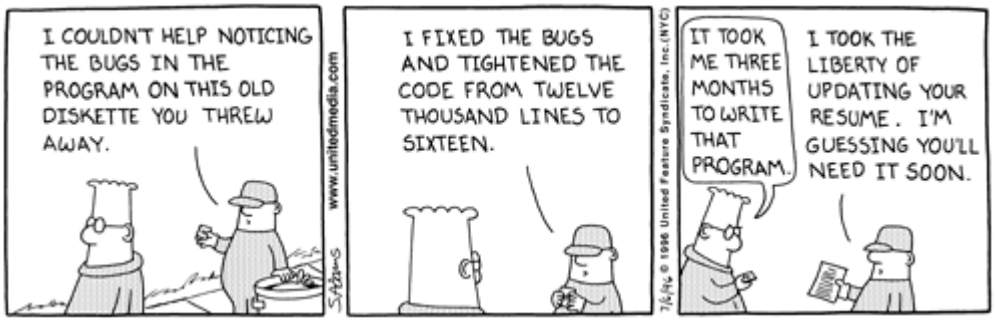
4.5 Using Default Values When Parameters Are Missing or Malformed

Online 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 creation of online résumés from user-submitted data. Now, the question is: what should the servlet do when the user fails to supply the necessary information? This question has two answers: use default values or redisplay the form (prompting the user for missing values). This section illustrates the use of default values; Section 4.8 illustrates redisplay of the form.

DILBERT reprinted by permission of United Feature Syndicate, Inc.



When examining request parameters, you need to check for three conditions:

1. **The value is `null`.** A call to `request.getParameter` returns `null` if the form contains no textfield or other element of the expected name so that the parameter name does not appear in the request at all. This can happen when the end user uses an incorrect HTML form or when a bookmarked URL containing `GET` data is used but the parameter names have changed since the URL was bookmarked. To avoid a `NullPointerException`, you have to check for `null` *before* you try to call any methods on the string that results from `getParameter`.

2. **The value is an empty string.** A call to `request.getParameter` returns an empty string (i.e., `""`) if the associated textfield is empty when the form is submitted. To check for an empty string, compare the string to `""` by using `equals` or compare the length of the string to 0. *Do not use the `==` operator*; in the Java programming language, `==` always tests whether the two arguments are the same object (at the same memory location), not whether the two objects look similar. Just to be safe, it is also a good idea to call `trim` to remove any white space that the user may have entered, since in most scenarios you want to treat pure white space as missing data. So, for example, a test for missing values might look like the following.

   ```
   String param = request.getParameter("someName");
   if ((param == null) || (param.trim().equals(""))) {
     doSomethingForMissingValues(...);
   } else {
     doSomethingWithParameter(param);
   }
   ```

3. **The value is a nonempty string of the wrong format.** What defines the wrong format is application specific: you might expect certain textfields to contain only numeric values, others to have exactly seven characters, and others to only contain single letters.

Note that the use of JavaScript for client-side validation does not remove the need for also doing this type of checking on the server. After all, you are responsible for the server-side application, and often another developer or group is responsible for the forms. You do not want *your* application to crash if *they* fail to detect every type of illegal input. Besides, clients can use their own HTML forms, can manually edit URLs that contain `GET` data, and can disable JavaScript.

Core Approach

Design your servlets to gracefully handle parameters that are missing (`null` or empty string) or improperly formatted. Test your servlets with missing and malformed data as well as with data in the expected format.

Listing 4.5 and Figure 4-5 show the HTML form that acts as the front end to the résumé-processing servlet. If you are not familiar with HTML forms, see Chapter 19. The form uses `POST` to submit the data and it gathers values for various parameter names. The important thing to understand here is what the servlet does with missing and malformed data. This process is summarized in the following list. Listing 4.6 shows the complete servlet code.

- **`name`**, **`title`**, **`email`**, **`languages`**, **`skills`**

  These parameters specify various parts of the résumé. Missing values should be replaced by default values specific to the parameter. The servlet uses a method called `replaceIfMissing` to accomplish this task.

- **`fgColor`**, **`bgColor`**

  These parameters give the colors of the foreground and background of the page. Missing values should result in black for the foreground and white for the background. The servlet again uses `replaceIfMissing` to accomplish this task.

- **`headingFont`**, **`bodyFont`**

  These parameters designate the font to use for headings and the main text, respectively. A missing value or a value of "default" should result in a sans-serif font such as Arial or Helvetica. The servlet uses a method called `replaceIfMissingOrDefault` to accomplish this task.

- **`headingSize`**, **`bodySize`**

  These parameters specify the point size for main headings and body text, respectively. Subheadings will be displayed in a slightly smaller size than the main headings. Missing values or nonnumeric values should result in a default size (32 for headings, 18 for body). The servlet uses a call to `Integer.parseInt` and a `try`/`catch` block for `NumberFormatException` to handle this case.

