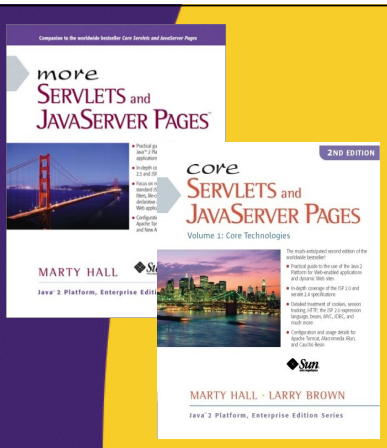
JSF: Validating User Input

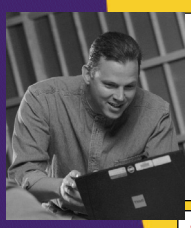
Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See http://www.coreservlets.com/JSF-Tutorial/jsf2/.

© 2012 Marty Hall



For live training on JSF 1 or 2, please see courses at http://courses.coreservlets.com/.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 6 or 7 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, Hadoop, SOAP-based and RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Manual validation**
- **Implicit automatic validation**
- **Explicit validation**
- **Defining your own validation methods**
- **Creating custom validators**

The Need for Validation

- **Two tasks that almost every Web application needs to perform:**
  - Checking that all required form fields are present and in the proper format
  - Redisplaying the form when values are missing or malformed
    - With error messages showing what the problem was
    - With OK values maintained
- **This is extremely cumbersome with standard servlet/JSP technology**
  - Even with the JSP 2.0 expression language
  - This is a (the?) major weakness in servlet/JSP technology

Validation Approaches

- **Manual validation**
  - Use string properties for bean
  - Do validation in setter methods and/or action controller
  - Return null to redisplay form
  - Create custom error messages
- **Implicit automatic validation**
  - Use int, double, etc. bean properties. Or add `required`.
  - System redisplays form if there is conversion error
  - Use h:message to display field-specific error message
- **Explicit automatic validation**
  - Use f:convertNumber, f:convertDateTime, f:validateLength, f:validateDoubleRange, or f:validateLongRange
  - System redisplays form if failure; use h:message again
- **Custom validation methods**
  - Create FacesMessage, wrap in ValidatorException

Validation in the Action Controller

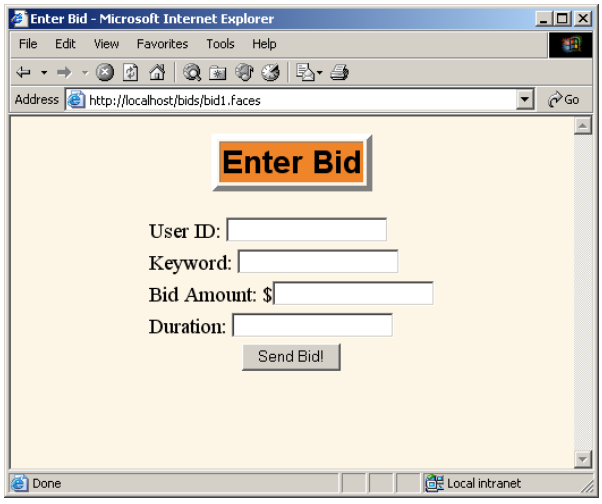
For when validation is closely tied to your business logic, or when you need to check complex interactions among fields

Manual Validation

- **Setter methods convert from strings**
  - Use try/catch blocks
  - Use application-specific logic
- **Action controller checks values**
  - If values are OK
    - Returns normal outcomes
  - If values are missing or illegal
    - Stores error messages in bean
    - Returns null
- **Input Form**
  - Displays error messages
    - Error message should be empty string by default
    - In h:outputText, use escape="false" if error messages contain HTML tags

Manual Validation: Example

- **Idea**
  - Collect bids for keywords at search engine site
- **Attributes**
  - UserID
    - Cannot be missing
  - Keyword
    - Cannot be missing
  - Bid amount
    - Must be legal double
    - Must be at least 0.10
  - Bid duration
    - Must be legal int
    - Must be at least 15



Bean Code: String Properties (No Conversion)

```
package coreservlets;
import java.util.*;

public class BidBean1 {
  private String userID = "";
  private String keyword = "";
  private String bidAmount;
  private double numericBidAmount = 0;
  private String bidDuration;
  private int numericBidDuration = 0;
  private List<String> errorMessages;

  public String getUserID() { return(userID); }

  public void setUserID(String userID) {
    this.userID = userID.trim();
  }

  public String getKeyword() { return(keyword); }

  public void setKeyword(String keyword) {
    this.keyword = keyword.trim();
  }
```


Bean Code: Numeric Properties (Conversion)

```
  public String getBidAmount() { return(bidAmount); }

  public void setBidAmount(String bidAmount) {
    this.bidAmount = bidAmount;
    try {
      numericBidAmount = Double.parseDouble(bidAmount);
    } catch(NumberFormatException nfe) {}
  }
  public double getNumericBidAmount() {
    return(numericBidAmount);
  }

  public String getBidDuration() { return(bidDuration); }

  public void setBidDuration(String bidDuration) {
    this.bidDuration = bidDuration;
    try {
      numericBidDuration = Integer.parseInt(bidDuration);
    } catch(NumberFormatException nfe) {}
  }
  public int getNumericBidDuration() {
    return(numericBidDuration);
  }
```

