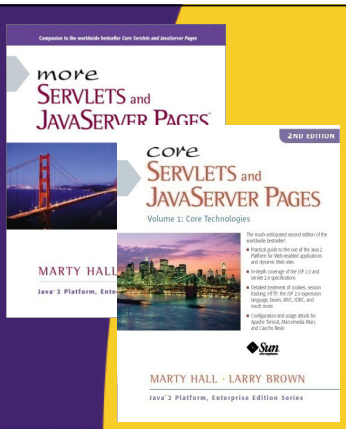
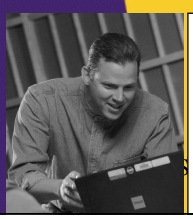
Handling the Client Request: Form Data

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/csajsp2.html





Agenda

- The role of form data
- Creating and submitting HTML forms
- Reading individual request parameters
- Reading the entire set of request parameters
- Handling missing and malformed data
- Dealing with incomplete form submissions
- Filtering special characters out of the request parameters

Form Basics

The Role of Form Data

- **Example URL at online travel agent**
  - http://host/path?user=Marty+Hall&origin=bwi&dest=lax
  - Names come from HTML author; values from end user
- **Parsing form (query) data in traditional CGI**
  - Read the data one way (QUERY_STRING) for GET requests, another way (standard input) for POST requests
  - Chop pairs at ampersands, then separate parameter names (left of the =) from parameter values (right of the =)
  - URL decode values (e.g., "%7E" becomes "~")
- **Greatly simplified in servlets**
  - Use request.getParameter in all cases.
  - Gives URL-decoded result

Creating Form Data: HTML Forms

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>A Sample Form Using GET</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2 ALIGN="CENTER">A Sample Form Using GET</H2>

