Server-side Web Development and Programming

Lecture 7:
Input Validation and Error Handling

Form Validation

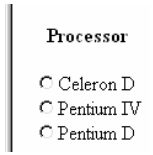
- Detecting user error
  - Invalid form information
  - Inconsistencies of forms to other entities
    - Enter ID not in database, etc.
- Correcting user error
  - Providing information or how to correct error
  - Reducing user memory load
- Preventing user error
  - Good instructions
  - Field types/values that prevent error
  - Error tolerance
    - Example: Accepting phone numbers in multiple formats

What to Validate

- Required fields have input
  - Text inputs non-empty
    - Trim method useful to remove leading, training spaces

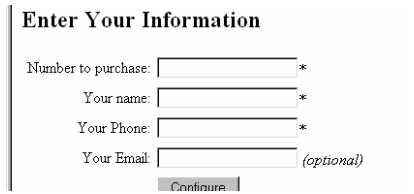
```
String name =
        (request.getParameter("name")).trim();
if (name.equals("")) { …
```

  - Radio button groups and lists have selection where required



Error Prevention

- Tell user what is required, optional

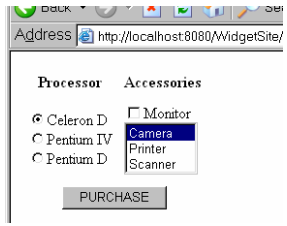


- Set default values where appropriate
  - CHECKED attribute for radio buttons

```
<input type="radio" name="Processor"
       value="Celeron D" CHECKED/>
```

  - SELECTED attribute for lists

```
<option value="camera" SELECTED/>
```



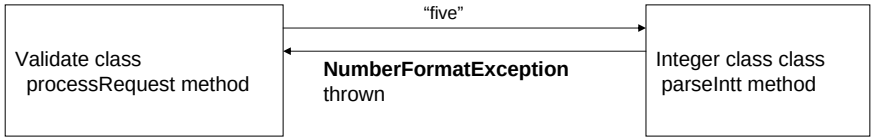
Validating Numeric Inputs

- What if user enters non-numeric value?


```
– String quantity = request.getParameter("quantity");
– int quantityNumber = Integer.parseInt(quantity);
```

Cannot parse “five”

- Exception thrown in Java



Validating Numeric Inputs

- Unhandled exceptions cause error screen

- Must handle with **try/catch** block

```
try {
     code which might cause exception
     …
     }
  catch (ExceptionType variable) {
     code to handle exception
     }
code after block
```

Jump here if exception

Skip if no exception

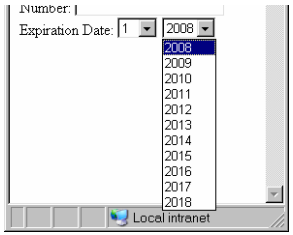
Usually forward to error page

Validating Numeric Inputs

```
// Parse quantity entered (exception if not an integer)
int quantityNumber = 0;
try {
    quantityNumber = Integer.parseInt(quantity);
}
catch (NumberFormatException ex) { // Can't parse quantity
     url = "/error.jsp";
}
```

Numeric Error Prevention

- Avoid direct numeric input if possible
- Provide dropdowns that list values if possible

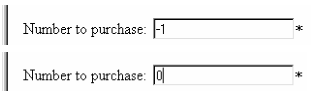


- Can use JSP to automate
  - Use loop to generate values

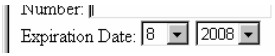
```
26  <select name="ExpirationYear">
27      <% for (int year = 2008; year <= 2018; year++) { %>
28      <option value="<%= year%>"><%= year %></option>
29      <% } %>
30  </select>
```

Validating Input

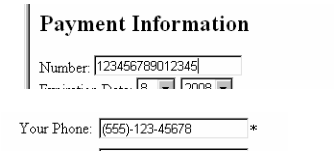
- Is numeric input valid?
  - Negative quantity should be detected
  - What about quantity of 0?



- Is combination of choices legal?