Bean Code: Action Controller

```
public String doBid() {
  errorMessages = new ArrayList<String>();
  if (getUserID().equals("")) {
    errorMessages.add("UserID required");
  }
  if (getKeyword().equals("")) {
    errorMessages.add("Keyword required");
  }
  if (getNumericBidAmount() <= 0.10) {
    errorMessages.add("Bid amount must be at least $0.10.");
  }
  if (getNumericBidDuration() < 15) {
    errorMessages.add("Duration must be at least 15 days.");
  }
  if (errorMessages.size() > 0) {
    return(null);
  } else {
    return("success");
  }
}
```

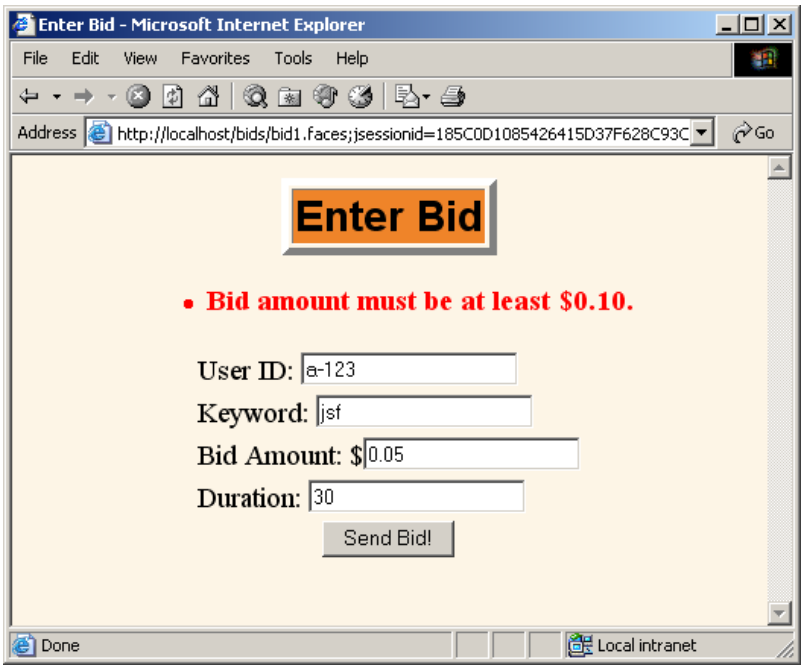

Bean Code: Error Messages Property

```
public String getErrorMessages() {
  String messageList;
  if ((errorMessages == null) ||
      (errorMessages.size() == 0)) {
    messageList = "";
  } else {
    messageList = "<FONT COLOR=RED><B><UL>\n";
    for(String message: errorMessages) {
      messageList = messageList + "<LI>" + message + "\n";
    }
    messageList = messageList + "</UL></B></FONT>\n";
  }
  return(messageList);
}
```

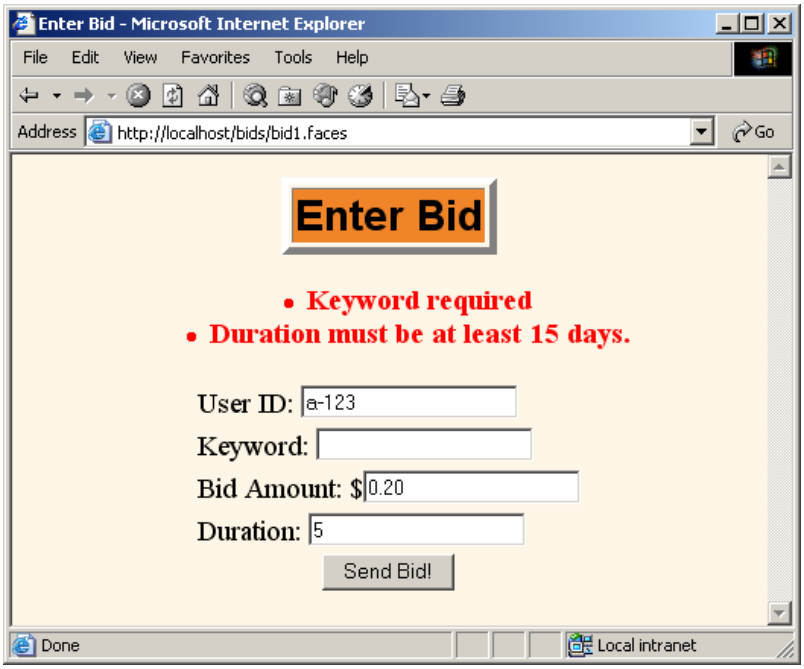
Input Form: Outputting Error Messages (bid1.jsp)

```
<h:form>
  <h:outputText value="#{bidBean1.errorMessages}"
                escape="false"/>
  <TABLE>
  <TR>
    <TD>User ID:
    <h:inputText value="#{bidBean1.userID}"/></TD></TR>
  <TR>
    <TD>Keyword:
    <h:inputText value="#{bidBean1.keyword}"/></TD></TR>
  <TR>
    <TD>Bid Amount:
    $<h:inputText value="#{bidBean1.bidAmount}"/></TD></TR>
  <TR>
    <TD>Duration:
    <h:inputText value="#{bidBean1.bidDuration}"/></TD></TR>
  <TR><TH>
      <h:commandButton value="Send Bid!"
                       action="#{bidBean1.doBid}"/></TH></TR>
  </TABLE>
</h:form>
```


Manual Validation: Results



Manual Validation: Results



Manual Validation: Results

Alternative to Building Own Error Messages (See Later Section)

