



Validating User Input

Originals of Slides and Source Code for Examples:

<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live training on JSF 2.x, 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**
 - Validation in the action controller method
- **Implicit automatic validation**
 - Type conversion and the “required” attribute
- **Explicit automatic validation**
 - Using `f:validateLength`, `f:validateRegex`, etc.
- **Defining your own validation methods**
 - Then using the “validator” attribute

5

© 2012 Marty Hall



Overview

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

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 valid values maintained in the form
- **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

7

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 and store in FacesMessage
 - Use h:messages to display list of error messages
- **Implicit automatic validation**
 - Use int, double, etc. bean properties. Or add required.
 - System redisplay form if there is conversion error
 - Use h:message to display field-specific error message
- **Explicit automatic validation**
 - Use f:validateLength, f:validateDoubleRange, f:validateLongRange, or f:validateRegex
 - System redisplay form if failure; use h:message again
- **Custom validation methods**
 - Create FacesMessage, wrap in ValidatorException

8



Validation in the Action Controller

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

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Manual Validation

- **Setter methods take strings only**
 - Do explicit conversion to other types in the code
 - Use try/catch blocks to handle illegal data
 - May require application-specific logic
- **Action controller checks values**
 - If values are valid
 - Return normal outcomes
 - If values are missing or invalid
 - Store error messages using `FacesContext.addMessage`
 - Return null to cause form to be redisplayed
- **Input form displays error messages**
 - Use `h:messages`
 - If there are no messages, this doesn't output anything

Standard System for Error Messages: Idea

1. Create error messages

- Store strings one at a time in request-scoped location
 - Use `FacesContext.addMessage`
 - Use null for form-wide messages
 - Use `"formId:inputElementId"` for element-specific messages
 - » Or, use `<h:form prependId="false">`, then just use the input element ID

2. Display error messages

- Output error messages as a group
 - `<h:messages/>` can automatically build `` list or `<table>` out of a set of messages
 - You can use messages for the form as a whole or messages specific to an input element
- Output single error messages
 - `<h:message.../>` will display message for a single element

11

Creating Error Messages: Details

1. Get the FacesContext

- `FacesContext.getCurrentInstance();`

2. Create a FacesMessage

- `new FacesMessage("Some warning");`
 - There are two additional options: you can store both a message summary and message details, and you can assign a severity to the problem. Later methods for outputting the messages (`h:messages`, `h:message`) can make use of these options.

3. Call the addMessage method

- With null: store a message for the form as a whole
 - `context.addMessage(null, someFacesMessage);`
- With component ID: store message specific to element
 - `context.addMessage("someID", someFacesMessage);`
 - This assumes you used `<h:form prependId="false">`
- Note
 - `getMessageList().size()` tells you if there are messages

12

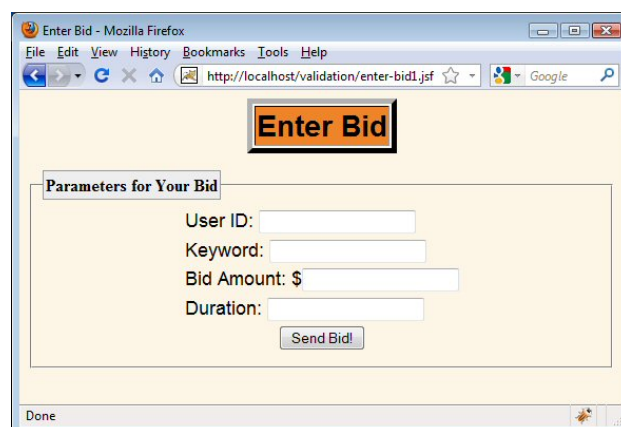
Displaying Error Messages: Details

- **h:messages – display set of messages**
 - These are usually for the form as a whole
 - `<h:form><h:messages/>...</h:form>`
 - `h:messages` takes several attributes. Most common:
 - `styleClass` (CSS name)
 - `for` (id of component whose error messages you show)
 - `layout` (default is “list”, but “table” is option)
- **h:message – display individual message**
 - These are usually for an individual element
 - `<h:inputText id="userId" .../><h:message for="userId"/>`
- **Note**
 - In early JSF 2 implementations, if you use “for”, message must come *after* the element. Fixed long ago.

