getValue().equals("yes"))) {
                    newbie = false;
                    break;
                }
            }
        }
    }
}
```
Using Cookies to Detect First-Time Visitors

String title;
if (newbie) { // the first time the cookie is created
    Cookie returnVisitorCookie =
        new Cookie("repeatVisitor", "yes");
    returnVisitorCookie.setMaxAge(60*60*24*365);
    response.addCookie(returnVisitorCookie);
    title = "Welcome Aboard";
} else {
    title = "Welcome Back";
}
response.setContentType("text/html");
PrintWriter out = response.getWriter();
... // (Output page with above title)
Using Cookie Attributes

- **getDomain/setDomain**
  - Lets you specify domain to which cookie applies - current host must be part of domain specified

- **getMaxAge/setMaxAge**
  - Gets/sets the cookie expiration time (in seconds)
  - If you fail to set this, cookie applies to current browsing session only

- **getName**
  - Gets the cookie name - there is no `setName` method; you supply name to constructor
  - For incoming cookie array, you use `getName` to find the cookie of interest.
Using Cookie Attributes

- **getPath/setPath**
  - Gets/sets the path to which cookie applies - if unspecified, cookie applies to URLs that are within or below directory containing current page

- **getSecure/setSecure**
  - Gets/sets flag indicating whether cookie should apply only to SSL connections or to all connections

- **getValue/setValue**
  - Gets/sets value associated with cookie
  - For new cookies, you supply value to constructor, not to setValue
  - For incoming cookie array, you use getName to find the cookie of interest, then call setValue on the result
  - If you set the value of an incoming cookie, you still have to send it back out with response.addCookie.
Example

[Image of a computer screen showing a Cookies window with a search for localhost, and a table listing cookies with the following details:

<table>
<thead>
<tr>
<th>Site</th>
<th>Cookie Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>localhost</td>
<td>repeatVisitor</td>
</tr>
<tr>
<td>localhost</td>
<td>emailAddress</td>
</tr>
<tr>
<td>localhost</td>
<td>lastName</td>
</tr>
<tr>
<td>localhost</td>
<td>firstName</td>
</tr>
</tbody>
</table>

Name: repeatVisitor
Content: yes
Host: localhost
Path: /coresjsp/
Send For: Any type of connection
Expires: April 11, 2012 9:36:15 AM

Remove Cookie  Remove All Cookies]
public class CookieTest extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        for(int i=0; i<3; i++) {
            Cookie cookie =
                    new Cookie("Session-Cookie-"+ i,
                            "Cookie-Value-S" + i);
            // No maxAge (ie maxAge = -1)
            response.addCookie(cookie);
            cookie = new Cookie("Persistent-Cookie-"+ i,
                                "Cookie-Value-P" + i);
            cookie.setMaxAge(3600);
            response.addCookie(cookie);
        }
    }
Session Cookies and Persistent Cookies (Cont)

... // Start an HTML table
Cookie[] cookies = request.getCookies();
if (cookies == null) {
    out.println("<TR><TH COLSPAN=2>No cookies";
} else {
    for(Cookie cookie: cookies) {
        out.println
        ("<TR><TD>" +
         " <TD>" + cookie.getName() + "\n" +
         " <TD>" + cookie.getValue());
    }
}
out.println("</TABLE></BODY></HTML>"};
}
Session Cookies from Persistent Cookies

- Result of initial visit to `CookieTest` servlet
  - Same result as when visiting the servlet, quitting the browser, waiting an hour, and revisiting the servlet.
Session Cookies and Persistent Cookies

- Result of revisiting **CookieTest** within an hour of original visit (same browser session)
  - I.e., browser stayed open between the original visit and the visit shown here

![Active Cookies - Mozilla Firefox](image)

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Cookie Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session-Cookie-0</td>
<td>Cookie-Value-S0</td>
</tr>
<tr>
<td>Persistent-Cookie-0</td>
<td>Cookie-Value-P0</td>
</tr>
<tr>
<td>Session-Cookie-1</td>
<td>Cookie-Value-S1</td>
</tr>
<tr>
<td>Persistent-Cookie-1</td>
<td>Cookie-Value-P1</td>
</tr>
<tr>
<td>Session-Cookie-2</td>
<td>Cookie-Value-S2</td>
</tr>
<tr>
<td>Persistent-Cookie-2</td>
<td>Cookie-Value-P2</td>
</tr>
</tbody>
</table>
Session Cookies from Persistent Cookies

- Result of revisiting `CookieTest` within an hour of original visit (different browser session)
  - I.e., browser was restarted between the original visit and the visit shown here.

  ![Active Cookies - Mozilla Firefox](image)

  **Active Cookies**

<table>
<thead>
<tr>
<th>Cookie Name</th>
<th>Cookie Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent-Cookie-2</td>
<td>Cookie-Value-P2</td>
</tr>
<tr>
<td>Persistent-Cookie-0</td>
<td>Cookie-Value-P0</td>
</tr>
<tr>
<td>Persistent-Cookie-1</td>
<td>Cookie-Value-P1</td>
</tr>
</tbody>
</table>

  Depending on browser settings, the persistent cookies could be deleted by the browser on exit. So you may also find no cookie at all!

  In Firefox if you “exit and save” you also keep the session cookies for the next session.
public class CookieUtilities {
    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++) {
                if (cookieName.equals(cookies[i].getName())) {
                    return(cookies[i].getValue());
                }
            }
        }
        return(defaultValue);
    }
    ...
}
Utility: Creating Long-Lived Cookies