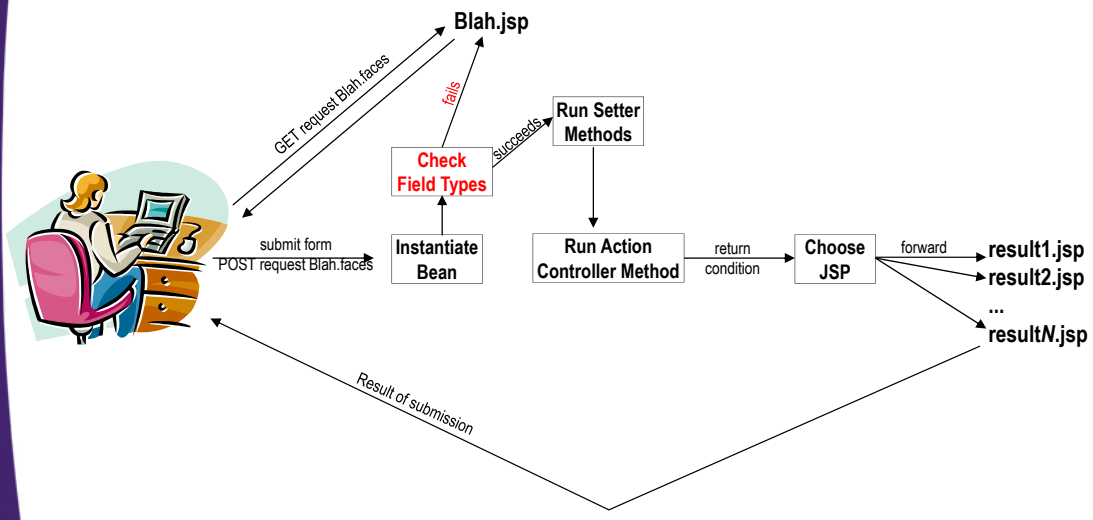
1. Create a FacesMessage
2. Store it in global list of messages via facesContext.addMessage
3. Still return null to redisplay input form
4. Output error messages in input form with h:messages or h:message

- **Advantages:**
  - Don't have to store messages yourself
  - Don't have to build HTML or use escape="false"
  - Fits with standard JSF validation
- **Disadvantages**
  - Less control over format of error messages

Validation via Type Conversion

For simple validation that just checks the types of the various input fields

JSF Flow of Control (Simplified)



Implicit Automatic Validation

- **Define bean properties to be simple standard types**
  - int/Integer, long/Long, double/Double, boolean/Boolean, etc.
    - Note that wrapper types let you have initially blank textfields
- **System attempts to convert automatically in same manner as with jsp:setProperty**
  - I.e., Integer.parseInt, Double.parseDouble, etc.
  - If there is conversion error, form redisplayed
    - And error message stored
- **Required fields**
  - You can also add `required` attribute to any input element to indicate that empty values are errors
- **Use h:message to display error messages**
  - h:message returns empty string if there is no message
  - h:message accepts `styleClass` for CSS style name
- **Add `immediate` attribute to bypass validation**
  - E.g., for h:commandButton with logout or cancel operation

Implicit Validation: Example

- **Change BidBean properties**
  - bidAmount is a double
  - bidDuration is an int
- **Change BidBean action controller**
  - Remove all validation logic.
    - However, note that *specific* values for bid amount and bid duration are no longer checked
- **Change input form**
  - Add `required` attributes to userID and keyword fields
  - Add `id` attributes to every field
  - Add h:message (with appropriate id) to the right of each input field

Bean Code: String and Numeric Properties

```
package coreservlets;

public class BidBean2 {
  private String userID;
  private String keyword;
  private double bidAmount;
  private int bidDuration;

  …

  public double getBidAmount() { return(bidAmount); }

  public void setBidAmount(double bidAmount) {
    this.bidAmount = bidAmount;
  }

  public int getBidDuration() { return(bidDuration); }

  public void setBidDuration(int bidDuration) {
    this.bidDuration = bidDuration;
  }
```

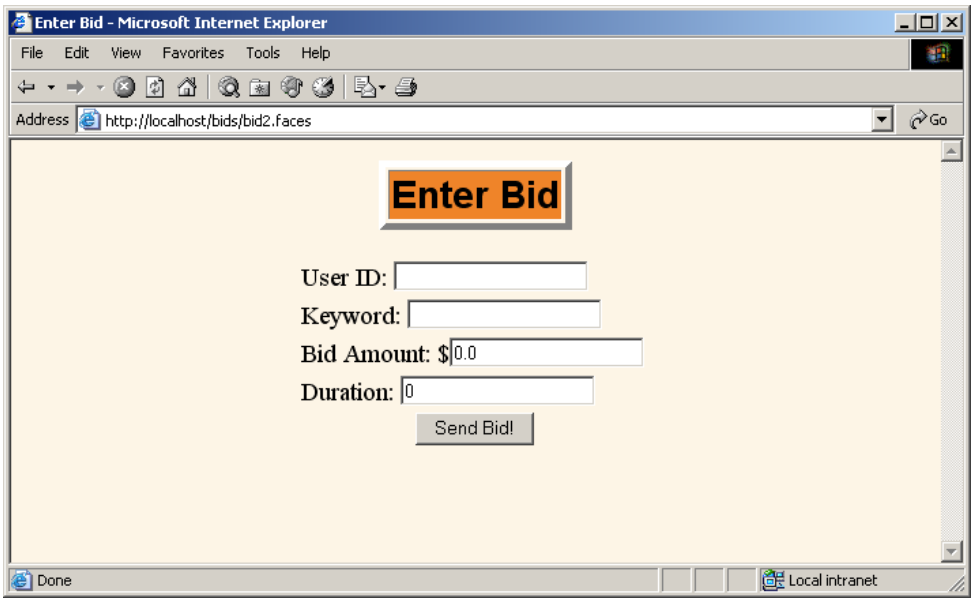
Bean Code: Action Controller

```
public String doBid() {
  return("success");
}
```