<FORM ACTION="http://localhost:8088/SomeProgram">
  <CENTER>
  First name:
  <INPUT TYPE="TEXT" NAME="firstName" VALUE="J. Random"><BR>
  Last name:
  <INPUT TYPE="TEXT" NAME="lastName" VALUE="Hacker"><P>
  <INPUT TYPE="SUBMIT"> <!-- Press this to submit form -->
  </CENTER>
</FORM>
</BODY></HTML>
```

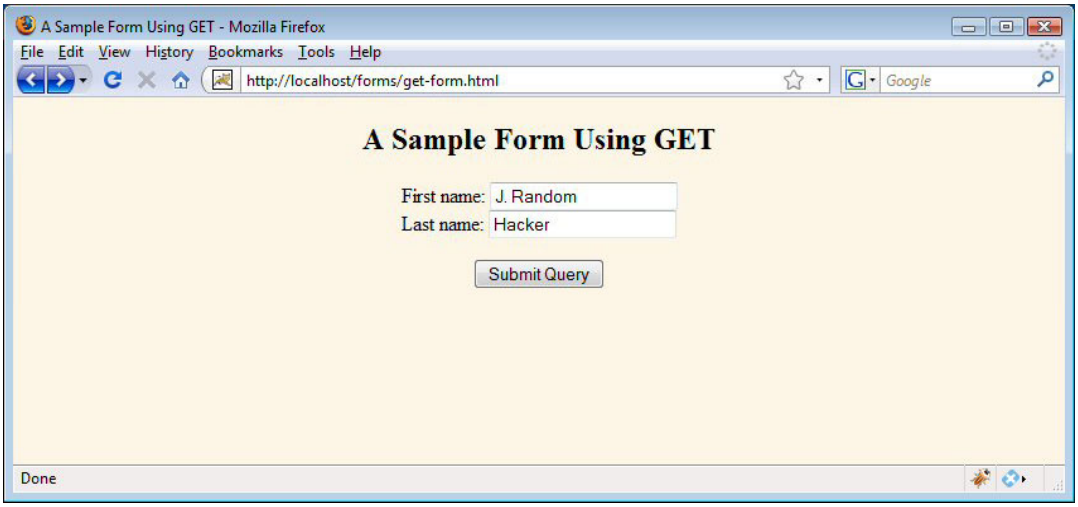
You normally use a relative URL for the ACTION. This URL is just for testing because I am running a test server on port 8088 that echoes the data it receives.

- **See *CSAJSP/2* Ch. 19 for details on forms**

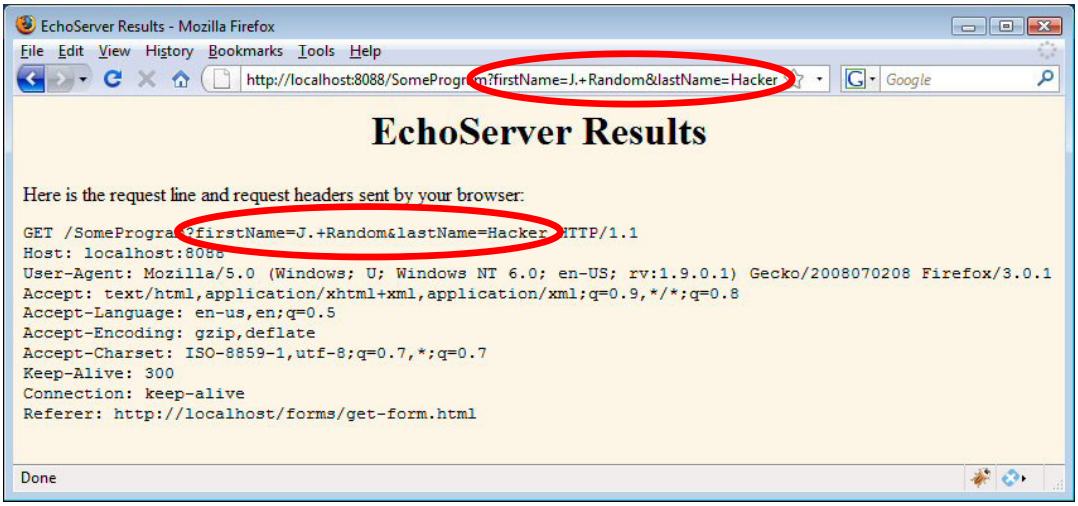
Aside: Installing HTML Files

- **HTML files do not go in src**
  - They go in WebContent
    - When deployed, that becomes the top-level Web application directory
    - In contrast, code under src gets deployed to the WEB-INF/classes folder of the Web app directory
- **Example**
  - Eclipse project name: forms
  - Files
    - WebContent/test1.html
    - WebContent/someDir/test2.html
  - URLs
    - http://localhost/forms/test1.html
    - http://localhost/forms/someDir/test2.html

GET Form: Initial Result



GET Form: Submission Result (Data Sent to EchoServer)

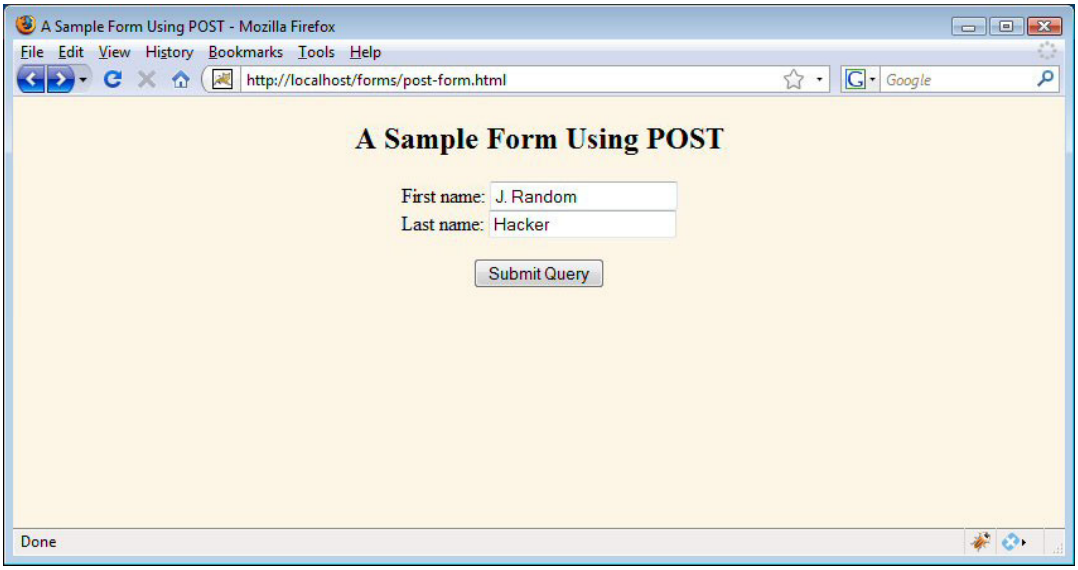


Sending POST Data

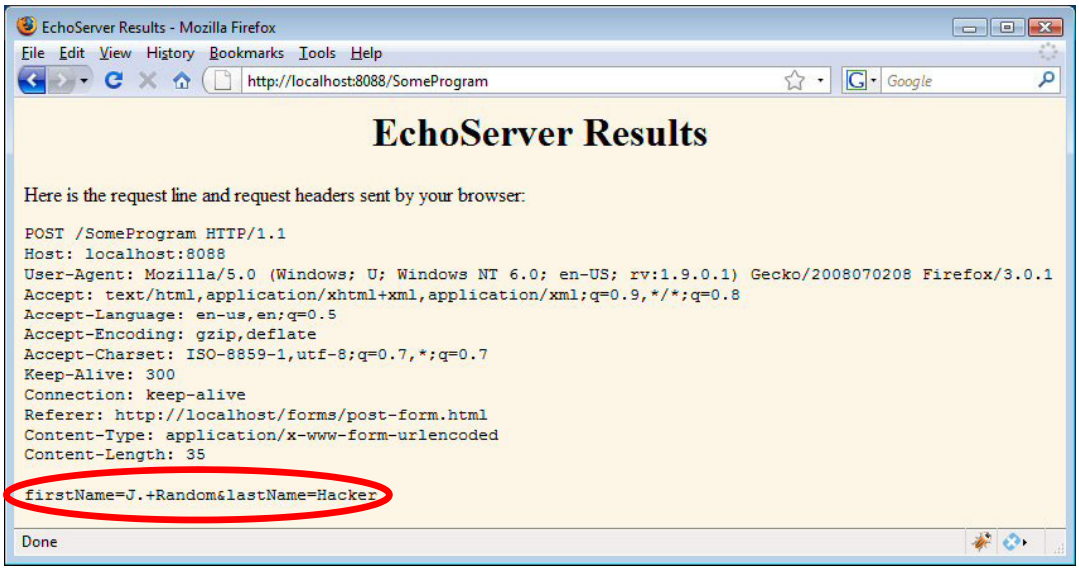
```
<!DOCTYPE ... >
<HTML>
<HEAD><TITLE>A Sample Form Using POST</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2 ALIGN="CENTER">A Sample Form Using POST</H2>

