Handling the Client Request: Form Data

Topics in This Chapter

- Using `getParameter` to read single values from prespecified parameters in the form data
- Using `getParameterValues` to read multiple values from prespecified parameters in the form data
- Using `getParameterNames` to discover what parameters are available
- Handling both `GET` and `POST` requests with a single servlet
- A servlet that makes a table of the input parameters
- An on-line resumé posting service
- Filtering HTML-specific characters

Chapter 3

One of the main motivations for building Web pages dynamically is so that the result can be based upon user input. This chapter shows you how to access that input.

3.1 The Role of Form Data

If you've ever used a search engine, visited an on-line bookstore, tracked stocks on the Web, or asked a Web-based site for quotes on plane tickets, you've probably seen funny-looking URLs like `http://host/path?user=Marty+Hall&origin=bwi&dest=lax`. The part after the question mark (i.e., `user=Marty+Hall&origin=bwi&dest=lax`) is known as *form data* (or *query data*) and is the most common way to get information from a Web page to a server-side program. Form data can be attached to the end of the URL after a question mark (as above), for `GET` requests, or sent to the server on a separate line, for `POST` requests. If you're not familiar with HTML forms, Chapter 16 (Using HTML Forms) gives details on how to build forms that collect and transmit data of this sort.

Extracting the needed information from this form data is traditionally one of the most tedious parts of CGI programming. First of all, you have to read the data one way for `GET` requests (in traditional CGI, this is usually through the `QUERY_STRING` environment variable) and a different way for `POST` requests (by reading the standard input in traditional CGI). Second, you have to chop the pairs at the ampersands, then separate the parameter names (left of the equal signs) from the parameter values (right of the equal signs). Third, you have to URL-decode the values. Alphanumeric characters are sent unchanged, but spaces are converted to plus signs and other characters are converted to *%XX* where *XX* is the ASCII (or ISO Latin-1) value of the character, in hex. Then, the server-side program has to reverse the process. For example, if someone enters a value of "`~hall, ~gates, and ~mcnealy`" into a textfield with the name `users` in an HTML form, the data is sent as "`users=%7Ehall%2C+%7Egates%2C+and+%7Emcnealy`", and the server-side program has to reconstitute the original string. Finally, the fourth reason that parsing form data is tedious is that values can be omitted (e.g., "`param1=val1&`**`param2=`**`&param3=val3`") or a parameter can have more than one value (e.g., "**`param1=val1`**`&param2=val2&`**`param1=val3`**"), so your parsing code needs special cases for these situations.

3.2 Reading Form Data from Servlets

One of the nice features of servlets is that all of this form parsing is handled automatically. You simply call the `getParameter` method of the `HttpServletRequest`, supplying the case-sensitive parameter name as an argument. You use `getParameter` exactly the same way when the data is sent by `GET` as you do when it is sent by `POST`. The servlet knows which request method was used and automatically does the right thing behind the scenes. The return value is a `String` corresponding to the URL-decoded value of the first occurrence of that parameter name. An empty `String` is returned if the parameter exists but has no value, and `null` is returned if there was no such parameter. If the parameter could potentially have more than one value, you should call `getParameterValues` (which returns an array of strings) instead of `getParameter` (which returns a single string). The return value of `getParameterValues` is `null` for nonexistent parameter names and is a one-element array when the parameter has only a single value.

Parameter names are case sensitive so, for example, `request.getParameter("Param1")` and `request.getParameter("param1")` are *not* interchangeable.

Core Warning

The values supplied to `getParameter` *and* `getParameterValues` *are case sensitive.*

Finally, although most real servlets look for a specific set of parameter names, for debugging purposes it is sometimes useful to get a full list. Use `getParameterNames` to get this list in the form of an `Enumeration`, each entry of which can be cast to a `String` and used in a `getParameter` or `getParameterValues` call. Just note that the `HttpServletRequest` API does not specify the order in which the names appear within that `Enumeration`.

Core Warning

Don't count on `getParameterNames` *returning the names in any particular order.*

3.3 Example: Reading Three Explicit Parameters

Listing 3.1 presents a simple servlet called `ThreeParams` that reads form data parameters named `param1`, `param2`, and `param3` and places their values in a bulleted list. Listing 3.2 shows an HTML form that collects user input and sends it to this servlet. By use of an `ACTION` of `/servlet/coreservlets.ThreeParams`, the form can be installed anywhere on the system running the servlet; there need not be any particular association between the directory containing the form and the servlet installation directory. Recall that the specific locations for installing HTML files vary from server to server. With the JSWDK 1.0.1 and Tomcat 3.0, HTML pages are placed somewhere in *`install_dir`*`/webpages` and are accessed via `http://host/path/file.html`. For example, if the form shown in Listing 3.2 is placed in *`install_dir`*`/webpages/forms/ThreeParamsForm.html` and the server is accessed from the same host that it is running on, the form would be accessed by a URL of `http://localhost/forms/ThreeParamsForm.html`.

Figures 3–1 and 3–2 show the result of the HTML front end and the servlet, respectively.

Listing 3.1 `ThreeParams.java`

```
package coreservlets;

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

public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>param2</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

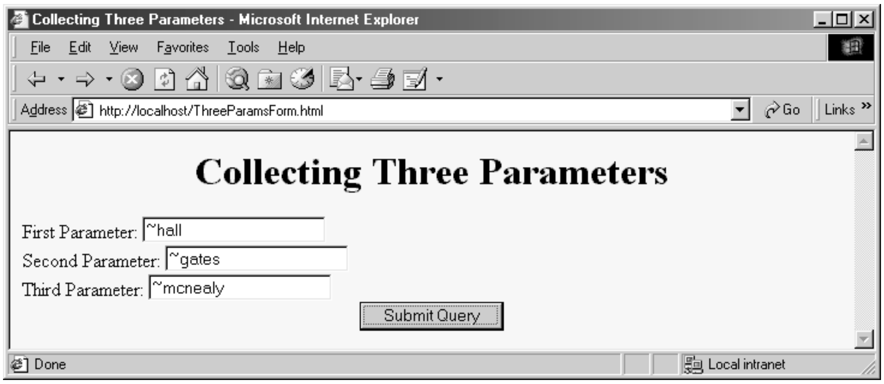
Although you are required to specify *response* settings (see Chapters 6 and 7) before beginning to generate the content, there is no requirement that you read the *request* parameters at any particular time.

If you're accustomed to the traditional CGI approach where you read `POST` data through the standard input, you should note that you can do the same thing with servlets by calling `getReader` or `getInputStream` on the `HttpServletRequest` and then using that stream to obtain the raw input. This is a bad idea for regular parameters since the input is neither parsed (separated into entries specific to each parameter) nor URL-decoded (translated so that plus signs become spaces and *%XX* gets replaced by the

ASCII or ISO Latin-1 character corresponding to the hex value *XX*). However, reading the raw input might be of use for uploaded files or `POST` data being sent by custom clients rather than by HTML forms. Note, however, that if you read the `POST` data in this manner, it might no longer be found by `getParameter`.

Listing 3.2 `ThreeParamsForm.html`

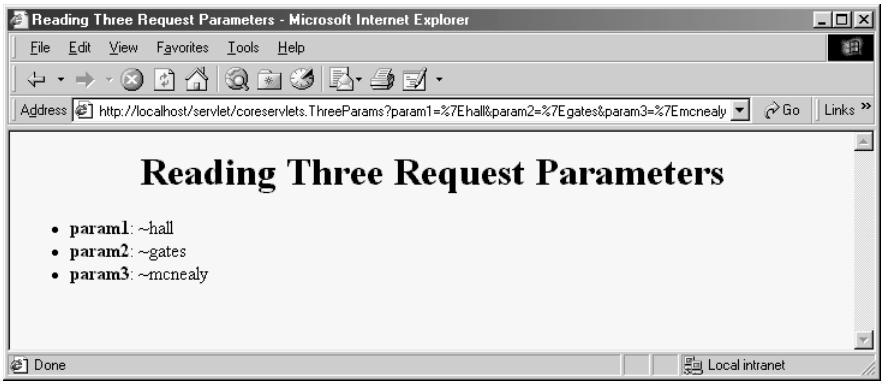
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Collecting Three Parameters</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">Collecting Three Parameters</H1>

<FORM ACTION="/servlet/coreservlets.ThreeParams">
  First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
  Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
  Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY>
</HTML>
```


