4.8 Redisplaying the Input Form When Parameters Are Missing or Malformed

It sometimes makes sense to use default values when the user fails to fill in certain form fields. Other times, however, there are no reasonable default values to use, and the form should be redisplayed to the user. Two desirable capabilities make the use of a normal HTML form impossible in this scenario:

- Users should not have to reenter values that they already supplied.
- Missing form fields should be marked prominently.

Redisplay Options

So, what can be done to implement these capabilities? Well, a full description of the possible approaches is a bit complicated and requires knowledge of several techniques (e.g., Struts, JSTL) that are not covered in Volume 1 of this book. So, you should refer to Volume 2 for details, but here is a quick preview.

- **Have the same servlet present the form, process the data, and present the results.** The servlet first looks for incoming request data: if it finds none, it presents a blank form. If the servlet finds partial request data, it extracts the partial data, puts it back into the form, and marks the other fields as missing. If the servlet finds the full complement of required data, it processes the request and displays the results. The form omits the `ACTION` attribute so that form submissions automatically go to the same URL as the form itself. This is the only approach for which we have already covered all of the necessary techniques, so this is the approach illustrated in this section.
- **Have one servlet present the form; have a second servlet process the data and present the results.** This option is better than the first since it divides up the labor and keeps each servlet smaller and more manageable. However, using this approach requires two techniques we have not yet covered: how to transfer control from one servlet to another and how to access user-specific data in one servlet that was created in another. Transferring from one servlet to another can be done with `response.sendRedirect` (see Chapter 6, "Generating the Server Response: HTTP Status Codes") or the `forward` method of `RequestDispatcher` (see Chapter 15, "Integrating Servlets and JSP: The Model View Controller (MVC) Architecture"). The easiest way to pass the data from the processing servlet back to the form-display servlet is to store it in the `HttpSession` object (see Chapter 9, "Session Tracking").
- **Have a JSP page "manually" present the form; have a servlet or JSP page process the data and present the results.** This is an excellent option, and it is widely used. However, it requires knowledge of JSP in addition to knowledge of the two techniques mentioned in the previous bullet (how to transfer the user from the data-processing servlet to the form page and how to use session tracking to store user-specific data). In particular, you need to know how to use JSP expressions (see Chapter 11, "Invoking Java Code with JSP Scripting Elements") or `jsp:getProperty` (see Chapter 14, "Using JavaBeans Components in JSP Documents") to extract the partial data from the data object and put it into the HTML form.
- **Have a JSP page present the form, automatically filling in the fields with values obtained from a data object. Have a servlet or JSP page process the data and present the results.** This is perhaps the best option of all. But, in addition to the

techniques described in the previous bullet, it requires custom JSP tags that mimic HTML form elements but use designated values automatically. You can write these tags yourself or you can use ready-made versions such as those that come with JSTL or Apache Struts (see Volume 2 for coverage of custom tags, JSTL, and Struts).

A Servlet That Processes Auction Bids

To illustrate the first of the form-redisplay options, consider a servlet that processes bids at an auction site. Figures 4-14 through 4-16 show the desired outcome: the servlet initially displays a blank form, redisplays the form with missing data marked when partial data is submitted, and processes the request when complete data is submitted.

Figure 4-14. Original form of servlet: it presents a form to collect data about a bid at an auction.