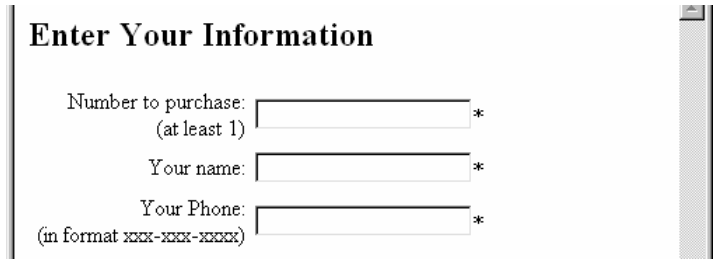


- Is format of input legal?
  - Credit card number 16 digits
  - Phone number in correct format



Error Prevention

- Tell user if format or other rules apply



Regular Expressions

- Tool for verifying an input string is in a given format
  - Easier than parsing it yourself!
- Examples:
  - Credit card contains 16 digits
  - Phone number in form (*3 digits*) *3 digits – 4 digts*
  - Email in form *characters@characters.characters*
- Note that correct format ≠ legal
  - Nonexistent phone number, etc.
  - Will need to verify against database

Regular Expressions

- Key idea:
  Wildcard characters match characters of a certain type

.	Matches any character
\\d	Matches any digit 0-9
\\D	Matches any non-digit
\\w	Matches “word” character a-z, A-Z, 0-9
\\W	Matches any non-“word” character
\\s	Matches any “space” character (, tab, return)
\\S	Matches any non-“space” character

  - Note: the extra “\” in front is required by Java

Regular Expressions

- Quantifiers give number of times a character must appear

*	Any number of times (including 0)
+	At least once
{*number*}	Exactly *number* times

- Examples:
  - Credit card number: **\\d{16}**
  - Phone number: **\\d{3}-\\d{3}-\\d{4}**
  - Email address: **\\w+@\\w+(\.\\w+)***

Regular Expressions

- Java syntax:
  - Create Pattern object from regular expression
  - Create Matcher object using matcher method of Pattern and the actual input to match with
  - Use matches method of the Matcher object to determine whether match exists

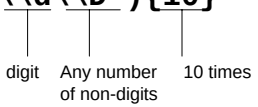
```
Pattern patternObject =
      Pattern.compile("regular expression");
Matcher matcherObject =
      patternObject.matcher(string to match with);
if (!matcherObject.matches()) {
      code to handle failed match
      }
```

Regular Expressions

```
44        Pattern creditCardPattern = Pattern.compile("\\d{16}");
45        Matcher creditCardMatcher = creditCardPattern.matcher(creditCardNumber);
46        if (!creditCardMatcher.matches()) {
47            System.out.println ("Credit card error");
48            url = "/error.jsp";
49        }
50        Pattern phonePattern = Pattern.compile("\\d{3}-\\d{3}-\\d{4}");
51        Matcher phoneMatcher = phonePattern.matcher(phone);
52        if (!phoneMatcher.matches()) {
53            System.out.println ("Phone error");
54            url = "/error.jsp";
55        }
```

Error Tolerance

- Should not reject based on format if any chance input valid
  - Example: other legal phone numbers
    - 555-555-5555
    - (555) 555-5555
    - 555.555.5555
    - …
- Choose most tolerant pattern to prevent false rejection
  - "A phone number is 10 digits separated by any number of non-digits"
  - Pattern: **(\\d\\D*){10}**



Calendar Dates in Java

- Construct a new **`GregorianCalendar`** object
  - Contains information about current date when created
  - Must import java.util.* library
- Use **`get(Calendar.fieldname)`** method to get component of that date
  - Field names = **`YEAR`**, **`MONTH`**, etc.
  - Returns an integer

```
// Get a current time object and extract the current year and month
GregorianCalendar calendar = new GregorianCalendar();
int yearNow = calendar.get(Calendar.YEAR);
int monthNow = calendar.get(Calendar.MONTH);
```