Collecting Three Parameters - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ThreeParamsForm.html
Go
Links
Done
Local intranet

Collecting Three Parameters

First Parameter: ~hall
Second Parameter: ~gates
Third Parameter: ~mcnealy
Submit Query

Figure 3–1 HTML front end resulting from `ThreeParamsForm.html`.


Reading Three Request Parameters - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.ThreeParams?param1=%7Ehall&param2=%7Egates&param3=%7Emcnealy
Go
Links
Done
Local intranet

Reading Three Request Parameters

- **param1**: ~hall
- **param2**: ~gates
- **param3**: ~mcnealy

Figure 3–2 Output of `ThreeParams` servlet.

3.4 Example: Reading All Parameters

The previous example extracted parameter values from the form data based upon prespecified parameter names. It also assumed that each parameter had exactly one value. Here's an example that looks up *all* the parameter names that are sent and puts their values in a table. It highlights parameters that have missing values as well as ones that have multiple values.

First, the servlet looks up all the parameter names by the `getParameterNames` method of `HttpServletRequest`. This method returns an `Enumeration` that contains the parameter names in an unspecified order. Next, the servlet loops down the `Enumeration` in the standard manner, using `hasMoreElements` to determine when to stop and using `nextElement` to get each entry. Since `nextElement` returns an `Object`, the servlet casts the result to a `String` and passes that to `getParameterValues`, yielding an array of strings. If that array is one entry long and contains only an empty string, then the parameter had no values and the servlet generates an italicized "No Value" entry. If the array is more than one entry long, then the parameter had multiple values and the values are displayed in a bulleted list. Otherwise, the one main value is placed into the table unmodified. The source code for the servlet is shown in Listing 3.3, while Listing 3.4 shows the HTML code for a front end that can be used to try the servlet out. Figures 3–3 and 3–4 show the result of the HTML front end and the servlet, respectively.

Notice that the servlet uses a `doPost` method that simply calls `doGet`. That's because I want it to be able to handle *both* `GET` and `POST` requests. This approach is a good standard practice if you want HTML interfaces to have some flexibility in how they send data to the servlet. See the discussion of the `service` method in Section 2.6 (The Servlet Life Cycle) for a discussion of why having `doPost` call `doGet` (or vice versa) is preferable to overriding `service` directly. The HTML form from Listing 3.4 uses `POST`, as should *all* forms that have password fields (if you don't know why, see Chapter 16). However, the `ShowParameters` servlet is not specific to that particular front end, so the source code archive site at `www.coreservlets.com` includes a similar HTML form that uses `GET` for you to experiment with.

Listing 3.3 `ShowParameters.java`

```
package coreservlets;

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

public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Reading All Request Parameters";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>Parameter Name<TH>Parameter Value(s)");
    Enumeration paramNames = request.getParameterNames();
    while(paramNames.hasMoreElements()) {
      String paramName = (String)paramNames.nextElement();
      out.print("<TR><TD>" + paramName + "\n<TD>");
      String[] paramValues =
        request.getParameterValues(paramName);
      if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0)
          out.println("<I>No Value</I>");
```

Listing 3.3 `ShowParameters.java` (continued)

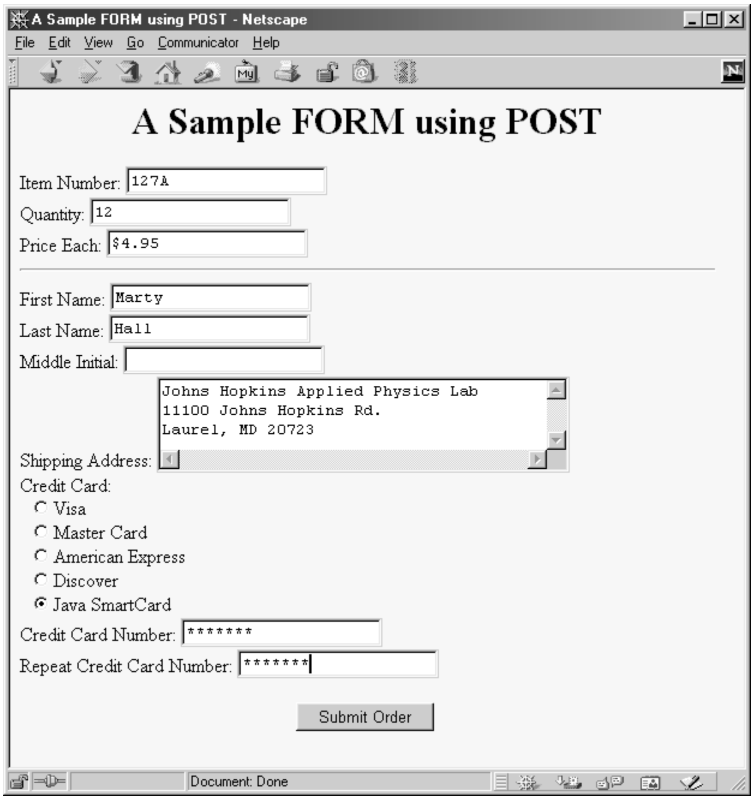
```
        else
          out.println(paramValue);
      } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) {
          out.println("<LI>" + paramValues[i]);
        }
        out.println("</UL>");
      }
    }
    out.println("</TABLE>\n</BODY></HTML>");
  }

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

Listing 3.4 `ShowParametersPostForm.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>

<FORM ACTION="/servlet/coreservlets.ShowParameters"
      METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  <HR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  Last Name: <INPUT TYPE="TEXT" NAME="lastName"><BR>
  Middle Initial: <INPUT TYPE="TEXT" NAME="initial"><BR>
  Shipping Address:
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card:<BR>
  &nbsp;&nbsp;<INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Visa">Visa<BR>
  &nbsp;&nbsp;<INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Master Card">Master Card<BR>
```