<FORM ACTION="http://localhost:8088/SomeProgram"
      METHOD="POST">
  <CENTER>
  First name:
  <INPUT TYPE="TEXT" NAME="firstName" VALUE="J. Random"><BR>
  Last name:
  <INPUT TYPE="TEXT" NAME="lastName" VALUE="Hacker"><P>
  <INPUT TYPE="SUBMIT">
  </CENTER>
</FORM>

</BODY></HTML>
```

The default method is GET. So, if a form says METHOD="GET" or it has no METHOD at all, GET is used.

POST Form: Initial Result



POST Form: Submission Result (Data Sent to EchoServer)



HTML 4 vs. XHTML Syntax

- **HTML 4**
  - Some end tags optional
  - Tag names and attribute names are case insensitive
- **XHTML**
  - End tags always required
    - If no body content, can use collapsed form like
  - Tag names and attribute names must be in lower case
- **HTML 5 DOCTYPE**
  - Most people who use the HTML 5 DOCTYPE do so as a convenience, and follow XHTML syntax in their pages.
- **Examples**
  - HTML 4
    - <INPUT TYPE="TEXT" NAME="Blah">
  - XHTML
    - <input type="text" name="Blah"/>

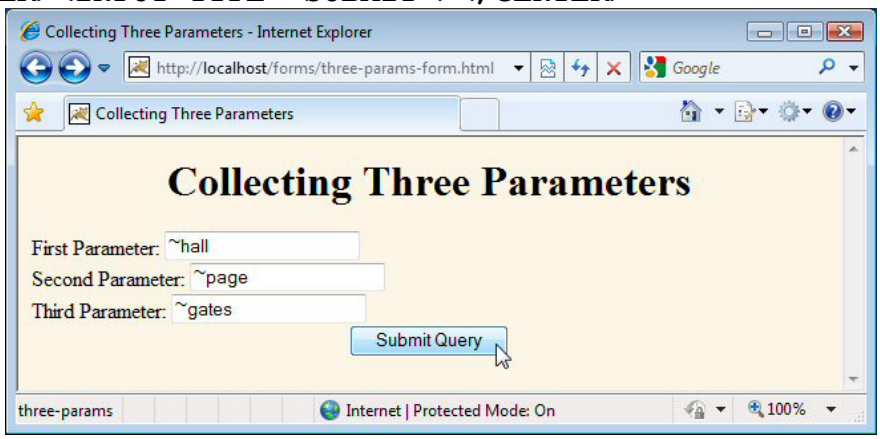
Reading Form Data

Reading Form Data In Servlets

- **request.getParameter("name")**
  - Returns URL-decoded value of first occurrence of name in query string
  - Works identically for GET and POST requests
  - Returns null if no such parameter is in query data
- **request.getParameterValues("name")**
  - Returns an array of the URL-decoded values of *all* occurrences of name in query string
  - Returns a one-element array if param not repeated
  - Returns null if no such parameter is in query
- **request.getParameterNames() or request.getParameterMap()**
  - Returns Enumeration or Map of request params
  - Usually reserved for debugging

An HTML Form With Three Parameters

```
<FORM ACTION="three-params">
  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>
```

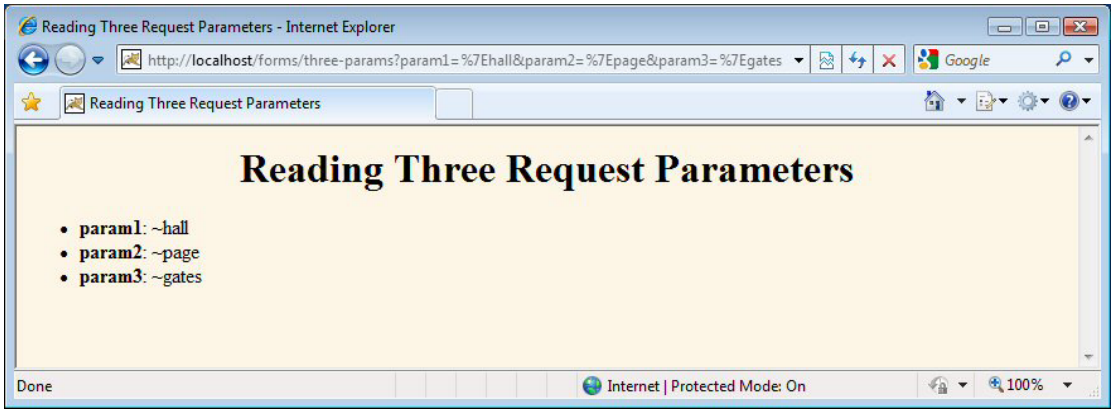


- Project name is "forms"
- Form installed in WebContent/three-params-form.html

Reading the Three Parameters

```
@WebServlet("/three-params")
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    ...
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>"+title + "</TITLE></HEAD>\n" +
                "<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>");
  }
}
```


Reading Three Parameters: Result



Reading All Parameters