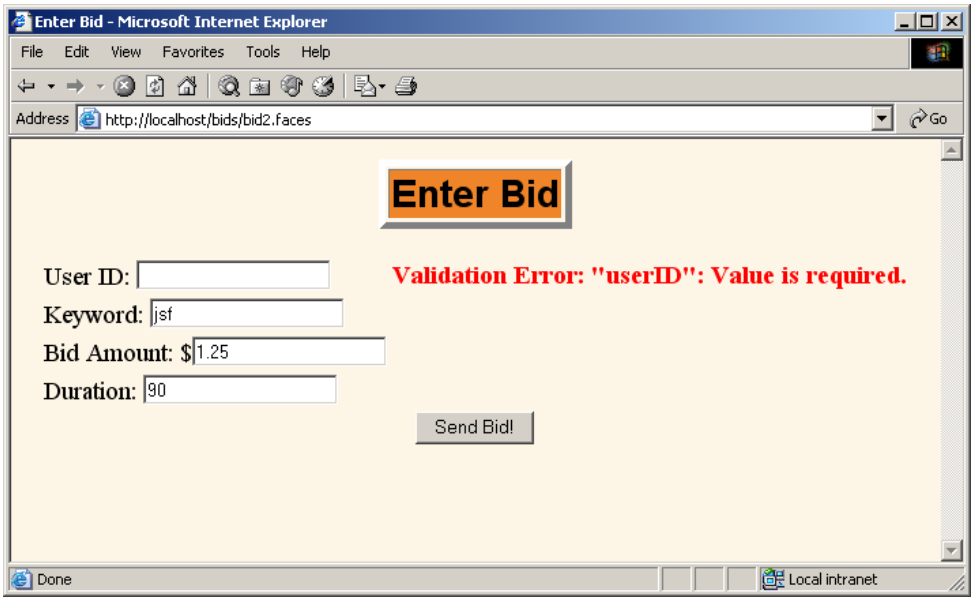

Input Form: Outputting Error Messages (bid2.jsp)

```
<h:form>
  <TABLE>
  <TR>
    <TD>User ID: <h:inputText value="#{bidBean2.userID}"
                              required="true"
                              id="userID"/></TD>
    <TD><h:message for="userID" styleClass="RED"/></TD></TR>
  <TR>
    <TD>Keyword: <h:inputText value="#{bidBean2.keyword}"
                              required="true"
                              id="keyword"/></TD>
    <TD><h:message for="keyword" styleClass="RED"/></TD></TR>
  <TR>
    <TD>Bid Amount: $<h:inputText value="#{bidBean2.bidAmount}"
                                  id="amount"/></TD>
    <TD><h:message for="amount" styleClass="RED"/></TD></TR>
  <TR>
    <TD>Duration: <h:inputText value="#{bidBean2.bidDuration}"
                               id="duration"/></TD>
    <TD><h:message for="duration" styleClass="RED"/></TD></TR>
  …</TABLE>…</h:form>
```