Listing 3.4 `ShowParametersPostForm.html` (continued)

```
  &nbsp;&nbsp;<INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Amex">American Express<BR>
  &nbsp;&nbsp;<INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Discover">Discover<BR>
  &nbsp;&nbsp;<INPUT TYPE="RADIO" NAME="cardType"
                     VALUE="Java SmartCard">Java SmartCard<BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  Repeat Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>

</BODY>
</HTML>
```

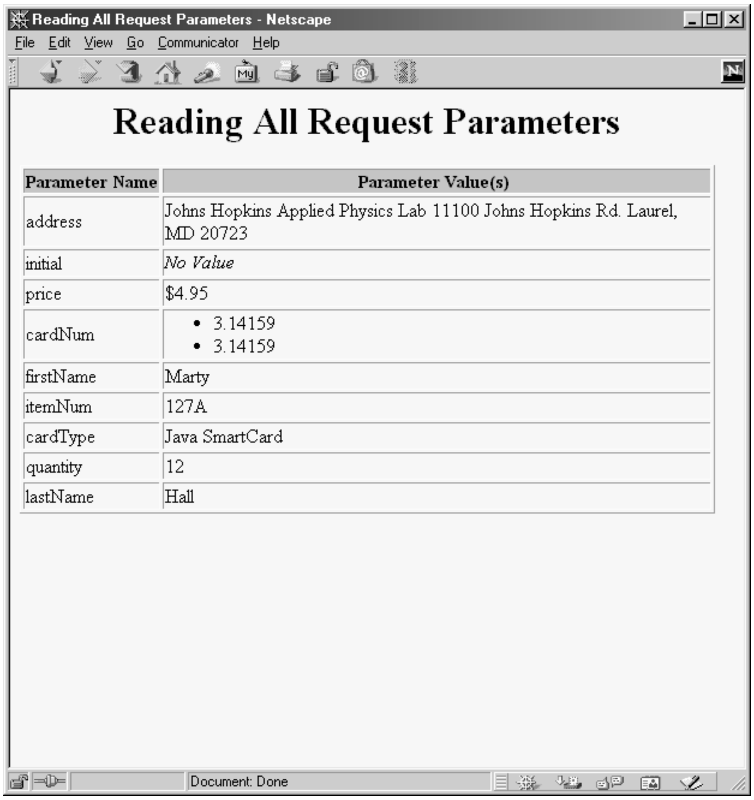

A Sample FORM using POST - Netscape
File Edit View Go Communicator Help
Submit Order
Document: Done

A Sample FORM using POST

Item Number: 127A
Quantity: 12
Price Each: $4.95

First Name: Marty
Last Name: Hall
Middle Initial:
Shipping Address: Johns Hopkins Applied Physics Lab
11100 Johns Hopkins Rd.
Laurel, MD 20723
Credit Card:
Visa
Master Card
American Express
Discover
[✓ Java SmartCard]
Credit Card Number: *******
Repeat Credit Card Number: *******

Figure 3–3 HTML front end that collects data for `ShowParameters` servlet.


Reading All Request Parameters - Netscape
File Edit View Go Communicator Help
Document: Done

Reading All Request Parameters

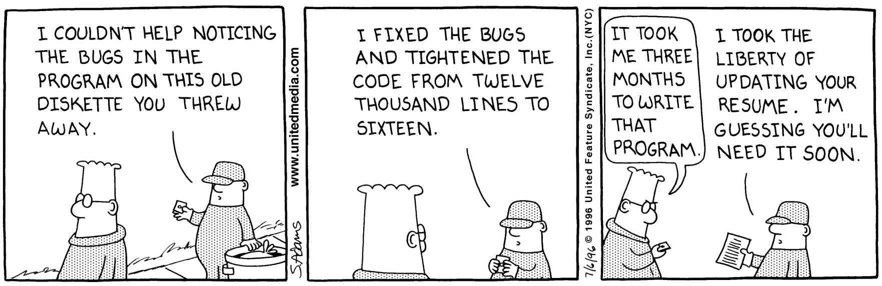
Parameter Name	Parameter Value(s)
address	Johns Hopkins Applied Physics Lab 11100 Johns Hopkins Rd. Laurel, MD 20723
initial	*No Value*
price	$4.95
cardNum	• 3.14159 • 3.14159
firstName	Marty
itemNum	127A
cardType	Java SmartCard
quantity	12
lastName	Hall

Figure 3–4 Output of `ShowParameters` servlet.

3.5 A Resumé Posting Service

On-line job services have become increasingly popular of late. A reputable site provides a useful service to job seekers by giving their skills wide exposure and provides a useful service to employers by giving them access to a large pool of prospective employees. This section presents a servlet that handles part of such a site: the submission of on-line resumés.

Listing 3.5 and Figure 3–5 show the HTML form that acts as the front end to the resumé-processing servlet. If you are not familiar with HTML forms, they are covered in detail in Chapter 16. The important thing to understand here is that the form uses `POST` to submit the data and that it gathers values for the following parameter names:


I COULDN'T HELP NOTICING THE BUGS IN THE PROGRAM ON THIS OLD DISKETTE YOU THREW AWAY.
I FIXED THE BUGS AND TIGHTENED THE CODE FROM TWELVE THOUSAND LINES TO SIXTEEN.
IT TOOK ME THREE MONTHS TO WRITE THAT PROGRAM.
I TOOK THE LIBERTY OF UPDATING YOUR RESUME. I'M GUESSING YOU'LL NEED IT SOON.
www.unitedmedia.com
7/6/96 © 1996 United Feature Syndicate, Inc.(NYC)

DILBERT reprinted by permission of United Syndicate, Inc.