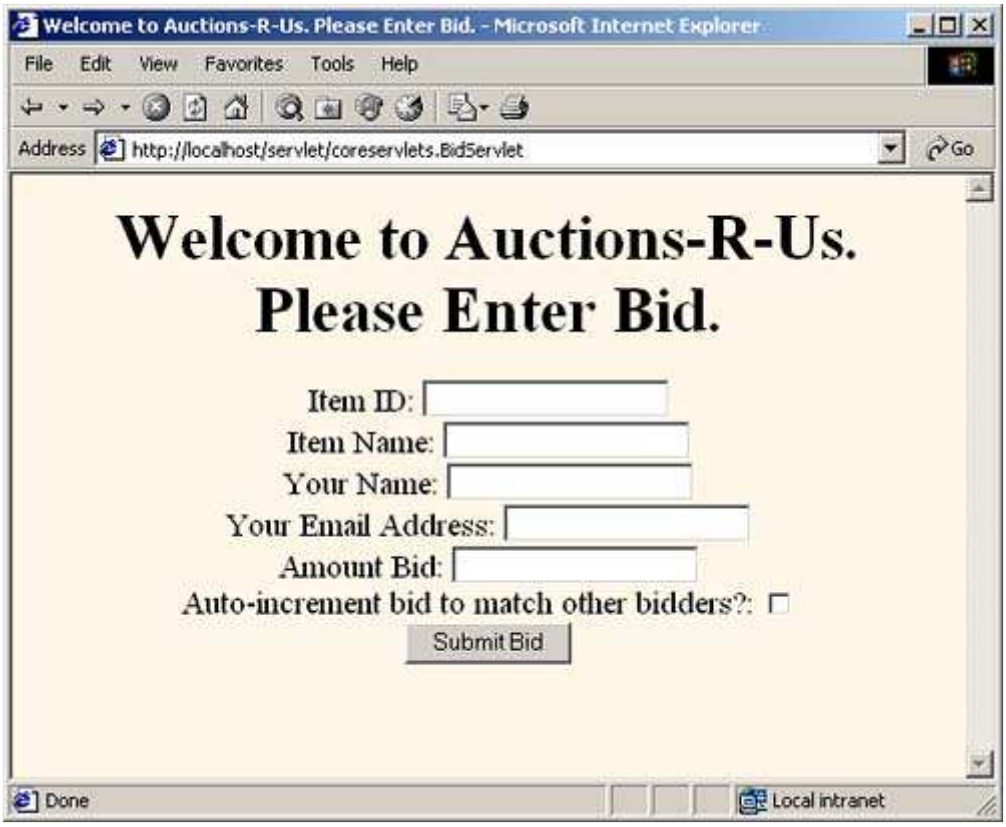

Figure 4-16. Bid servlet with complete data: it presents the results.

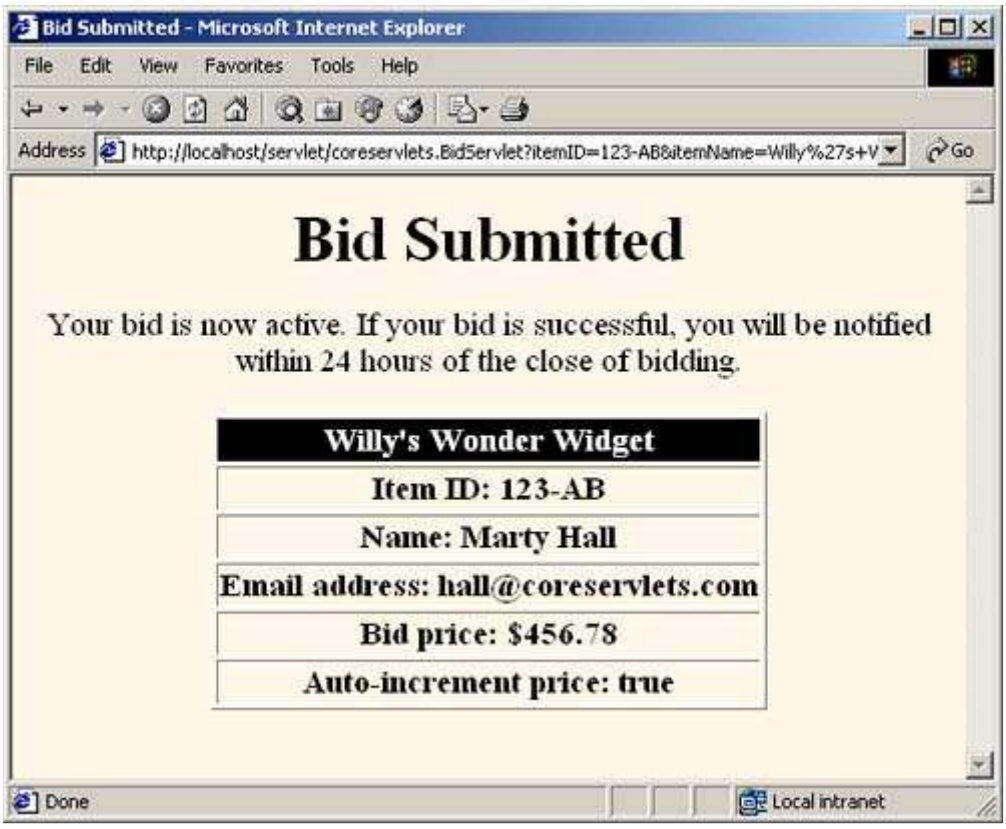

Figure 4-15. Bid servlet with incomplete data. If the user submits a form that is not fully filled in, the bid servlet redisplays the form. The user's previous partial data is maintained, and missing fields are highlighted.