Calendar Dates in Java

- Can use to validate things about dates entered by user

```
// If the year selected is this year and the month selected is before
// thi smonth, then the card has expired.
if (Integer.parseInt(creditCardYear)== yearNow &&
    Integer.parseInt(creditCardMonth) < monthNow) {
    request.setAttribute("cardDateError", "Your card has expired!");
    url = "/information.jsp";
}
```

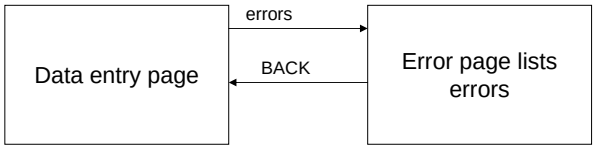
- Caution:
  - Date for user may be different from server
    - Inaccurate clocks
    - International date boundary
  - Safest to only use for month, year.

Error Messages

- Give user information necessary to correct error
  - Bad: “Invalid quantity”
  - Good: “Quantity must be a numeric value greater than zero”
  - Better: “You must give a quantity” *or*
    “Quantity must be a number” *or*
    “Quantity must be at least 1”
    Depending on the specific problem

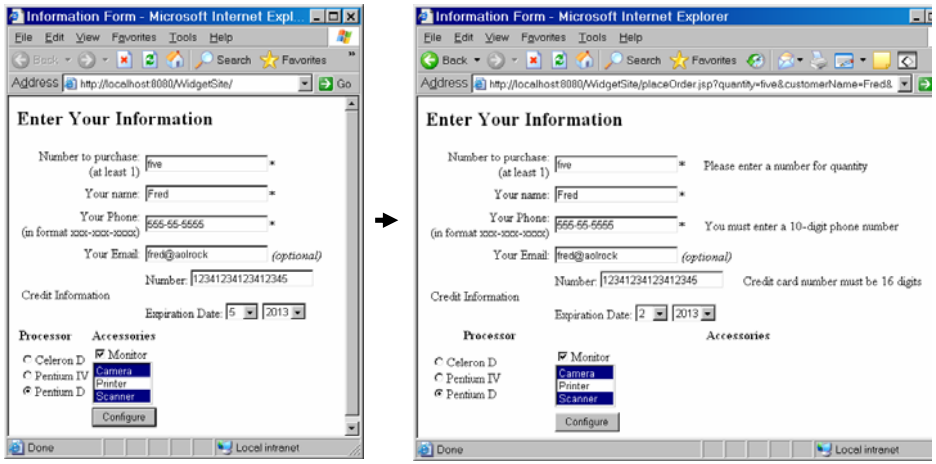
Error Pages

- Put error message next to source of error
  - Allows user to see where correction is needed
- Echo back inputs user provided
  - User can see error they made
  - No need to reenter correct values
  - Goal: reduced memory load



User will have forgotten what errors were listed!

Error Pages



Echoing Values in Text Input

- Get value from request
- Use to set VALUE attribute of text element

```
<%
  String customerName =
     request.getParameter("customerName");
%>
…
Name: <input type = "text"
            name = "customerName";
            value = "<%= customerName %>"
>
```

Echoing Values in Checkboxes

- Determine whether checked on requesting page by comparing to null
- If so, insert CHECKED into the tag

```
<%
  String monitor =
       request.getParameter("monitor");
%>
  …
<input type = "checkbox"
       name = "monitor"
       <% if (monitor != null) { %> checked <% } %>
       >Monitor
```

Echoing Values in Radio Buttons

- Determine if checked on requesting page by comparing to its value
  - May need to check whether null to prevent error
  - Set value to "" or some default value
- If so, insert CHECKED into the tag

```
<% String processor = request.getParameter("processor");
   if (processor == null) processor = "Celeron D"; %>
   …
<input type = "radio" name = "processor" value = "Celeron D"
  <% if (processor.equals("Celeron D") { %> checked <% } %>
      >Celeron D
<input type = "radio" name = "processor" value = "Pentium IV"
   <% if (processor.equals("Pentium IV") { %> checked <% } %>
      >Pentium IV
<input type = "radio" name = "processor" value = "Pentium D"
   <% if (processor.equals("Pentium D") { %> checked <% } %>
      >Pentium D
```