- **`headingFont`**
  Headings will be displayed in this font. A value of "default" results in a sans-serif font such as Arial or Helvetica.
- **`headingSize`**
  The person's name will be displayed in this point size. Subheadings will be displayed in a slightly smaller size.
- **`bodyFont`**
  The main text (languages and skills) will be displayed in this font.
- **`bodySize`**
  The main text will be displayed in this point size.
- **`fgColor`**
  Text will be this color.
- **`bgColor`**
  The page background will be this color.
- **`name`**
  This parameter specifies the person's name. It will be centered at the top of the resumé in the font and point size previously specified.
- **`title`**
  This parameter specifies the person's job title. It will be centered under the name in a slightly smaller point size.
- **`email`**
  The job applicant's email address will be centered under the job title inside a `mailto` link.
- **`languages`**
  The programming languages listed will be placed in a bulleted list in the on-line resumé.
- **`skills`**
  Text from the skills text area will be displayed in the body font at the bottom of the resumé under a heading called "Skills and Experience."

Listing 3.6 shows the servlet that processes the data from the HTML form. When the "Preview" button is pressed, the servlet first reads the font and color parameters. Before using any of the parameters, it checks to see if the value is `null` (i.e., there is an error in the HTML form and thus the parameter is missing) or is an empty string (i.e., the user erased the default value but did not enter anything in its place). The servlet uses a default value appropriate to each parameter in such a case. Parameters that represent numeric values are passed to `Integer.parseInt`. To guard against the possibility of improperly formatted numbers supplied by the user, this `Integer.parseInt` call is placed inside a `try/catch` block that supplies a default value when the parsing fails. Although it may seem a bit tedious to handle these cases, it generally is not too much work if you make use of some utility methods such as `replaceIfMissing` and `replaceIfMissingOrDefault` in Listing 3.6. Tedious or not, users will sometimes overlook certain fields or misunderstand the required field format, so it is critical that your servlet handle malformed parameters gracefully and that you test it with both properly formatted and improperly formatted data.

Core Approach

Design your servlets to gracefully handle missing or improperly formatted parameters. Test them with malformed data as well as with data in the expected format.

Once the servlet has meaningful values for each of the font and color parameters, it builds a cascading style sheet out of them. If you are unfamiliar with style sheets, they are a standard way of specifying the font faces, font sizes, colors, indentation, and other formatting information in an HTML 4.0 Web page. Style sheets are usually placed in a separate file so that several Web pages at a site can share the same style sheet, but in this case it is more convenient to embed the style information directly in the page by using the `STYLE` element. For more information on style sheets, see `http://www.w3.org/TR/REC-CSS1`.

After creating the style sheet, the servlet places the job applicant's name, job title, and e-mail address centered under each other at the top of the page. The heading font is used for these lines, and the e-mail address is placed inside a `mailto:` hypertext link so that prospective employers can contact the applicant directly by clicking on the address. The programming languages specified in the `languages` parameter are parsed using `StringTokenizer` (assuming spaces and/or commas are used to separate the language names) and placed in a bulleted list beneath a "Programming Languages" heading.

Finally, the text from the `skills` parameter is placed at the bottom of the page beneath a "Skills and Experience" heading.

Figures 3–6 through 3–8 show a couple of possible results. Listing 3.7 shows the underlying HTML of the first of these results.

Listing 3.5 `SubmitResume.html`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>Free Resume Posting</TITLE>
  <LINK REL=STYLESHEET
        HREF="jobs-site-styles.css"
        TYPE="text/css">
</HEAD>
<BODY>
<H1>hotcomputerjobs.com</H1>
<P CLASS="LARGER">
To use our <I>free</I> resume-posting service, simply fill
out the brief summary of your skills below. Use "Preview"
to check the results, then press "Submit" once it is
ready. Your mini resume will appear on-line within 24 hours.</P>
<HR>
<FORM ACTION="/servlet/coreservlets.SubmitResume"
      METHOD="POST">
<DL>
<DT><B>First, give some general information about the look of
your resume:</B>
<DD>Heading font:
    <INPUT TYPE="TEXT" NAME="headingFont" VALUE="default">
<DD>Heading text size:
    <INPUT TYPE="TEXT" NAME="headingSize" VALUE=32>
<DD>Body font:
    <INPUT TYPE="TEXT" NAME="bodyFont" VALUE="default">
<DD>Body text size:
    <INPUT TYPE="TEXT" NAME="bodySize" VALUE=18>
<DD>Foreground color:
    <INPUT TYPE="TEXT" NAME="fgColor" VALUE="BLACK">
<DD>Background color:
    <INPUT TYPE="TEXT" NAME="bgColor" VALUE="WHITE">

<DT><B>Next, give some general information about yourself:</B>
<DD>Name: <INPUT TYPE="TEXT" NAME="name">
<DD>Current or most recent title:
    <INPUT TYPE="TEXT" NAME="title">
<DD>Email address: <INPUT TYPE="TEXT" NAME="email">
<DD>Programming Languages:
    <INPUT TYPE="TEXT" NAME="languages">
```

Listing 3.5 `SubmitResume.html` **(continued)**

```
<DT><B>Finally, enter a brief summary of your skills and
    experience:</B> (use &lt;P&gt; to separate paragraphs.
    Other HTML markup is also permitted.)
<DD><TEXTAREA NAME="skills"
              ROWS=15 COLS=60 WRAP="SOFT"></TEXTAREA>
</DL>
  <CENTER>
    <INPUT TYPE="SUBMIT" NAME="previewButton" Value="Preview">
    <INPUT TYPE="SUBMIT" NAME="submitButton" Value="Submit">
  </CENTER>
</FORM>
<HR>
<P CLASS="TINY">See our privacy policy
<A HREF="we-will-spam-you.html">here</A>.</P>
</BODY>
</HTML>
```

Listing 3.6 `SubmitResume.java`

```
package coreservlets;

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

/** Servlet that handles previewing and storing resumes
 *  submitted by job applicants.
 */

public class SubmitResume extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    if (request.getParameter("previewButton") != null) {
      showPreview(request, out);
    } else {
      storeResume(request);
      showConfirmation(request, out);
    }
  }
```

Listing 3.6 SubmitResume.java (continued)

```
// Shows a preview of the submitted resume. Takes
// the font information and builds an HTML
// style sheet out of it, then takes the real
// resume information and presents it formatted with
// that style sheet.