To accomplish this behavior, the servlet (Listing 4.16) performs the following steps.

1. Fills in a `BidInfo` object (Listing 4.17) from the request data, using `BeanUtilities.populateBean` (see Section 4.7, "Automatically Populating Java Objects from Request Parameters: Form Beans") to automatically match up request parameter names with bean properties and to perform simple type conversion.

2. Checks whether that `BidInfo` object is completely empty (no fields changed from the default). If so, it calls `showEntryForm` to display the initial input form.

3. Checks whether the `BidInfo` object is partially empty (some, but not all, fields changed from the default). If so, it calls `showEntryForm` to display the input form with a warning message and with the missing fields highlighted. Fields in which the user already entered data keep their previous values.

4. Checks whether the `BidInfo` object is completely filled in. If so, it calls `showBid` to process the data and present the result.

Listing 4.16 BidServlet.java

```
package coreservlets;

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

/** Example of simplified form processing. Shows two features:
 *  <OL>
 *    <LI>Automatically filling in a bean based on the
 *        incoming request parameters.
 *    <LI>Using the same servlet both to generate the input
 *        form and to process the results. That way, when
 *        fields are omitted, the servlet can redisplay the
 *        form without making the user reenter previously
 *        entered values.
 *  </UL>
 */

public class BidServlet extends HttpServlet {

  /** Try to populate a bean that represents information
   *  in the form data sent by the user. If this data is
   *  complete, show the results. If the form data is
   *  missing or incomplete, display the HTML form
   *  that gathers the data.
   */

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    BidInfo bid = new BidInfo();
    BeanUtilities.populateBean(bid, request);
    if (bid.isComplete()) {
      // All required form data was supplied: show result.
      showBid(request, response, bid);
    } else {
      // Form data was missing or incomplete: redisplay form.
      showEntryForm(request, response, bid);
    }
  }

  /** All required data is present: show the results page. */
```

```
private void showBid(HttpServletRequest request,
                     HttpServletResponse response,
                     BidInfo bid)
    throws ServletException, IOException {
  submitBid(bid);
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String title = "Bid Submitted";
  out.println
    (DOCTYPE +
     "<HTML>\n" +
     "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
     "<BODY BGCOLOR=\"#FDF5E6\"><CENTER>\n" +
     "<H1>" + title + "</H1>\n" +
     "Your bid is now active. If your bid is successful,\n" +
     "you will be notified within 24 hours of the close\n" +
     "of bidding.\n" +
     "<P>\n" +
     "<TABLE BORDER=1>\n" +
     "  <TR><TH BGCOLOR=\"BLACK\"><FONT COLOR=\"WHITE\">" +
     bid.getItemName() + "</FONT>\n" +
     "  <TR><TH>Item ID: " +
     bid.getItemID() + "\n" +
     "  <TR><TH>Name: " +
     bid.getBidderName() + "\n" +
     "  <TR><TH>Email address: " +
     bid.getEmailAddress() + "\n" +
     "  <TR><TH>Bid price: $" +
     bid.getBidPrice() + "\n" +
     "  <TR><TH>Auto-increment price: " +
     bid.isAutoIncrement() + "\n" +
     "</TABLE></CENTER></BODY></HTML>");
}

/** If the required data is totally missing, show a blank
 *  form. If the required data is partially missing,
 *  warn the user, fill in form fields that already have
 *  values, and prompt user for missing fields.
 */

private void showEntryForm(HttpServletRequest request,
                   HttpServletResponse response,
                   BidInfo bid)
    throws ServletException, IOException {
  boolean isPartlyComplete = bid.isPartlyComplete();
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String title =
    "Welcome to Auctions-R-Us. Please Enter Bid.";
  out.println
    (DOCTYPE +
     "<HTML>\n" +
     "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
     "<BODY BGCOLOR=\"#FDF5E6\"><CENTER>\n" +
     "<H1>" + title + "</H1>\n" +
     warning(isPartlyComplete) +
     "<FORM>\n" +
     inputElement("Item ID", "itemID",
                  bid.getItemID(), isPartlyComplete) +
     inputElement("Item Name", "itemName",
                  bid.getItemName(), isPartlyComplete) +
     inputElement("Your Name", "bidderName",
                  bid.getBidderName(), isPartlyComplete) +
```

```
    inputElement("Your Email Address", "emailAddress",
                 bid.getEmailAddress(), isPartlyComplete) +
    inputElement("Amount Bid", "bidPrice",
                 bid.getBidPrice(), isPartlyComplete) +
    checkbox("Auto-increment bid to match other bidders?",
             "autoIncrement", bid.isAutoIncrement()) +
    "<INPUT TYPE=\"SUBMIT\" VALUE=\"Submit Bid\">\n" +
    "</CENTER></BODY></HTML>");
}

private void submitBid(BidInfo bid) {
  // Some application-specific code to record the bid.
  // The point is that you pass in a real object with
  // properties populated, not a bunch of strings.
}

private String warning(boolean isFormPartlyComplete) {
  if(isFormPartlyComplete) {
    return("<H2>Required Data Missing! " +
           "Enter and Resubmit.</H2>\n");
  } else {
    return("");
  }
}

/** Create a textfield for input, prefaced by a prompt.
 *  If this particular textfield is missing a value but
 *  other fields have values (i.e., a partially filled form
 *  was submitted), then add a warning telling the user that
 *  this textfield is required.
 */

private String inputElement(String prompt,
                            String name,
                            String value,
                            boolean shouldPrompt) {
  String message = "";
  if (shouldPrompt && ((value == null) || value.equals(""))) {
    message = "<B>Required field!</B> ";
  }
  return(message + prompt + ": " +
         "<INPUT TYPE=\"TEXT\" NAME=\"" + name + "\"" +
         " VALUE=\"" + value + "\"><BR>\n");
}

private String inputElement(String prompt,
                            String name,
                            double value,
                            boolean shouldPrompt) {
  String num;
  if (value == 0.0) {
    num = "";
  } else {
    num = String.valueOf(value);
  }
  return(inputElement(prompt, name, num, shouldPrompt));
}

private String checkbox(String prompt,
                        String name,
                        boolean isChecked) {
  String result =
    prompt + ": " +
    "<INPUT TYPE=\"CHECKBOX\" NAME=\"" + name + "\"";
```

```
    if (isChecked) {
      result = result + " CHECKED";
    }
    result = result + "><BR>\n";
    return(result);
  }

  private final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n";
}
```