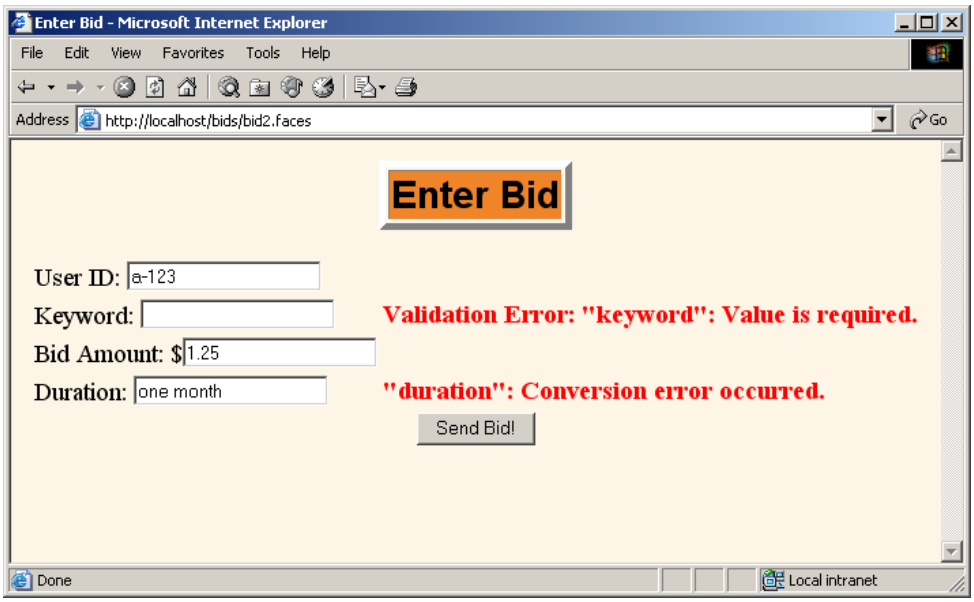
Implicit Validation: Results



Implicit Validation: Results



Implicit Validation: Results

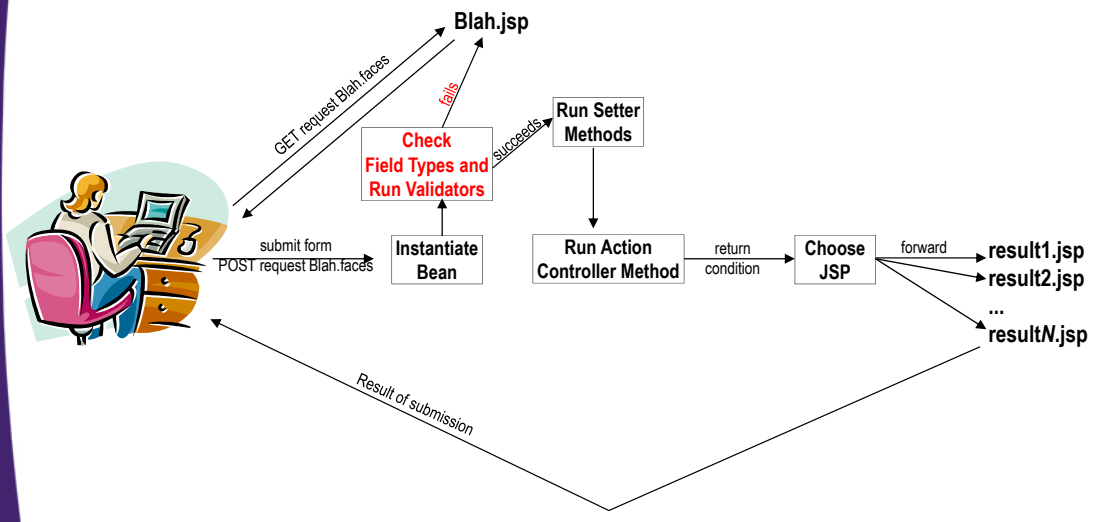


Implicit Validation: Results

Validation using the JSF Validator Tags

For when validation that is not tied to your business logic where you want to check that values are in certain ranges or of certain lengths

JSF Flow of Control (Simplified)



Explicit Automatic Validation

- **Define bean properties to be simple types**
  - int, long, double, boolean, char, etc.
- **Add f:validate*Blah* or f:convert*Blah* elements**
- **System checks that fields follow restrictions**
  - f:validate*Blah* attributes let you control format
- **If there is validation error, form redisplayed**
  - And error message stored
- **Other approaches stay the same**
  - You can still add `required` attribute to any input element to indicate that empty values are errors
  - Still use h:message to display error messages
    - h:message returns empty string if there is no message
  - Still add `immediate` attribute to bypass validation
    - E.g., for h:commandButton when implementing logout or cancel operation

Explicit Validation: Example

- **Leave BidBean unchanged from last example**
- **Change input form**
  - Enforce that userID is 5 or 6 characters long
  - Enforce that keyword is 3 or more characters long
  - Enforce that bid amount is at least 10 cents
  - Enforce that bid duration is at least 15 days

Input Form: Specifying Validation Rules (bid3.jsp)

```
<h:form>
  <TABLE>
  <TR>
    <TD>User ID:
    <h:inputText value="#{bidBean2.userID}"
                 id="userID">
      <f:validateLength minimum="5" maximum="6"/>
    </h:inputText></TD>
    <TD><h:message for="userID" styleClass="RED"/></TD></TR>
  <TR>
    <TD>Keyword:
    <h:inputText value="#{bidBean2.keyword}"
                 id="keyword">
      <f:validateLength minimum="3"/>
    </h:inputText></TD>
    <TD><h:message for="keyword" styleClass="RED"/></TD></TR>
```


Input Form: Specifying Validation Rules (bid3.jsp)

```
  <TR>
    <TD>Bid Amount:
    $<h:inputText value="#{bidBean2.bidAmount}"
                 id="amount">
       <f:validateDoubleRange minimum="0.10"/>
     </h:inputText></TD>
    <TD><h:message for="amount" styleClass="RED"/></TD></TR>
  <TR>
    <TD>Duration:
    <h:inputText value="#{bidBean2.bidDuration}"
                id="duration">
      <f:validateLongRange minimum="15"/>
    </h:inputText></TD>
    <TD><h:message for="duration" styleClass="RED"/></TD></TR>
  <TR><TH COLSPAN=2>
      <h:commandButton value="Send Bid!"
                       action="#{bidBean2.doBid}"/></TH></TR>
  </TABLE>
```

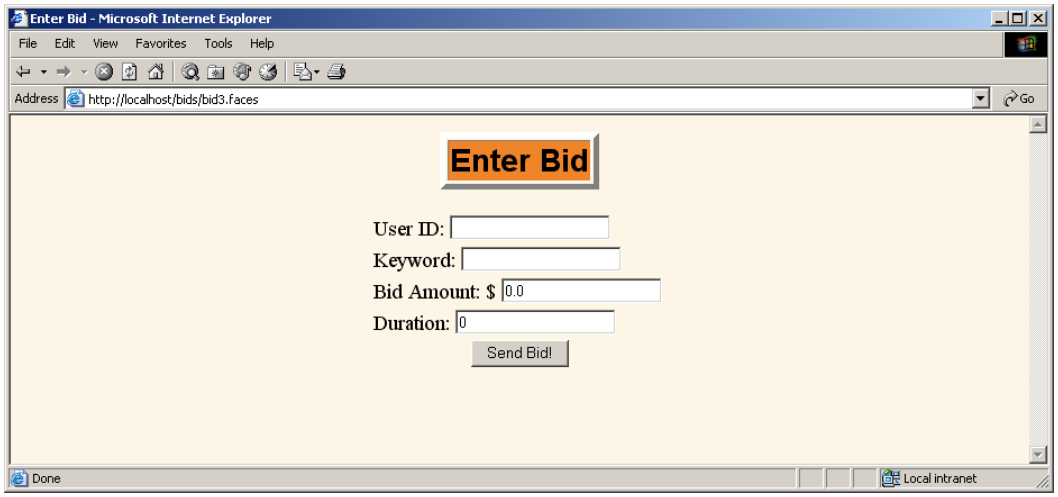
Input Form: Specifying Validation Rules (bid3.jsp)

```
<h:form>
  <TABLE>
  <TR>
    <TD>User ID:
    <h:inputText value="#{bidBean2.userID}"
                 id="userID">
      <f:validateLength minimum="5" maximum="6"/>
    </h:inputText></TD>
    <TD><h:message for="userID" styleClass="RED"/></TD></TR>
```