private void showPreview(HttpServletRequest request,
                         PrintWriter out) {
  String headingFont = request.getParameter("headingFont");
  headingFont = replaceIfMissingOrDefault(headingFont, "");
  int headingSize =
    getSize(request.getParameter("headingSize"), 32);
  String bodyFont = request.getParameter("bodyFont");
  bodyFont = replaceIfMissingOrDefault(bodyFont, "");
  int bodySize =
    getSize(request.getParameter("bodySize"), 18);
  String fgColor = request.getParameter("fgColor");
  fgColor = replaceIfMissing(fgColor, "BLACK");
  String bgColor = request.getParameter("bgColor");
  bgColor = replaceIfMissing(bgColor, "WHITE");
  String name = request.getParameter("name");
  name = replaceIfMissing(name, "Lou Zer");
  String title = request.getParameter("title");
  title = replaceIfMissing(title, "Loser");
  String email = request.getParameter("email");
  email =
    replaceIfMissing(email, "contact@hotcomputerjobs.com");
  String languages = request.getParameter("languages");
  languages = replaceIfMissing(languages, "<I>None</I>");
  String languageList = makeList(languages);
  String skills = request.getParameter("skills");
  skills = replaceIfMissing(skills, "Not many, obviously.");
  out.println
    (ServletUtilities.DOCTYPE + "\n" +
     "<HTML>\n" +
     "<HEAD>\n" +
     "<TITLE>Resume for " + name + "</TITLE>\n" +
     makeStyleSheet(headingFont, headingSize,
                    bodyFont, bodySize,
                    fgColor, bgColor) + "\n" +
     "</HEAD>\n" +
     "<BODY>\n" +
     "<CENTER>\n"+
     "<SPAN CLASS=\"HEADING1\">" + name + "</SPAN><BR>\n" +
     "<SPAN CLASS=\"HEADING2\">" + title + "<BR>\n" +
     "<A HREF=\"mailto:" + email + "\">" + email +
         "</A></SPAN>\n" +
```

Listing 3.6 `SubmitResume.java` **(continued)**

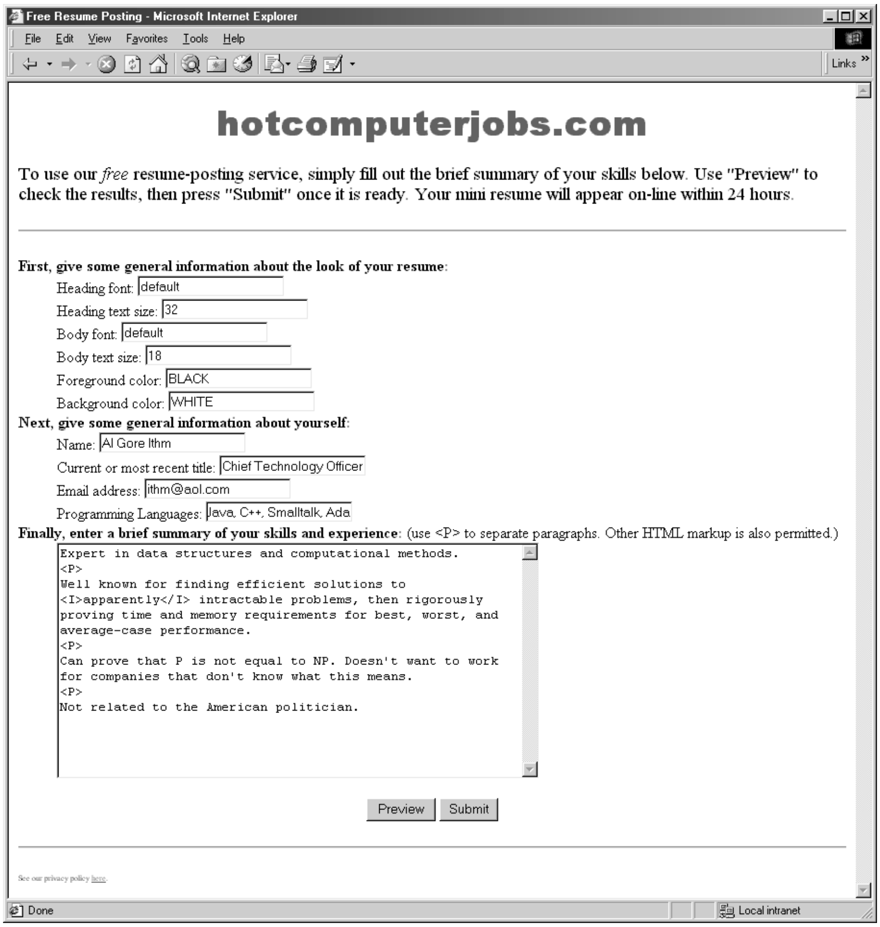
```
          "</CENTER><BR><BR>\n" +
          "<SPAN CLASS=\"HEADING3\">Programming Languages" +
          "</SPAN>\n" +
          makeList(languages) + "<BR><BR>\n" +
          "<SPAN CLASS=\"HEADING3\">Skills and Experience" +
          "</SPAN><BR><BR>\n" +
          skills + "\n" +
          "</BODY></HTML>");
  }

  // Builds a cascading style sheet with information
  // on three levels of headings and overall
  // foreground and background cover. Also tells
  // Internet Explorer to change color of mailto link
  // when mouse moves over it.

  private String makeStyleSheet(String headingFont,
                                int heading1Size,
                                String bodyFont,
                                int bodySize,
                                String fgColor,
                                String bgColor) {
    int heading2Size = heading1Size*7/10;
    int heading3Size = heading1Size*6/10;
    String styleSheet =
      "<STYLE TYPE=\"text/css\">\n" +
      "<!--\n" +
      ".HEADING1 { font-size: " + heading1Size + "px;\n" +
      "            font-weight: bold;\n" +
      "            font-family: " + headingFont +
                     "Arial, Helvetica, sans-serif;\n" +
      "}\n" +
      ".HEADING2 { font-size: " + heading2Size + "px;\n" +
      "            font-weight: bold;\n" +
      "            font-family: " + headingFont +
                     "Arial, Helvetica, sans-serif;\n" +
      "}\n" +
      ".HEADING3 { font-size: " + heading3Size + "px;\n" +
      "            font-weight: bold;\n" +
      "            font-family: " + headingFont +
                     "Arial, Helvetica, sans-serif;\n" +
      "}\n" +
      "BODY { color: " + fgColor + ";\n" +
      "       background-color: " + bgColor + ";\n" +
      "       font-size: " + bodySize + "px;\n" +
      "       font-family: " + bodyFont +
                    "Times New Roman, Times, serif;\n" +
```

Listing 3.6 `SubmitResume.java` **(continued)**

```
      "}\n" +
      "A:hover { color: red; }\n" +
      "-->\n" +
      "</STYLE>";
    return(styleSheet);
  }

  // Replaces null strings (no such parameter name) or
  // empty strings (e.g., if textfield was blank) with
  // the replacement. Returns the original string otherwise.

  private String replaceIfMissing(String orig,
                                  String replacement) {
    if ((orig == null) || (orig.length() == 0)) {
      return(replacement);
    } else {
      return(orig);
    }
  }

  // Replaces null strings, empty strings, or the string
  // "default" with the replacement.
  // Returns the original string otherwise.

  private String replaceIfMissingOrDefault(String orig,
                                           String replacement) {
    if ((orig == null) ||
        (orig.length() == 0) ||
        (orig.equals("default"))) {
      return(replacement);
    } else {
      return(orig + ", ");
    }
  }