Listing 4.5 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>hot-computer-jobs.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 online 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">

<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=10 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 4.6 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);
  }
}

/** 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@hot-computer-jobs.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><HEAD><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" +
     "</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" +
      "}\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.trim().equals(""))) {
      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.trim().equals("")) ||
      (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);
  }
}

// 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);
}

/** Shows a confirmation page when the user clicks 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 online 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.
  }
}
```

Figure 4-5. Front end to résumé-previewing servlet.

Once the servlet has meaningful values for each of the font and color parameters, it builds a cascading style sheet out of them. Style sheets 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 use of 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 email address centered under each other at the top of the page. The heading font is used for these lines, and the email 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 by `StringTokenizer` (assuming spaces 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 4-6 and 4-7 show results when the required data is supplied and omitted, respectively. Figure 4-8 shows the result of clicking Submit instead of Preview.

Figure 4-6. Preview of a résumé submission that contained the required form data.

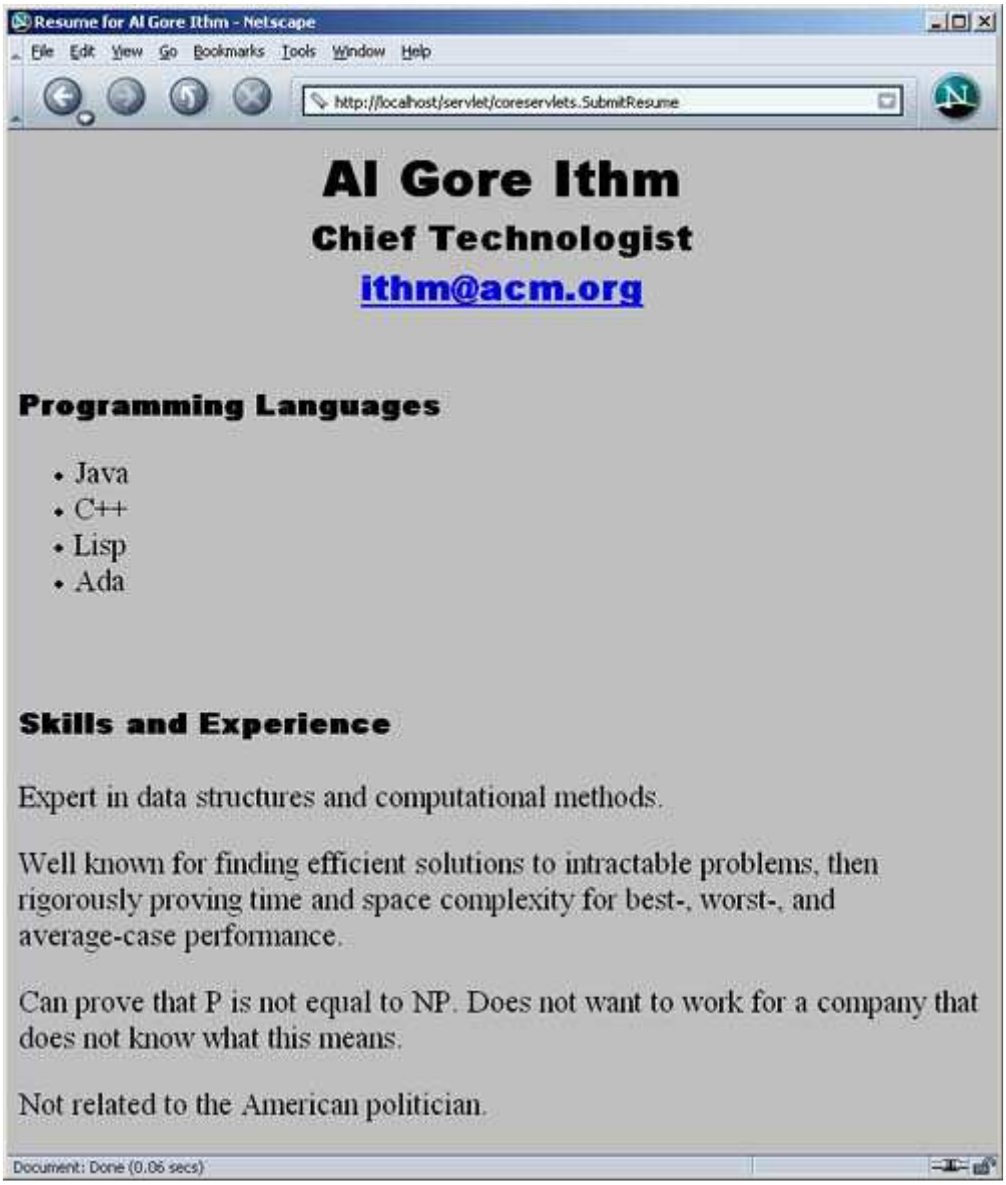


Figure 4-7. Preview of a submission that was missing much of the required data: default values replace the omitted values.

Figure 4-8. Result of submitting the résumé to the database.

[Team LiB]

PREVIOUS NEXT