Echoing Values in Lists

- Determine if option selected on requesting page by comparing to its value
  - May need to check whether null to prevent error
- If so, insert SELECTED into the OPTION tag

```
<% String cardYear = request.getParameter("ExpirationYear");
    if (cardYear == null) cardYear = "2008" %>
      …
<select name = "ExpirationYear">
   <% for (int year = 2008; year < 2018; year++ %>
      <option value = "<%= year %>"
          <% if (cardYear.equals(year)) %> selected <% } %>
          ><%= year %>
  <% } %>
</select>
```

Echoing Values in Multiple Lists

- Must use getParameterValues to get array of options selected
- For each option, must search array to find whether its value is in the array
- Much easier if create simple search function first

```
<%!
    public boolean find(String[] list, String target) {
       if (list == null) {return false;}
       for (int i = 0; i < list.length; i++) {
          if (target.equals(list[i])) {return true;}
          }
       return false;
       }
%>
```

Note: syntax of creating function in JSP

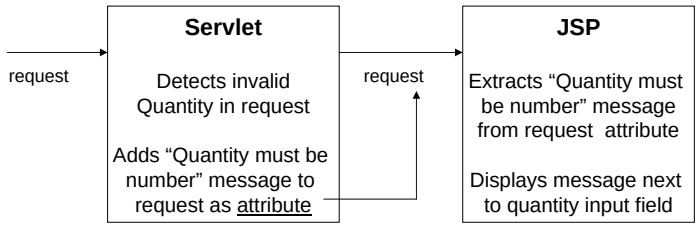
Echoing Values in Multiple Lists

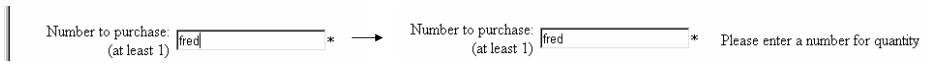
- Call the find function within each option in list
  - If returns true, insert SELECTED (will highlight all selected)

```
<select name="peripherals" size="3" multiple>
   <option value="Camera"
      <% if (find(peripherals, "Camera")) { %> SELECTED <% } %>
            >Camera
   <option value="Printer"
      <% if (find(peripherals, "Printer")) { %> SELECTED <% } %>
            >Printer
   <option value="Scanner"
      <% if (find(peripherals, "Scanner")) { %> SELECTED <% } %>
            >Scanner
     </select>
```

Displaying Error Messages

- Bad approach:
  Force JSP to repeat validation done in servlet to determine which messages to display
- Better approach:
  Once servlet detects error, it creates error message and passes to JSP as attribute





Creating Error Messages in Servlet

- `if (error condition) {`
  `request.setAttribute(errorAttributeName,`
  `message to display);`

```
// Get the parameter values from the request
String name = (request.getParameter("customerName")).trim();
String email = (request.getParameter("customerEmail")).trim();
String phone = (request.getParameter("customerPhone")).trim();
String quantity = (request.getParameter("quantity")).trim();
String creditCardNumber = (request.getParameter("cardNumber")).trim();
String creditCardYear = (request.getParameter("ExpirationYear")).trim();
String creditCardMonth = (request.getParameter("ExpirationMonth")).trim

// If any are empty, set the url to forward to to the error page.
// Otherwise, forward to the normal reciept
if (name.equals("")) {
    url = "/error.jsp";
    request.setAttribute("nameError", "You must enter a name");
}
```