```
@WebServlet("/show-params")
public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String docType =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
    "Transitional//EN\">\n";
    String title = "Reading All Request Parameters";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>"+title + "</TITLE></HEAD>\n"+
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<TABLE BORDER=1 ALIGN=CENTER>\n" +
                "<TR BGCOLOR=\"#FFAD00\">\n" +
                "<TH>Parameter Name<TH>Parameter Value(s)");
```

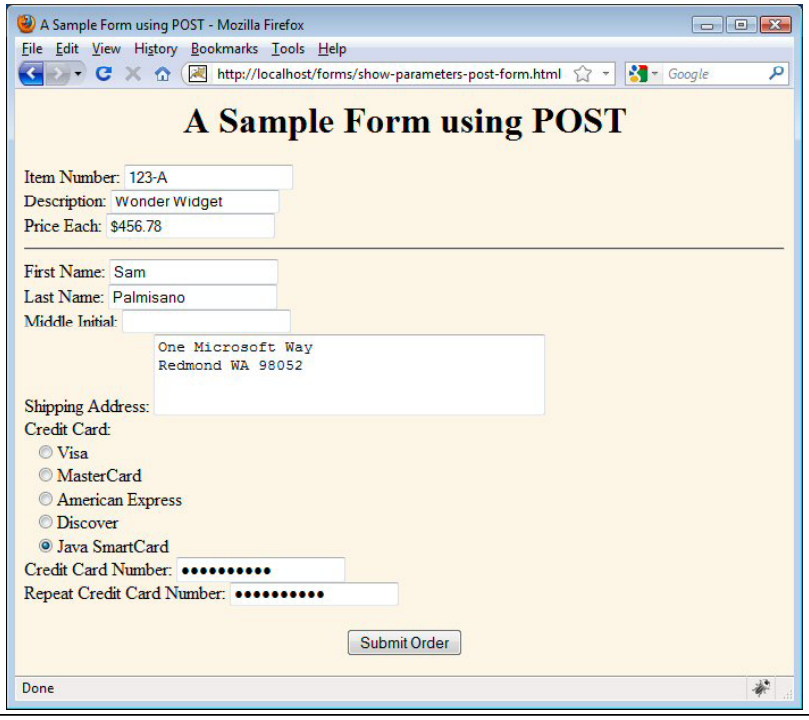

Reading All Parameters (Continued)