Listing 4.17 BidInfo.java

```
package coreservlets.beans;

import coreservlets.*;

/** Bean that represents information about a bid at
 *  an auction site. Used to demonstrate redisplay of forms
 *  that have incomplete data.
 */

public class BidInfo {
  private String itemID = "";
  private String itemName = "";
  private String bidderName = "";
  private String emailAddress = "";
  private double bidPrice = 0;
  private boolean autoIncrement = false;
  public String getItemName() {
    return(itemName);
  }

  public void setItemName(String itemName) {
    this.itemName = ServletUtilities.filter(itemName);
  }

  public String getItemID() {
    return(itemID);
  }

  public void setItemID(String itemID) {
    this.itemID = ServletUtilities.filter(itemID);
  }

  public String getBidderName() {
    return(bidderName);
  }

  public void setBidderName(String bidderName) {
    this.bidderName =  ServletUtilities.filter(bidderName);
  }

  public String getEmailAddress() {
    return(emailAddress);
  }

  public void setEmailAddress(String emailAddress) {
    this.emailAddress = ServletUtilities.filter(emailAddress);
  }

  public double getBidPrice() {
    return(bidPrice);
```

```
  }

  public void setBidPrice(double bidPrice) {
    this.bidPrice = bidPrice;
  }

  public boolean isAutoIncrement() {
    return(autoIncrement);
  }

  public void setAutoIncrement(boolean autoIncrement) {
    this.autoIncrement = autoIncrement;
  }

  /** Has all the required data been entered? Everything except
      autoIncrement must be specified explicitly (autoIncrement
      defaults to false).
  */

  public boolean isComplete() {
    return(hasValue(getItemID()) &&
           hasValue(getItemName()) &&
           hasValue(getBidderName()) &&
           hasValue(getEmailAddress()) &&
           (getBidPrice() > 0));
  }

  /** Has any of the data been entered? */

  public boolean isPartlyComplete() {
    boolean flag =
           (hasValue(getItemID()) ||
           hasValue(getItemName()) ||
           hasValue(getBidderName()) ||
           hasValue(getEmailAddress()) ||
           (getBidPrice() > 0) ||
           isAutoIncrement());
    return(flag);
  }

  private boolean hasValue(String val) {
    return((val != null) && (!val.equals("")));
  }
}
```