  // Takes a string representing an integer and returns it
  // as an int. Returns a default if the string is null
  // or in an illegal format.

  private int getSize(String sizeString, int defaultSize) {
    try {
      return(Integer.parseInt(sizeString));
    } catch(NumberFormatException nfe) {
      return(defaultSize);
    }
  }
```

Listing 3.6 `SubmitResume.java` (continued)

```
  // Given "Java,C++,Lisp", "Java C++ Lisp" or
  // "Java, C++, Lisp", returns
  // "<UL>
  //   <LI>Java
  //   <LI>C++
  //   <LI>Lisp
  //  </UL>"

  private String makeList(String listItems) {
    StringTokenizer tokenizer =
      new StringTokenizer(listItems, ", ");
    String list = "<UL>\n";
    while(tokenizer.hasMoreTokens()) {
      list = list + "  <LI>" + tokenizer.nextToken() + "\n";
    }
    list = list + "</UL>";
    return(list);
  }

  // Show a confirmation page when they press the
  // "Submit" button.

  private void showConfirmation(HttpServletRequest request,
                                PrintWriter out) {
    String title = "Submission Confirmed.";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY>\n" +
                "<H1>" + title + "</H1>\n" +
                "Your resume should appear on-line within\n" +
                "24 hours. If it doesn't, try submitting\n" +
                "again with a different email address.\n" +
                "</BODY></HTML>");
  }

  // Why it is bad to give your email address to untrusted sites

  private void storeResume(HttpServletRequest request) {
    String email = request.getParameter("email");
    putInSpamList(email);
  }

  private void putInSpamList(String emailAddress) {
    // Code removed to protect the guilty.
  }
}
```


Free Resume Posting - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Done
Local intranet

hotcomputerjobs.com

To use our *free* resume-posting service, simply fill out the brief summary of your skills below. Use "Preview" to check the results, then press "Submit" once it is ready. Your mini resume will appear on-line within 24 hours.

First, give some general information about the look of your resume:

Heading font: default
Heading text size: 32
Body font: default
Body text size: 18
Foreground color: BLACK
Background color: WHITE

Next, give some general information about yourself:

Name: Al Gore Ithm
Current or most recent title: Chief Technology Officer
Email address: ithm@aol.com
Programming Languages: Java, C++, Smalltalk, Ada

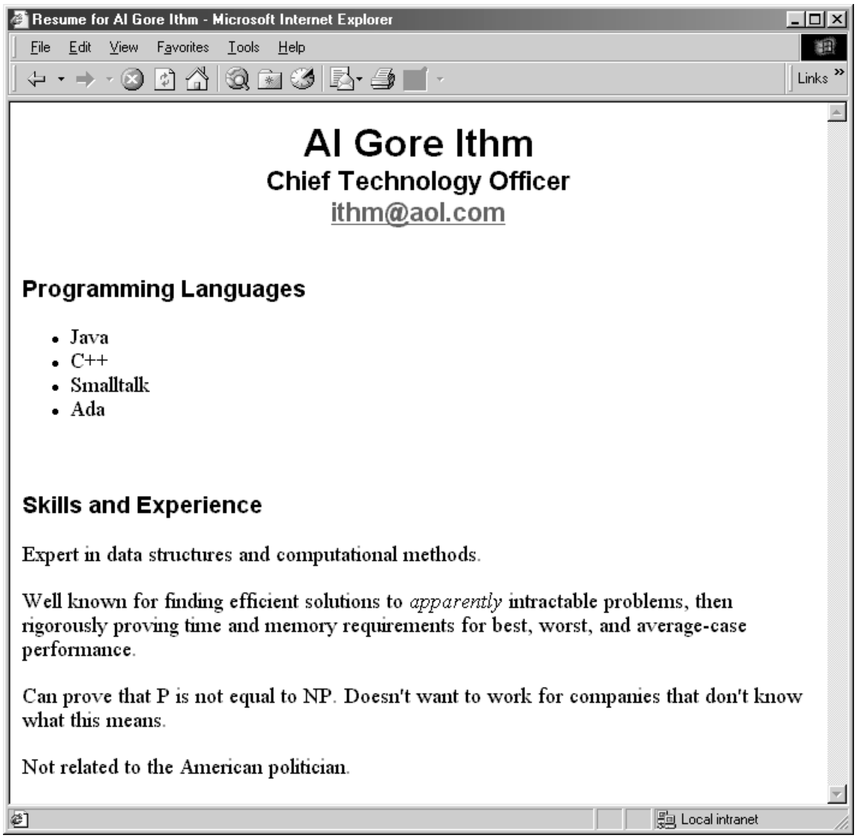
Finally, enter a brief summary of your skills and experience: (use <P> to separate paragraphs. Other HTML markup is also permitted.)

```
Expert in data structures and computational methods.
<P>
Well known for finding efficient solutions to
<I>apparently</I> intractable problems, then rigorously
proving time and memory requirements for best, worst, and
average-case performance.
<P>
Can prove that P is not equal to NP. Doesn't want to work
for companies that don't know what this means.
<P>
Not related to the American politician.
```

Preview Submit

See our privacy policy here.

Figure 3–5 Front end to `SubmitResume` servlet.


Resume for Al Gore Ithm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Local intranet

Al Gore Ithm
Chief Technology Officer
ithm@aol.com

Programming Languages

- Java
- C++
- Smalltalk
- Ada

Skills and Experience

Expert in data structures and computational methods.

Well known for finding efficient solutions to *apparently* intractable problems, then rigorously proving time and memory requirements for best, worst, and average-case performance.

Can prove that P is not equal to NP. Doesn't want to work for companies that don't know what this means.

Not related to the American politician.

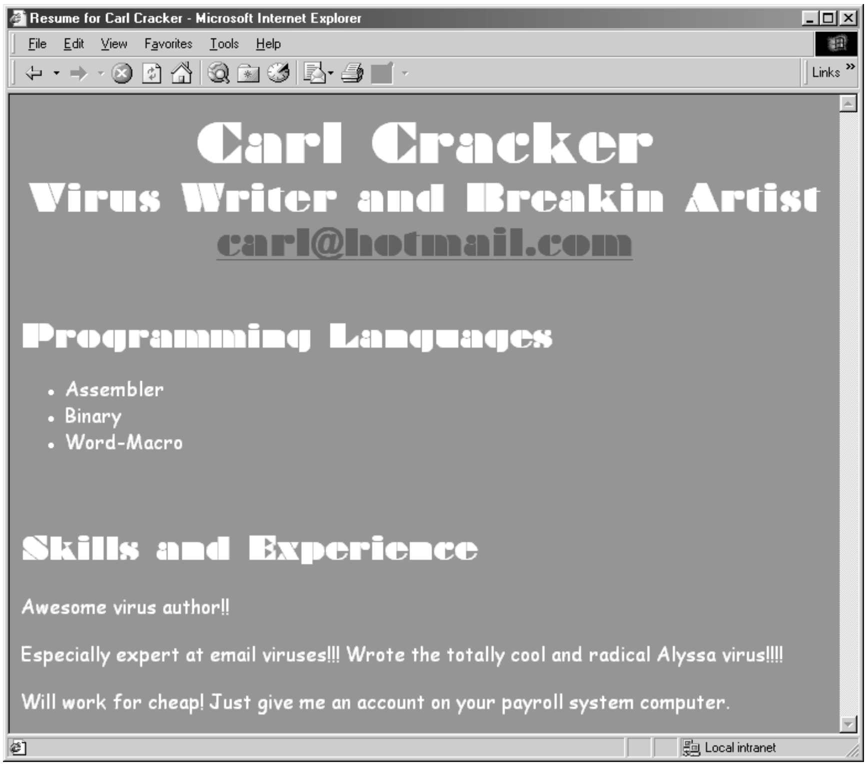
Figure 3–6 `SubmitResume` servlet after "Preview" button is pressed in Figure 3–5.

Listing 3.7 HTML source of `SubmitResume` output shown in Figure 3–6.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Resume for Al Gore Ithm</TITLE>
<STYLE TYPE="text/css">
<!--
.HEADING1 { font-size: 32px;
            font-weight: bold;
            font-family: Arial, Helvetica, sans-serif;
}
```