```
    Enumeration<String> 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>");
        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>");
  }
```

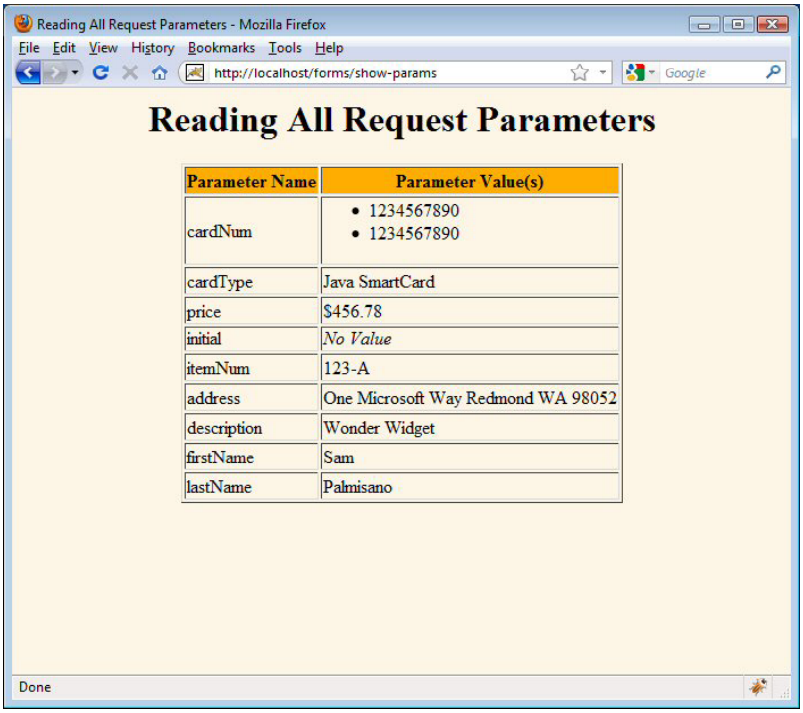
Reading All Parameters (Continued)

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


Reading All Parameters (Sample Form)



Reading All Parameters (Result)



Handling Missing and Malformed Data

Checking for Missing and Malformed Data

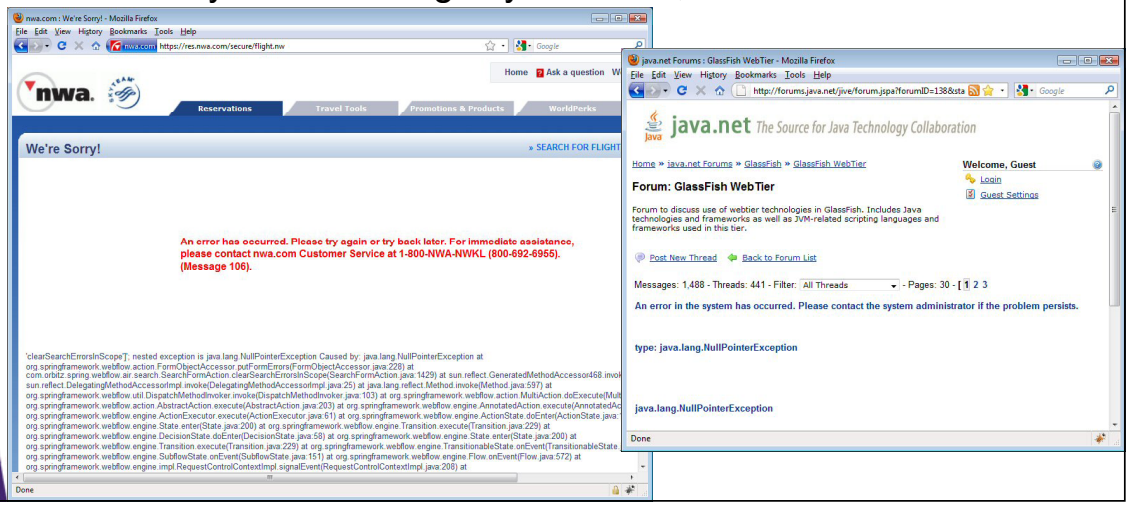
- **Missing**
  - Field missing in form
    - getParameter returns null
  - Field blank when form submitted
    - getParameter returns an empty string (or possibly a string with whitespace in it)
  - Must check for null before checking for empty string

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

- **Malformed**
  - Value is a nonempty string in the wrong format

Checking for Missing and Malformed Data

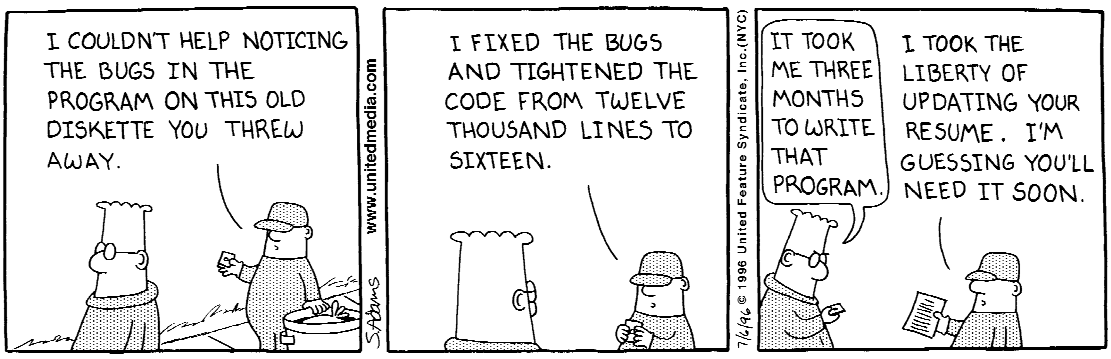
- **Principles**
  - Assume user data could be missing or in wrong format
  - Users should *never* see Java error messages
    - Only error messages you create, aimed at end users



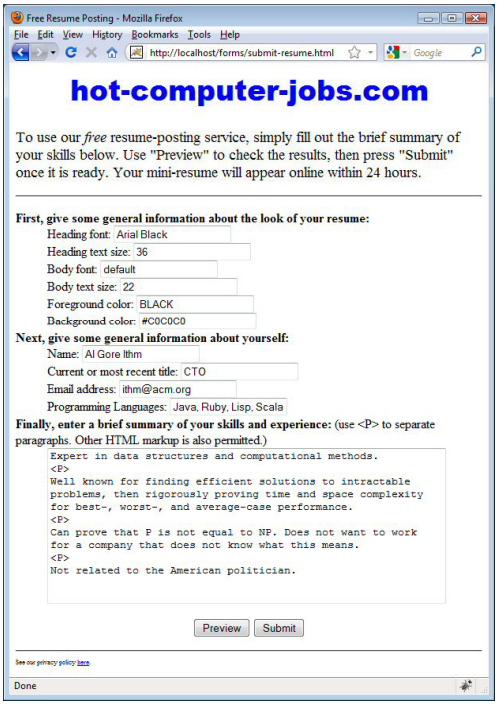
Handling Missing and Malformed Data

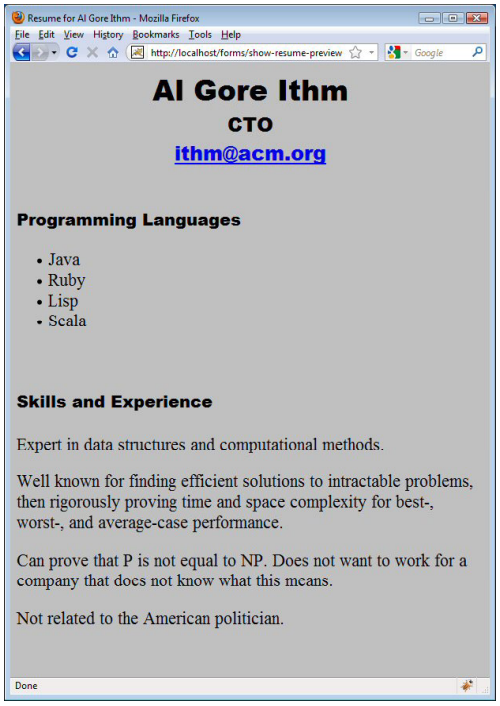
- **Use default values**
  - Replace missing values with application-specific standard values
  - See following example
- **Redisplay the form**
  - Show the form again, with missing values flagged
  - Previously-entered values should be preserved
  - Best option for implementing this: use framework like JSF or Struts
    - Covered in later tutorials
  - Four options to implement this directly
    - Have the same servlet present the form, process the data, and present the results.
      - See book for example
    - Have one servlet present the form; have a second servlet process the data and present the results.
    - Have a JSP page "manually" present the form; have a servlet or JSP page process the data and present the results.
    - 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.

Example of Using Default Values: A Résumé-Posting Site



Résumé-Posting Site: Input Form and Good Data Results





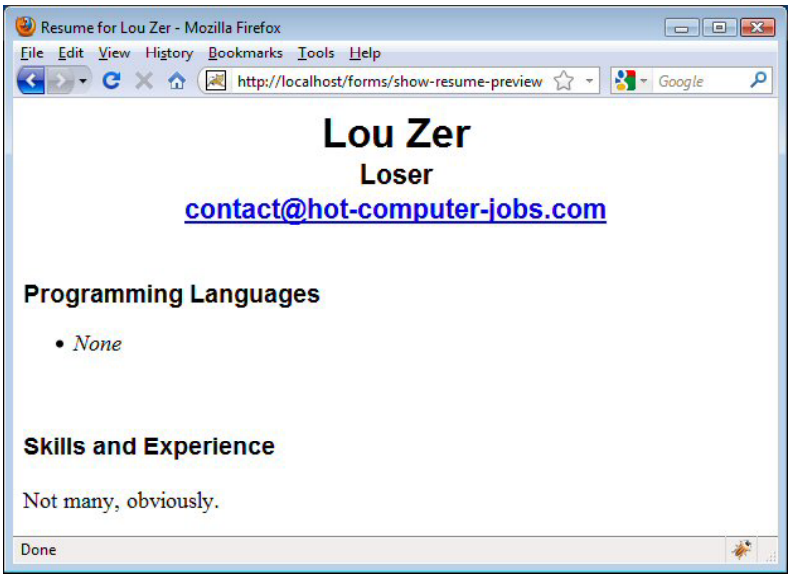
Résumé-Posting Site: Servlet Code