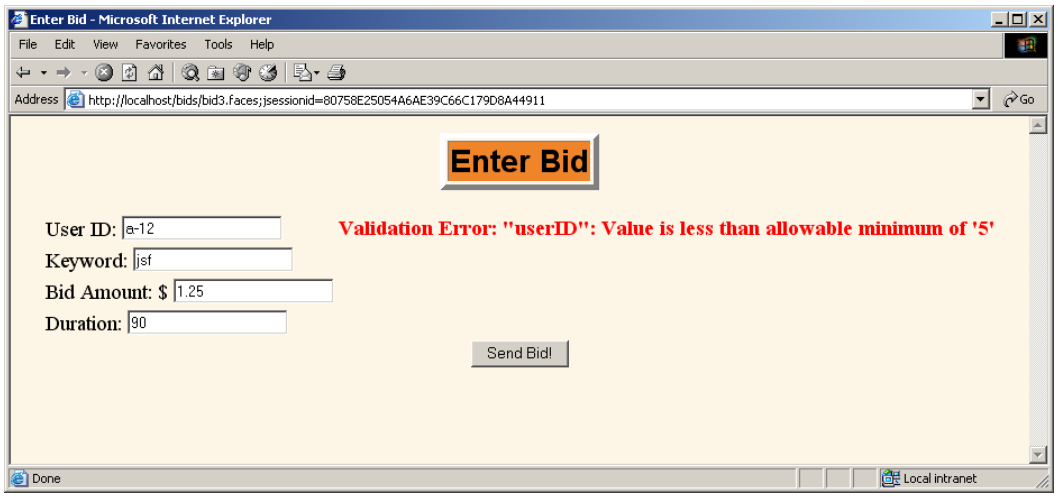

Input Form: Specifying Rules (bid3.jsp)

```
<h:form>
  <TABLE>
  <TR>
    <TD>User ID:
    <h:inputText value="#{bidBean2.userID}"
                 id="userID">
      <f:validateLength minimum="5" maximum="6"/>
    </h:inputText></TD>
    <TD><h:message for="userID" styleClass="RED"/></TD>
  </TR>
```