Listing 3.7 HTML source of `SubmitResume` output shown in Figure 3–6. (continued)

```
.HEADING2 { font-size: 22px;
            font-weight: bold;
            font-family: Arial, Helvetica, sans-serif;
}

.HEADING3 { font-size: 19px;
            font-weight: bold;
            font-family: Arial, Helvetica, sans-serif;
}
BODY { color: BLACK;
       background-color: WHITE;
       font-size: 18px;
       font-family: Times New Roman, Times, serif;
}
A:hover { color: red; }
-->
</STYLE>
</HEAD>
<BODY>
<CENTER>
<SPAN CLASS="HEADING1">Al Gore Ithm</SPAN><BR>
<SPAN CLASS="HEADING2">Chief Technology Officer<BR>
<A HREF="mailto:ithm@aol.com">ithm@aol.com</A></SPAN>
</CENTER><BR><BR>
<SPAN CLASS="HEADING3">Programming Languages</SPAN>

<UL>
  <LI>Java
  <LI>C++
  <LI>Smalltalk
  <LI>Ada
</UL><BR><BR>
<SPAN CLASS="HEADING3">Skills and Experience</SPAN><BR><BR>
Expert in data structures and computational methods.
<P>

Well known for finding efficient solutions to
<I>apparently</I> intractable problems, then rigorously
proving time and memory requirements for best, worst, and
average-case performance.
<P>
Can prove that P is not equal to NP. Doesn't want to work
for companies that don't know what this means.
<P>
Not related to the American politician.
</BODY></HTML>
```


Resume for Carl Cracker - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Local intranet

Carl Cracker

Virus Writer and Breakin Artist

carl@hotmail.com

Programming Languages

- Assembler
- Binary
- Word-Macro

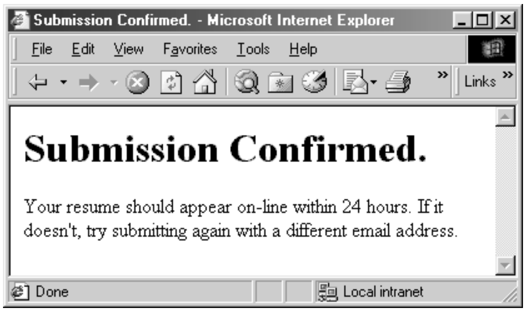
Skills and Experience

Awesome virus author!!

Especially expert at email viruses!!! Wrote the totally cool and radical Alyssa virus!!!!

Will work for cheap! Just give me an account on your payroll system computer.

Figure 3–7 Another possible result of `SubmitResume` servlet.


Submission Confirmed. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Done
Local intranet

Submission Confirmed.

Your resume should appear on-line within 24 hours. If it doesn't, try submitting again with a different email address.

Figure 3–8 `SubmitResume` servlet when "Submit" button is pressed.

3.6 Filtering Strings for HTML-Specific Characters

Normally, when a servlet wants to generate HTML that will contain characters like < or >, it simply uses `&lt;` or `&gt;`, the standard HTML character entities. Similarly, if a servlet wants a double quote or an ampersand to appear inside an HTML attribute value, it uses `&quot;` or `&amp;`. Failing to make these substitutions results in malformed HTML code, since < or > will often get interpreted as part of an HTML markup tag, a double quote in an attribute value may be interpreted as the end of the value, and ampersands are just plain illegal in attribute values. In most cases, it is easy to note the special characters and use the standard HTML replacements. However, there are two cases when it is not so easy to make this substitution manually.

The first case where manual conversion is difficult occurs when the string is derived from a program excerpt or another source where it is already in some standard format. Going through manually and changing all the special characters can be tedious in such a case, but forgetting to convert even one special character can result in your Web page having missing or improperly formatted sections (see Figure 3–9 later in this section).

The second case where manual conversion fails is when the string is derived from HTML form data. Here, the conversion absolutely must be performed at runtime, since of course the query data is not known at compile time. Failing to do this for an internal Web page can also result in missing or improperly formatted sections of the servlet's output if the user ever sends these special characters. Failing to do this filtering for externally-accessible Web pages also lets your page become a vehicle for the *cross-site scripting attack*. Here, a malicious programmer embeds `GET` parameters in a URL that refers to one of your servlets. These `GET` parameters expand to HTML `<SCRIPT>` elements that exploit known browser bugs. However, by embedding the code in a URL that refers to your site and only distributing the URL, not the malicious Web page itself, the attacker can remain undiscovered more easily and can also exploit trusted relationships to make users think the scripts are coming from a trusted source (your servlet). For more details on this issue, see `http://www.cert.org/advisories/ CA-2000-02.html` and `http://www.microsoft.com/technet/security/crssite.asp`.

Code for Filtering

Replacing `<`, `>`, `"`, and `&` in strings is a simple matter, and there are a number of different approaches that would accomplish the task. However, it is important to remember that Java strings are immutable (i.e., can't be modified), so string concatenation involves copying and then discarding many string segments. For example, consider the following two lines:

```
String s1 = "Hello";
String s2 = s1 + " World";
```

Since `s1` cannot be modified, the second line makes a copy of `s1` and appends `"World"` to the copy, then the copy is discarded. To avoid the expense of generating these temporary objects (garbage), you should use a mutable data structure, and `StringBuffer` is the natural choice. Listing 3.8 shows a static `filter` method that uses a `StringBuffer` to efficiently copy characters from an input string to a filtered version, replacing the four special characters along the way.