```
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");
```

Résumé-Posting Site: Servlet Code (Continued)

```
private String replaceIfMissing(String orig,
                                String replacement) {
  if ((orig == null) || (orig.trim().equals(""))) {
    return(replacement);
  } else {
    return(orig);
  }
}
```


Résumé-Posting Site: Result for Incomplete Data



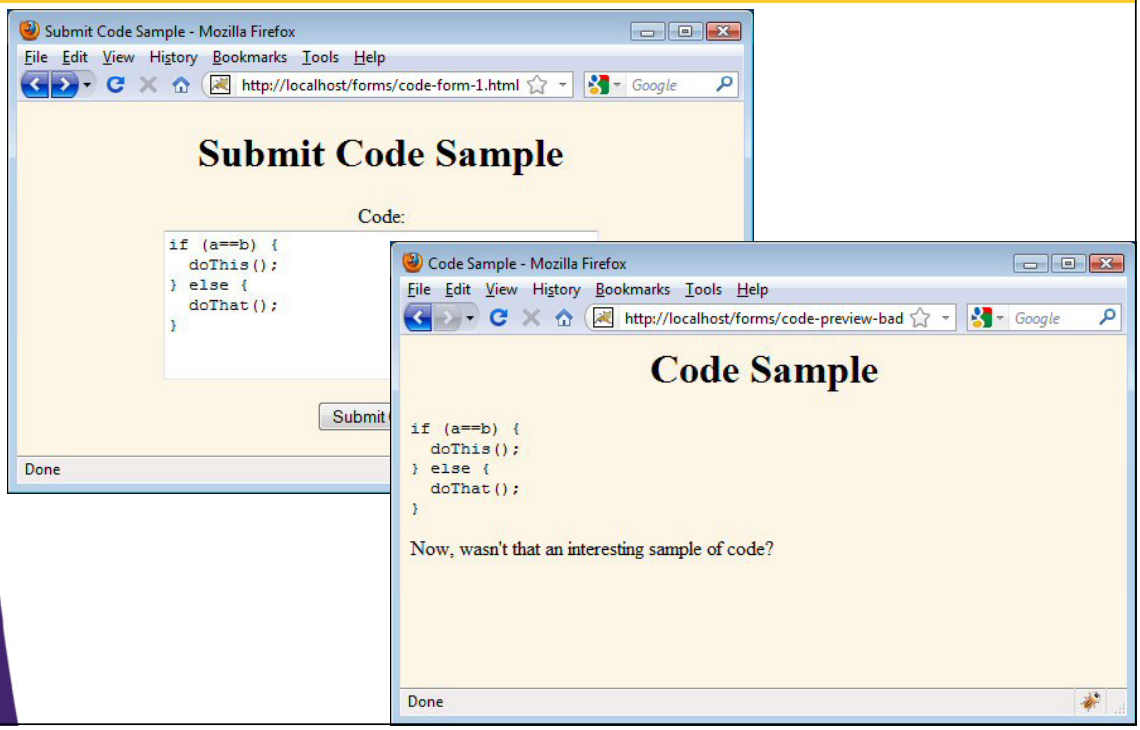
Filtering Strings for HTML-Specific Characters (Code)