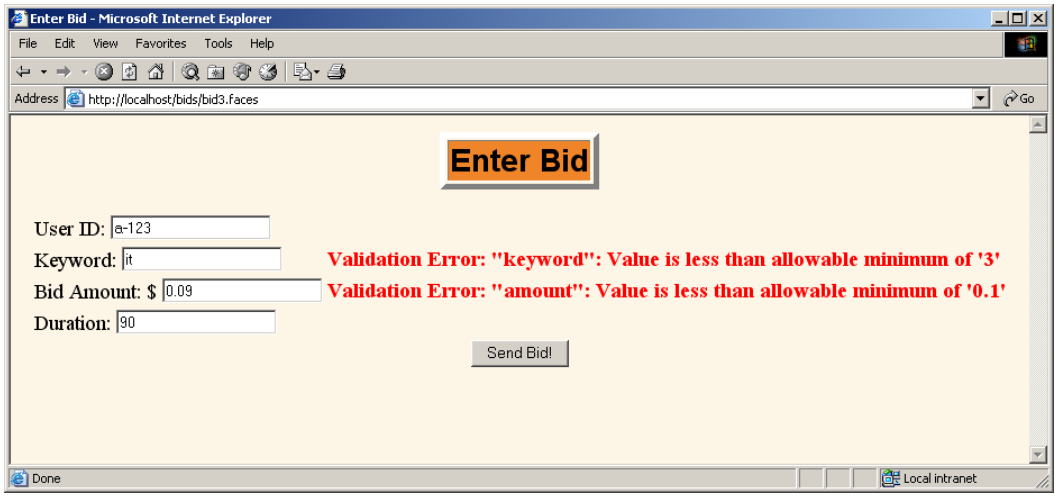
Explicit Validation: Results



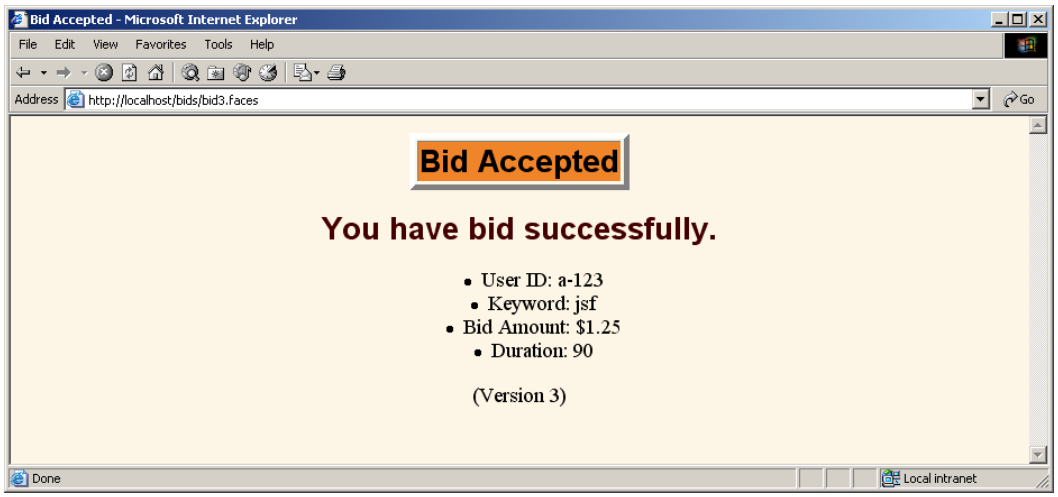
Explicit Validation: Results



Explicit Validation: Results



Explicit Validation: Results



Conversion vs. Validation

- **Both f:convert*Blah* and f:validate*Blah* check format and redisplay form if field is invalid**
  - But f:convert*Blah* also changes the format in which the field is displayed
  - f:validate*Blah* makes sense only with h:inputText
  - f:convert*Blah* makes sense with both h:inputText and h:outputText
- **Example**

```
<h:inputText  value="#{orderBean.discountCode}">
  <f:convertNumber maxFractionDigits="2"/>
</h:inputText>
```

  - Displays 0.75 or something similar (not 0.749)

Converter and Validator Attributes

- **f:convertNumber**
  - currencyCode, currencySymbol
  - groupingUsed
  - integerOnly
  - locale
  - max(min)FractionDigits
  - max(min)IntegerDigits
  - pattern (ala DecimalFormat)
  - type
    - number, currency, percentage
- **f:convertDateTime**
  - type
    - date, time, both
  - dateStyle, timeStyle
    - default, short, medium, long,full
  - pattern (ala SimpleDateFormat)
  - locale
  - timeZone
- **f:validateLength**
  - minimum
  - maximum
- **f:validateLongRange**
  - minimum
  - maximum
- **f:validateDoubleRange**
  - minimum
  - maximum

Custom Converters and Validators

- **Converters**
  - Implement Converter interface
  - getAsObject
    - Takes String; returns Object
  - getAsString
    - Takes Object; returns String
  - Conversion errors
    - Throw ConverterException with FacesMessage
  - Register in faces-config.xml
    - converter, converter-id, converter-class
  - Use `converter` attribute

```
<h:inputText
  value="#{…}"
  converter="someID"/>
```

- **Validators**
  - Implement Validator interface
  - Override validate method
  - Conversion errors
    - Throw ValidationException with FacesMessage
  - Register in faces-config.xml
    - validator, validator-id, validator-class
  - Use f:validator tag

```
<h:inputText value="#{…}">
  <f:validator
    validatorId="someID"/>
</h:inputText>
```


Customizing Error Messages

- **Create and load global properties file**
  - <application>
    <message-bundle>MyMessages</message-bundle>
    </application>
    - Assumes WEB-INF/classes/MyMessages.properties
- **Use explicit property names as given in Messages.properties**
  - javax.faces.component.UIInput.CONVERSION
  - javax.faces.component.UIInput.REQUIRED
- **Example**

```
javax.faces.component.UIInput.CONVERSION=Illegal format!
javax.faces.component.UIInput.REQUIRED=Missing value!
```

  - Changes "Conversion error occurred" to "Illegal format!"
  - Changes "Value is required" to "Missing value!"

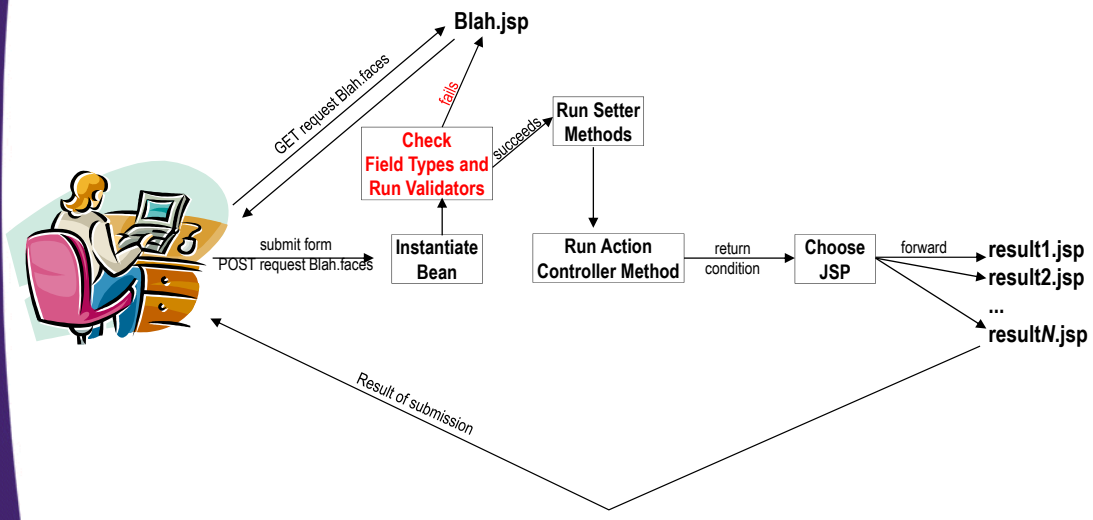
Messages.properties (from jsf-impl.jar)

```
javax.faces.component.UIInput.CONVERSION=
    {0}Conversion error occurred.
javax.faces.component.UIInput.REQUIRED=
    Validation Error\: {0}Value is required.
javax.faces.validator.NOT_IN_RANGE=
    Validation Error\: {2}Specified attribute is not between the expected
    values of {0} and {1}.
javax.faces.validator.DoubleRangeValidator.MAXIMUM=
    Validation Error\: {1}Value is greater than allowable maximum of
    "{0}"
javax.faces.validator.DoubleRangeValidator.MINIMUM=
    Validation Error\: {1}Value is less than allowable minimum of "{0}"
javax.faces.validator.DoubleRangeValidator.TYPE=
    Validation Error\: {0}Value is not of the correct type
...
```