13

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



The screenshot shows a web browser window titled "Enter Bid - Mozilla Firefox". The address bar shows the URL "http://localhost/validation/enter-bid1.jsf". The page has a yellow background and a title "Enter Bid" in a black box. Below the title is a section titled "Parameters for Your Bid" containing four input fields: "User ID:", "Keyword:", "Bid Amount: \$", and "Duration:". A "Send Bid!" button is located at the bottom right of the form. The browser's status bar at the bottom shows "Done".

14

Bean Code: String Properties (No Conversion)

```
@ManagedBean
public class BidBean1 {
    private String userID = "";
    private String keyword = "";
    private String bidAmount;
    private double numericBidAmount = 0;
    private String bidDuration;
    private int numericBidDuration = 0;

    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();
    }
}
```

15

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);
}
}
```

16

Bean Code: Action Controller

```
public String doBid() {
    FacesContext context = FacesContext.getCurrentInstance();
    if (getUserID().equals("")) {
        context.addMessage(null,
            new FacesMessage("UserID required"));
    }
    if (getKeyword().equals("")) {
        context.addMessage(null,
            new FacesMessage("Keyword required"));
    }
    if (getNumericBidAmount() <= 0.10) {
        context.addMessage(null,
            new FacesMessage("Bid amount must be at least $0.10.));
    }
    if (getNumericBidDuration() < 15) {
        context.addMessage(null,
            new FacesMessage("Duration must be at least 15 days.));
    }
}
```

I am using null as the first argument to addMessage because I will later use <h:messages/> to output a set of error messages for the form as a whole. If I were going to output a single message for a single element, I would provide the element id instead of null, then use <h:message for="theid"/>

17

Bean Code: Action Controller (Continued)

```
if (context.getMessageList().size() > 0) {
    return(null);
} else {
    doBusinessLogicForValidData();
    return("show-bid1");
}
}
```

Returning null means to redisplay the form

18

Input Form: Displaying Error Messages (enter-bid1.xhtml)

```
<h:form>
  <h:messages styleClass="error"/>
  <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>
```

If there are error messages, outputs them as list.
The CSS style (error) is defined by the page author.

19

CSS File (styles.css)

```
.error { font-weight: bold;
        color: red;
}
...
```

This is the CSS style referred to on the previous slide. JSF provides many hooks for passing in CSS styles, but does not define any style names. Defining them is the job of the page author.

20

Results Page (show-bid1.xhtml)