```
public class ServletUtilities {
  public static String filter(String input) {
    if (!hasSpecialChars(input)) {
      return(input);
    }
    StringBuilder filtered =
      new StringBuilder(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
      c = input.charAt(i);
      switch(c) {
        case '<': filtered.append("&lt;"); break;
        case '>': filtered.append("&gt;"); break;
        case '"': filtered.append("&quot;"); break;
        case '&': filtered.append("&amp;"); break;
        default: filtered.append(c);
      }
    }
    return(filtered.toString());
  }
  …
```

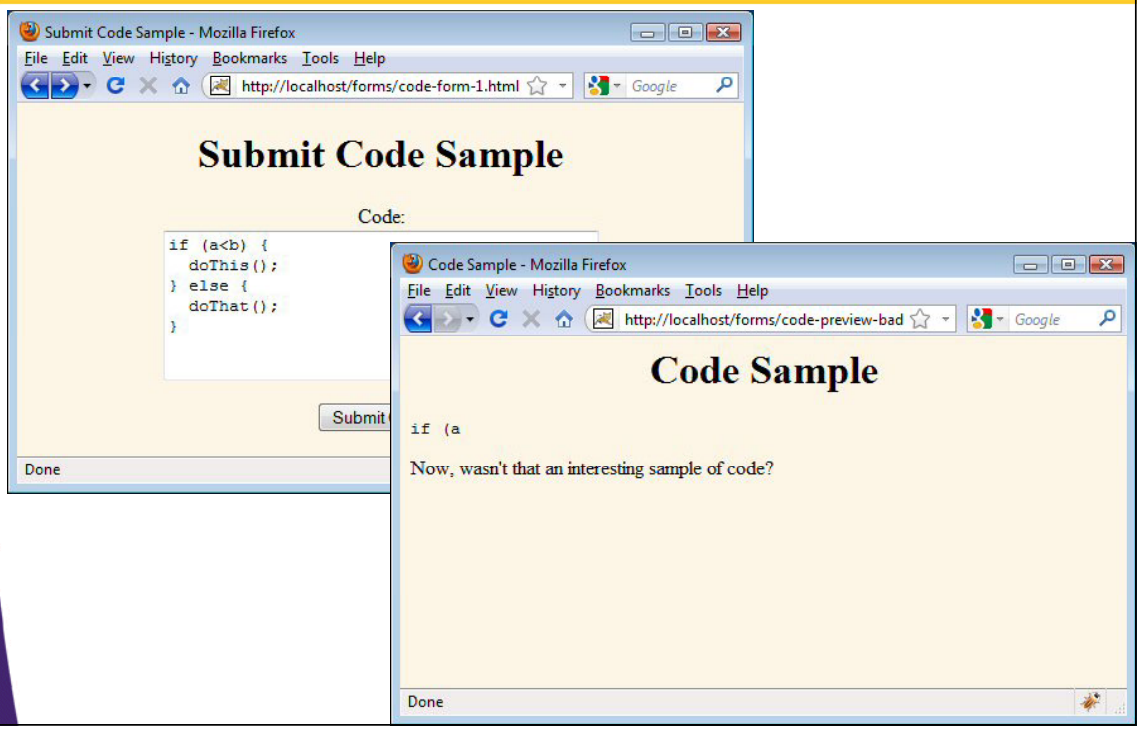

A Servlet that Displays Code Samples: No Filtering

```
@WebServlet("/code-preview-bad")
public class CodePreviewBad extends HttpServlet {
  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    …
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>"+title+"</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n"+
                "<PRE>\n" +
                getCode(request) +
                "</PRE>\n" +
                "Now, wasn't that an interesting sample\n" +
                "of code?\n" +
                "</BODY></HTML>");
  }
  protected String getCode(HttpServletRequest request) {
    return(request.getParameter("code"));
  }
}
```

A Servlet that Displays Code Samples: No Special Chars



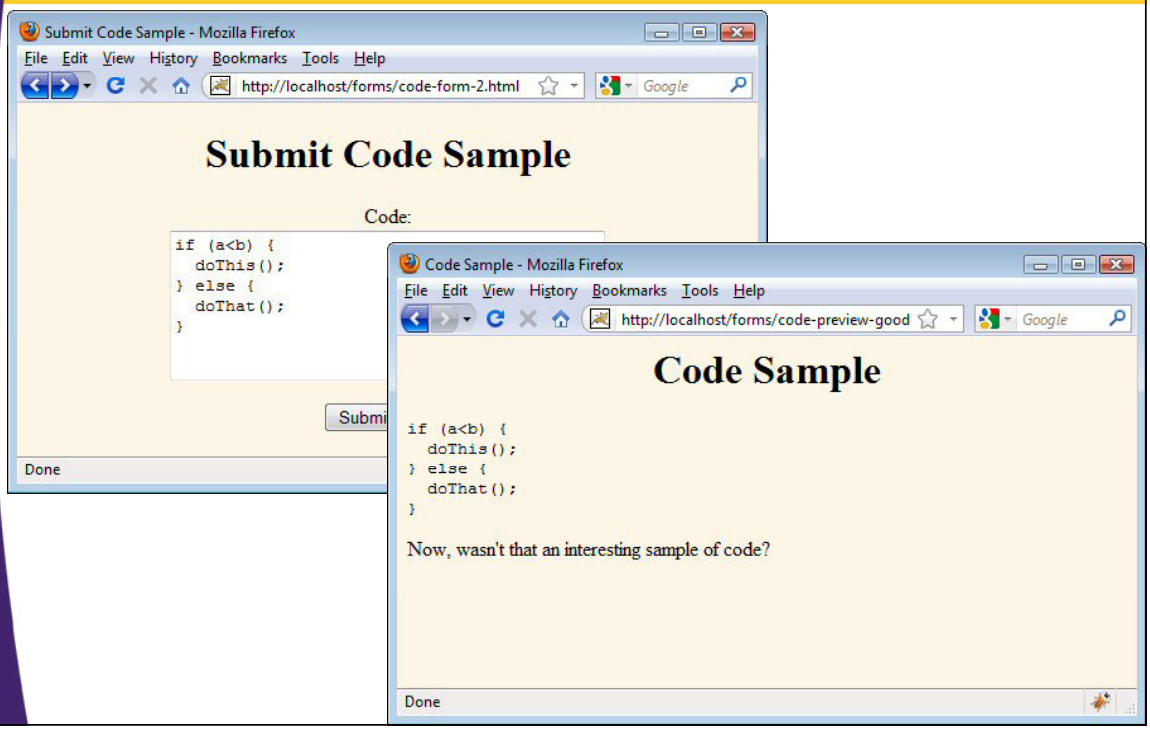
A Servlet that Displays Code Samples: Special Chars



A Servlet that Displays Code Samples: Filtering