Validation using Custom Validator Methods

For when validation that is not tied to your business logic, but when there is no builtin JSF validator

JSF Flow of Control (Simplified)



Writing Your Own Validator Methods

- **JSP**
  - For input component specify method name explicitly
    - <h:inputText id="someID"
      validator="#{someBean.someMethod}"/>
  - Use h:message as before
    - <h:message for="someID"/>
- **Java**
  - Throw ValidatorException with a FacesMessage if validation fails. Do nothing if validation succeeds.
  - Method arguments:
    - FacesContext
      - The context
    - UIComponent
      - The component being validated
    - Object
      - The submitted value (primitives use wrappers)

Custom Validator Methods: Example (JSP)

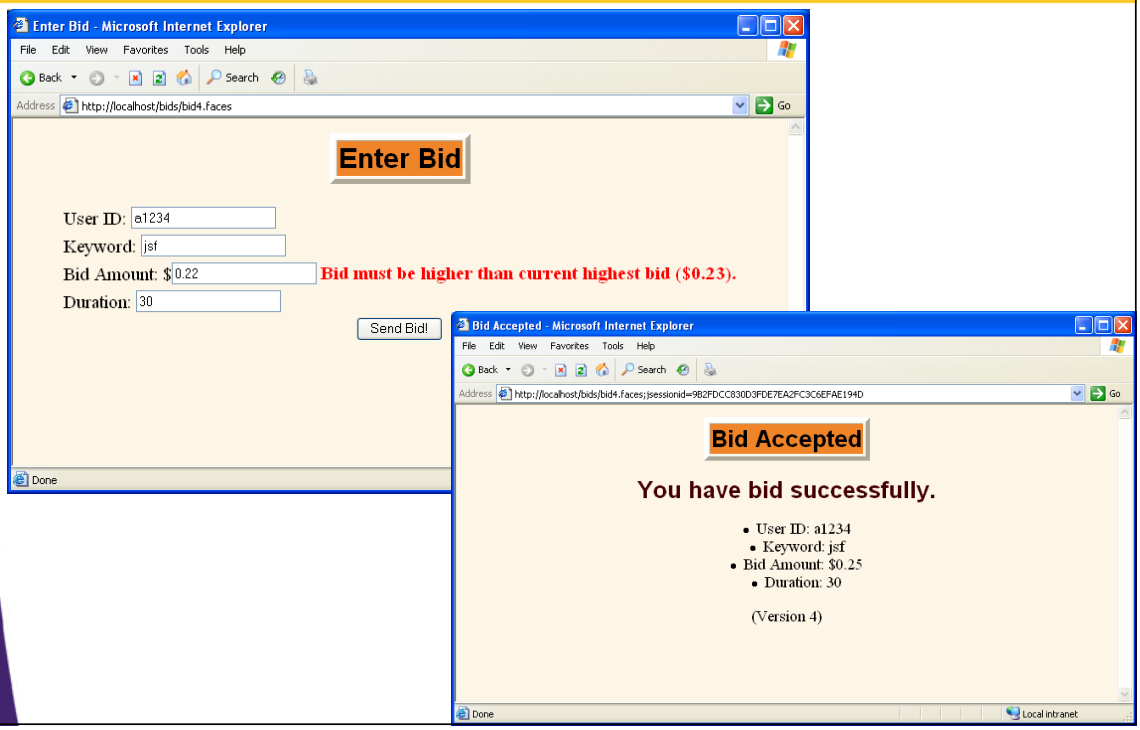
```
...
  <TR>
    <TD>Bid Amount:
    $<h:inputText value="#{bidBean2.bidAmount}"
                  id="amount" required="true"
                  validator="#{bidBean2.validateBidAmount}"/>
    </TD>
    <TD><h:message for="amount" styleClass="RED"/></TD></TR>
...
```


Custom Validator Methods: Example (Java)

```
public void validateBidAmount(FacesContext context,
                              UIComponent componentToValidate,
                              Object value)
    throws ValidatorException {
  double bidAmount = ((Double)value).doubleValue();
  double previousHighestBid = currentHighestBid();
  if (bidAmount <= previousHighestBid) {
    FacesMessage message =
      new FacesMessage("Bid must be higher than current " +
                       "highest bid ($" +
                       previousHighestBid + ").");
    throw new ValidatorException(message);
  }
}

public double currentHighestBid() {
  return(0.23); // Get from database in real life
}
```

Custom Validator Methods: Results



Summary

- **Manual validation**
  - Most flexible
  - Most work
- **Implicit automatic validation**
  - Moderately flexible
  - Very little work
- **Explicit automatic validation**
  - Very flexible
  - Little to moderate work
- **Making your own validator methods**
  - Moderately flexible
  - Moderately little work

Questions?