Creating Error Messages in Servlets

- Can use several conditions to create detailed messages

```
// This is a nested structure which determines the type of error
// replated to the quantity, and sets the appropriate error message.
int quantityNumber = 0;
if (quantity.equals("")) {
    url = "/error.jsp";
    request.setAttribute("quantityError", "You must enter a quantity");
}
else {
  //Parse quantity entered (exception if not an integer)
  try {
      quantityNumber = Integer.parseInt(quantity);
     if (quantityNumber < 1) {
         request.setAttribute("quantityError", "Quantity must be at least 1");
      }
  }
  catch (NumberFormatException ex) { // Can't parse quantity
      request.setAttribute("quantityError", "Please enter a number for quantity");
      url = "/error.jsp";
  }
}
```

Displaying Error Messages in JSP

- Get attribute value from request
- If no error, will have value NULL
  - Set value to empty string to avoid strange output
- Display the value next to the appropriate field

```
<% String errorAttributeValue =
   (String)request.getAttribute("errorAttributeName");
    if (errorAttributeValue == null)
                        errorAttributeValue = ""; %>
…
<someInputField …> <%= errorAttributeValue>
```

Field where error occurred

Message describing error (or nothing if no error)

Displaying Error Messages in JSP

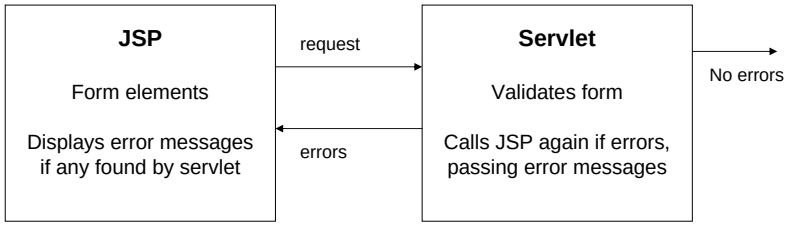
```
41  String quantity = request.getParameter("quantity");
42  if (quantity == null) quantity = "1";
43  String quantityError = (String)request.getAttribute("quantityError");
44  if (quantityError == null) quantityError = "";
```

```
94  <tr>
95    <td align="right">Number to purchase:
96                      <br/>(at least 1)</td>
97    <td>
98      <input type="text" name="quantity" value="<%= quantity %>">*   
99      <%= quantityError %>
100   </td>
101 </tr>
```

Number to purchase: (at least 1) fred * Please enter a number for quantity

Single Input/Error Page

- Bad design: Having separate pages to get initial input, echo back for errors
  - Changes to form have to be made to both pages
- Better design: single page for both



Single Input/Error Page

- If first time page called, must insert default values instead of previous values
  - Check whether previous value null

```
<% fieldValue = request.getParameter("fieldName");
   if (fieldValue == null) fieldValue = defaultValue;
  %>
```

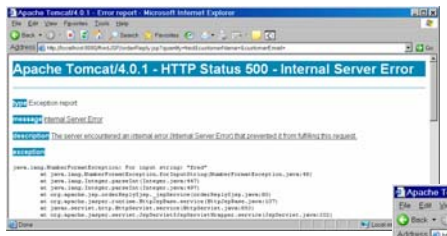
```
<input type="text" name="fieldname"
       value= "<%= fieldValue %>" >
```

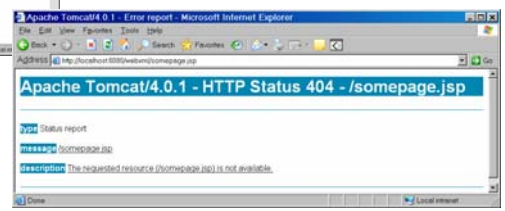
Single Input/Error Page

```
41    String quantity = request.getParameter("quantity");
42    if (quantity == null) quantity = "1";
43    String quantityError = (String)request.getAttribute("quantityError");
44    if (quantityError == null) quantityError = "";
45
46    String name = request.getParameter("customerName");
47    if (name == null) name = "";
48    String nameError = (String)request.getAttribute("nameError");
49    if (nameError == null) nameError = "";
50
51    String phone = request.getParameter("customerPhone");
52    if (phone == null) phone = "";
53    String phoneError = (String)request.getAttribute("phoneError");
54    if (phoneError == null) phoneError = "";
55
56    String email = request.getParameter("customerEmail");
57    if (email == null) email = "";
58
59    String cardNumber = request.getParameter("cardNumber");
60    if (cardNumber == null) cardNumber = "";
61    String cardNumberError = (String)request.getAttribute("cardNumberError");
62    if (cardNumberError == null) cardNumberError = "";
63
64    String monthSelected = request.getParameter("ExpirationMonth");
65    if (monthSelected == null) monthSelected = "1";
66    String yearSelected = request.getParameter("ExpirationYear");
67    if (yearSelected == null) yearSelected = "2008";
68    String cardDateError = (String)request.getAttribute("cardDateError");
69    if (cardDateError == null) cardDateError = "";
70
71    String processor = request.getParameter("processor");
72    if (processor == null) processor = "Celeron D";
73    String processorError = (String)request.getAttribute("processorError");
74    if (processorError == null) processorError = "";
```