```
@WebServlet("/code-preview-good")
public class CodePreviewGood extends CodePreviewBad {
  protected String getCode(HttpServletRequest request) {
    return
      (ServletUtilities.filter(super.getCode(request)));
  }
}
```


Fixed Servlet that Displays Code Samples: Special Chars



Advanced Topics

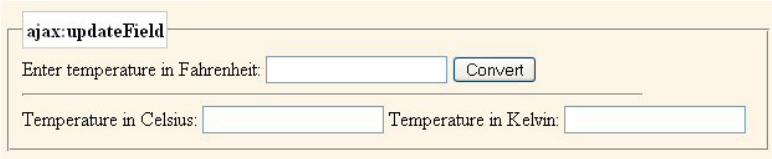
Tags for Form Aesthetics

- **label**
  - If you use the label tag for prompts associated with fields, clicking on the label transfers focus to the input field
  - You can either use the "for" attribute or enclose the field within the label
    - <label for="fname">First name:</label>
      <input type="text" name="userFirstName" id="fname"/>
    - <label>First name:
      <input type="text" name="userFirstName"
      </label>
- **fieldset and legend**
  - Grouping all or part of a form inside fieldset draws attention to it and separates it from the rest of the page
  - Using style sheets for the legend is particularly useful

Tags for Form Aesthetics: Example

- **HTML**

```
<fieldset>
  <legend>ajax:updateField</legend>
  <form ...>
    <label for="f">Enter temperature in Fahrenheit:</label>
    <input type="text" id="f"/>
    <input type="button" id="convertButton" value="Convert"/>
    <hr width="500" align="left"/>
    <label for="c">Temperature in Celsius:</label>
    <input type="text" id="c"/>
    <label for="k">Temperature in Kelvin:</label>
    <input type="text" id="k"/>
  </form>
</fieldset>
```



- **CSS**

```
legend {
  font-weight: bold;
  color: black;
  background-color: white;
  border: 1px solid #cccccc;
  padding: 4px 2px;
}
```


Handling Input in Multiple Languages

- **Use server's default character set**

```
String firstName =
  request.getParameter("firstName");
```

- **Convert from English (Latin-1) to Japanese**

```
String firstNameWrongEncoding =
  request.getParameter("firstName");
String firstName =
  new String(firstNameWrongEncoding.getBytes(),
             "Shift_JIS");
```

- **Accept either English or Japanese**

```
request.setCharacterEncoding("JISAutoDetect");
String firstName =
  request.getParameter("firstName");
```

Reading Raw Form Data and Parsing Uploaded Files

- **Raw data**
  - request.getReader
  - request.getInputStream
    - Data no longer available via getParameter after this
- **Parsing uploaded files**
  - HTML has a way of submitting entire files
    - <INPUT TYPE="FILE"…>
      - See Section 19.7 of 2nd Edition of *Core Servlets and JSP*.
  - Servlet/JSP APIs have no builtin way to parse files
  - Popular third-party library available from the Apache/Jakarta "Commons" library
    - http://jakarta.apache.org/commons/fileupload/

More Advanced Topics (See book for details/examples)

- **Automatically filling in a data object with request parameter values**
  - Use a JavaBean (Java object with methods named getBlah and setBlah) to store incoming data
  - Request param named blah passed to appropriate setBlah method. Automatic type conversion. Default values used for errors.
- **Redisplaying the input form when parameters are missing or malformed**
  - The same servlet presents the form, processes the data, and presents the results. JSF provides better solution – see later tutorials!
    - 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.

Summary

- **Make a form: <form …> … </form>**
  - Relative URL for "action". Textfields need "name". Should always have submit button.
- **Read data: request.getParameter("name")**
  - Results in value as entered into form, not necessarily as sent over network. I.e., *not* URL-encoded.
- **Check for missing or malformed data**
  - Missing: null or empty string
  - Special case: query data that contains special HTML characters
    - Need to be filtered if query data will be placed into resultant HTML page

Questions?