Listing 3.8 `ServletUtilities.java`

```
package coreservlets;

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

public class ServletUtilities {

  // Other methods in ServletUtilities shown elsewhere...

  /** Given a string, this method replaces all occurrences of
   *  '<' with '&lt;', all occurrences of '>' with
   *  '&gt;', and (to handle cases that occur inside attribute
   *  values), all occurrences of double quotes with
   *  '&quot;' and all occurrences of '&' with '&amp;'.
   *  Without such filtering, an arbitrary string
   *  could not safely be inserted in a Web page.
   */

  public static String filter(String input) {
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
      c = input.charAt(i);
      if (c == '<') {
        filtered.append("&lt;");
      } else if (c == '>') {
        filtered.append("&gt;");
```

Listing 3.8 ServletUtilities.java (continued)

```
      } else if (c == '"') {
        filtered.append("&quot;");
      } else if (c == '&') {
        filtered.append("&amp;");
      } else {
        filtered.append(c);
      }
    }
    return(filtered.toString());
  }
}
```

Example

By means of illustration, consider a servlet that attempts to generate a Web page containing the following code listing:

```
if (a<b) {
  doThis();
} else {
  doThat();
}
```

If the code was inserted into the Web page verbatim, the `<b` would be interpreted as the beginning of an HTML tag, and all of the code up to the next > would likely be interpreted as malformed pieces of that tag. For example, Listing 3.9 shows a servlet that outputs this code fragment, and Figure 3–9 shows the poor result. Listing 3.10 presents a servlet that changes nothing except for filtering the string containing the code fragment, and, as Figure 3–10 illustrates, the result is fine.

Listing 3.9 BadCodeServlet.java

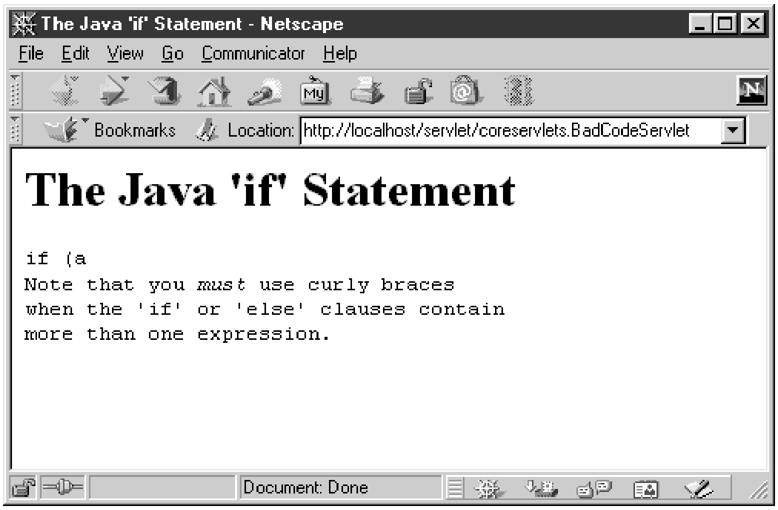
```
package coreservlets;

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

/** Servlet that displays a fragment of some Java code,
 *  but forgets to filter out the HTML-specific characters
 *  (the less-than sign in this case).
 */

public class BadCodeServlet extends HttpServlet {
  private String codeFragment =
```

Listing 3.9 `BadCodeServlet.java` **(continued)**

```
    "if (a<b) {\n" +
    "  doThis();\n" +
    "} else {\n" +
    "  doThat();\n" +
    "}\n";

  public String getCodeFragment() {
    return(codeFragment);
  }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "The Java 'if' Statement";

    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY>\n" +
                "<H1>" + title + "</H1>\n" +
                "<PRE>\n" +
                getCodeFragment() +
                "</PRE>\n" +
                "Note that you <I>must</I> use curly braces\n" +
                "when the 'if' or 'else' clauses contain\n" +
                "more than one expression.\n" +
                "</BODY></HTML>");
  }
}
```

Listing 3.10 `FilteredCodeServlet.java`

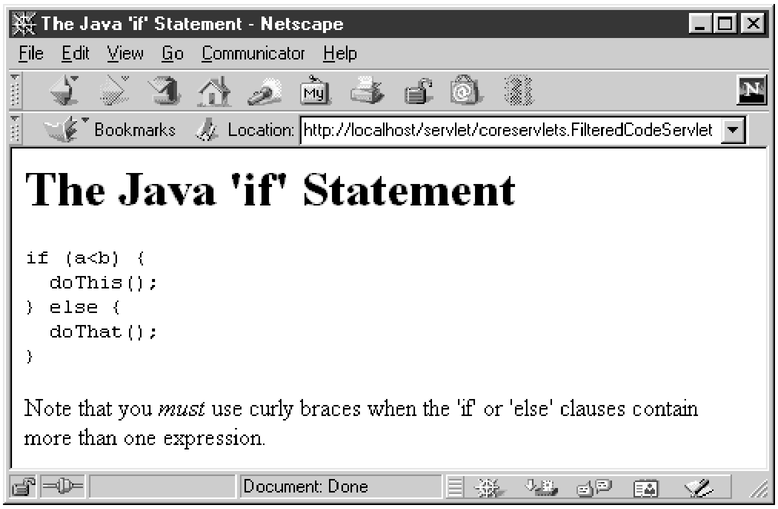
```
package coreservlets;

/** Subclass of BadCodeServlet that keeps the same doGet method
 *  but filters the code fragment for HTML-specific characters.
 *  You should filter strings that are likely to contain
 *  special characters (like program excerpts) or strings
 *  that are derived from user input.
 */

public class FilteredCodeServlet extends BadCodeServlet {
  public String getCodeFragment() {
    return(ServletUtilities.filter(super.getCodeFragment()));
  }
}
```


The Java 'if' Statement - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/servlet/coreservlets.BadCodeServlet
Document: Done

The Java 'if' Statement

```
if (a
Note that you must use curly braces
when the 'if' or 'else' clauses contain
more than one expression.
```

Figure 3–9 Result of `BadCodeServlet`: much of the code fragment is lost, and the text following the code fragment is incorrectly displayed in a monospaced font.


The Java 'if' Statement - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://localhost/servlet/coreservlets.FilteredCodeServlet
Document: Done

The Java 'if' Statement

```
if (a<b) {
  doThis();
} else {
  doThat();
}
```

Note that you *must* use curly braces when the 'if' or 'else' clauses contain more than one expression.

Figure 3–10 Result of `FilteredCodeServlet`: use of the `filter` method solves problems with strings containing special characters.