```java
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);
    }
}
```
Applying Utilities: RepeatVisitor2

```java
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";
        }
    }
}
```

Servlet call
Modifying Cookie Values

- Replacing a cookie value
  - Send the same cookie name with a different cookie value
  - Reusing incoming Cookie objects
    - Need to call `response.addCookie`; merely calling `setValue` is not sufficient
    - Also need to reapply any relevant cookie attributes by calling `setMaxAge`, `setPath`, etc.—cookie attributes are not specified for incoming cookies
    - Usually not worth the bother, so new Cookie object used

- Instructing the browser to delete a cookie
  - Use `setMaxAge` to assign a maximum age of 0.
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);
Tracking User Access Counts (Continued)

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

Access Count Servlet

This is visit number 6 by this browser.

Access Count Servlet

This is visit number 15 by this browser.

Call servlet
Using Cookies to Remember User Preferences

1. RegistrationForm servlet
   - Uses cookie values to prepopulate form field values
   - Uses default values if no cookies are found
   - Will be redone in JSP later

2. Registration servlet
   - Creates cookies based on request parameters received
   - Displays values if all parameters are present
   - Redirects to form if any parameter is missing.
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 = "RegistrationServlet";
        String firstName = CookieUtilities.getCookieValue(request, "firstName", "");
        String lastName = CookieUtilities.getCookieValue(request, "lastName", "");
        String emailAddress = CookieUtilities.getCookieValue(request, "emailAddress", "");
    }
}

If nothing is found it returns the empty string
RegistrationForm Servlet (Continued)

```java
out.println
(docType +
"<HTML>
" +
"<HEAD><TITLE>" + title + "</TITLE></HEAD>
" +
"<BODY BGCOLOR="#FDF5E6">\n" +
"<CENTER>\n" +
"<H1>" + title + "</H1>\n" +
"<FORM ACTION="" + actionURL + "">\n" +
"First Name:\n" +
"  <INPUT TYPE="TEXT" NAME="firstName" " +
"  VALUE="" + firstName + "">\n" +
"Last Name:\n" +
"  <INPUT TYPE="TEXT" NAME="lastName" " +
"  VALUE="" + lastName + "">\n" +
"Email Address: \n" +
"  <INPUT TYPE="TEXT" NAME="emailAddress" " +
"  VALUE="" + emailAddress + "">\n" +
"<INPUT TYPE="SUBMIT" VALUE="Register">\n" +
"</FORM></CENTER></BODY></HTML>";
}
```
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;
        }
        ...
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 =
    "RegistrationForm";
if (isMissingValue) {
    response.sendRedirect(formAddress);
} else { ... } // print back to the user the
    // registration data.
RegistrationForm (Initial Result)

Please Register

First Name: 
Last Name: 
Email Address: 

Register

Call servlet
RegistrationForm (Submitting Incomplete Data)
RegistrationForm (Submitting Complete Form)

Please Register

First Name: francesco
Last Name: ricci
Email Address: fricci@unibz.it

Register

Thanks for Registering

- First Name: francesco
- Last Name: ricci
- Email address: fricci@unibz.it
Registration Form (Initial Result on Later Visit)

Please Register

First Name: francesco
Last Name: ricci
Email Address: fricci@unibz.it

Register