Last Resort Error Handling

- User should never see Tomcat-generated error page!
  - Reduces confidence in your entire site
  - Confuses user (did they do something wrong?)



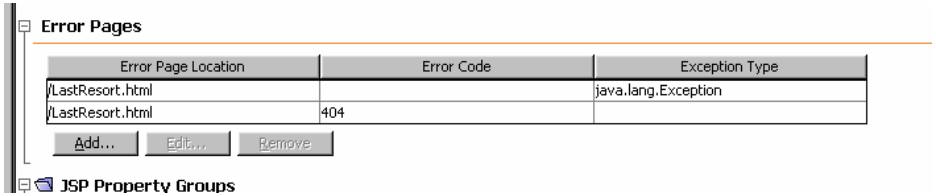


Last Resort Error Handling

- "Last Resort" error page
  - Called if unhandled error
  - Should contain:
    - Identifiable company **logo** and **design** so the user can be sure that they are still on your site
    - Main **navigation bar** which offers the user a way to try something else
    - A reassuring **message** telling this is not user's fault
    - A link to **email** the webmaster to inform them of the problem

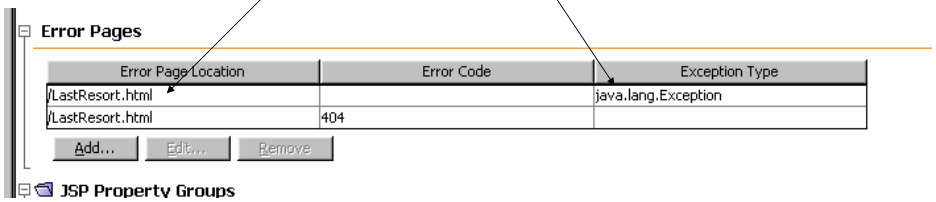
Default Error Pages

- Can specify default page for:
  - Unhandled exceptions (such as `NumberFormatExceptions`)
  - Missing pages and other server-related errors
- Done in web.xml file
  - Error pages under pages tab



Default Exception Handling

- Specify page to jump to and type of exception

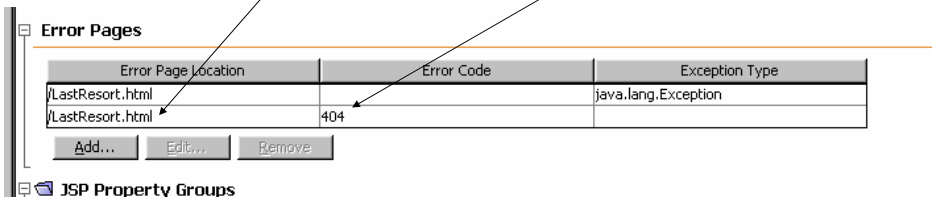


  - Must use full name of class (including *library.package.classname*)
  - Can use base class **`java.lang.Exception`** to catch everything

- If this type of exception occurs and is not handled inside a try/catch, jump to this page

Handling Missing Pages

- Unavoidable in complex web sites with multiple developers
- Causes error code **404**
- Specify page to jump to and error code



- If this error code occurs within, jump to this page