```
...
<h2>You have bid successfully.</h2>
<ul>
  <li>User ID: #{bidBean1.userID}</li>
  <li>Keyword: #{bidBean1.keyword}</li>
  <li>Bid Amount: $#{bidBean1.bidAmount}</li>
  <li>Duration: #{bidBean1.bidDuration}</li>
</ul>
...
```

The results page does not use error messages. Instead of using separate results pages to show missing data, we redisplay the input form when data is missing.

21

Manual Validation: Results (Bad Input)

The image displays two screenshots of a web browser window titled 'Enter Bid - Mozilla Firefox'. The browser's address bar shows the URL 'http://localhost/validation/enter-bid1.jsf'. The page features a form titled 'Enter Bid' with a sub-header 'Parameters for Your Bid'. The form includes four input fields: 'User ID', 'Keyword', 'Bid Amount: \$', and 'Duration:'. A 'Send Bid!' button is located at the bottom right of the form. The left screenshot shows the form with three validation errors listed in red text: '• UserID required', '• Keyword required', and '• Bid amount must be at least \$0.10.' The 'Duration' field is filled with '10'. The right screenshot shows the form with one validation error: '• Keyword required'. The 'User ID' field is filled with 'a1234', the 'Bid Amount' field is filled with '\$ 0.15', and the 'Duration' field is filled with '60'. Both screenshots show a 'Done' status at the bottom of the browser window.

22

Manual Validation: Results (Good Input)



23

Alternative: Build Your Own Error Messages

- **Builtin approach**
 - Store messages with `context.addMessage`
 - Output with `h:messages` or `h:message`
 - Pros/cons
 - Simple to store, simple to output, flexible options
 - Can mix predefined messages with custom ones
 - Can output only string, `` or `<table>`
- **Roll-your-own approach**
 - Store messages in `List<String>`
 - Make accessor to produce output
 - `<h:outputText value="#{myBean.errorMessagees}" escape="false"/>`
 - `getErrorMessagees` builds explicit HTML
 - Pros/cons
 - Complete control over the HTML that is built
 - Much more work, much less integrated

24



Validation via Type Conversion

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

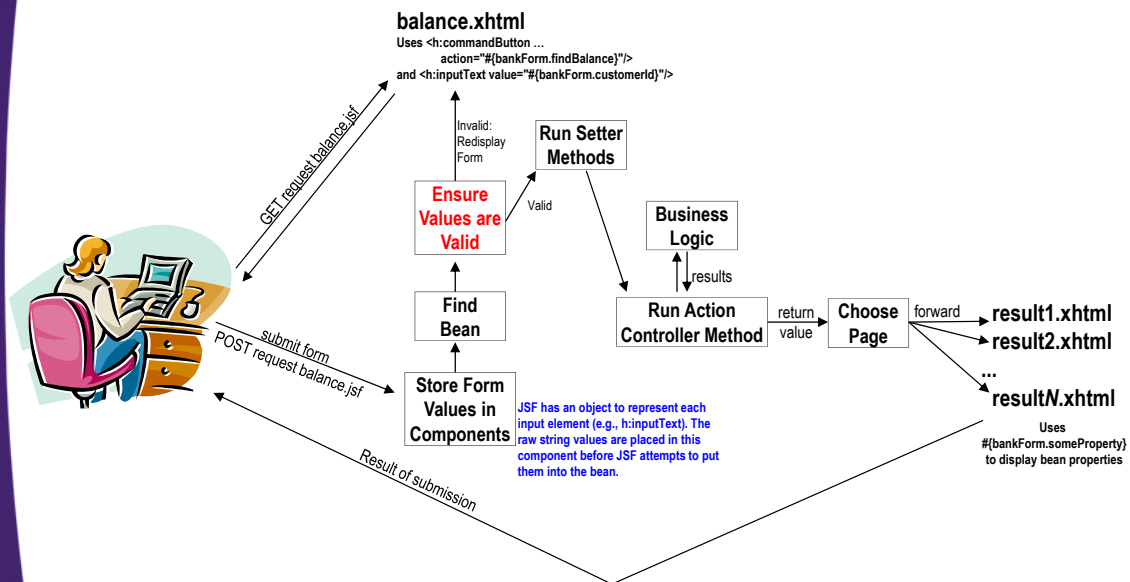
Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

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**
 - Converted 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 automatically.
 - Use `converterMessage` attribute to customize error message
- **Required fields (fields where empty vals are illegal)**
 - You can add `required` attribute to any input element
 - Use `requiredMessage` attribute to customize error message
- **Use `h:message` to display error messages**
 - `h:message` returns empty string if there is no message

JSF Flow of Control (Simplified)



"Valid" here means non-empty (if "required" is set) and able to be converted to the type that the setter method expects. But, upcoming sections will add other meanings to "valid".

27

Precedence of Validity Tests

- **Required**
 - If you mark a field as "required", and end user omits a value, then error message for missing type is generated
 - And types/validators not checked
 - Warning: if your setter expects a String, whitespace satisfies "required". I.e., whitespace in a field is not considered empty!
- **Type**
 - If field passes "required" validation (if any), then JSF checks if string can be converted to expected type
- **Validators**
 - If field passes required and type validation, then any explicit validators (see next section) are checked
- **Bypassing validation**
 - As discussed in event-handling section, a button can use immediate="true" to bypass the above behavior
 - E.g. for event-handler button, or for "Cancel" button

28

Implicit Validation: Example

- **Change BidBean properties**
 - bidAmount is a Double
 - bidDuration is an Integer
 - Use wrapper types so that textfields are blank initially. If you use double or int, *some* number must be in textfield
- **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) after each input field

29

Bean Code: Properties

```
@ManagedBean
public class BidBean2 {
    private String userID;
    private String keyword;
    private Double bidAmount;
    private Integer bidDuration;

    ...

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

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

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

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

30

Bean Code: Action Controller

```
public String doBid() {  
    doBusinessLogicForValidData();  
    return("show-bid2");  
}
```

No validation logic in the action controller, so action controller can concentrate on business logic.

31

Input Form: Top (enter-bid2.xhtml)

```
<h:form>  
  <table>  
    <tr>  
      <td>User ID:  
        <h:inputText value="#{bidBean2.userID}"  
                      required="true"  
                      requiredMessage="You must enter a user ID"  
                      id="userID"/></td>  
      <td><h:message for="userID" styleClass="error"/></td></tr>  
    <tr>  
      <td>Keyword:  
        <h:inputText value="#{bidBean2.keyword}"  
                      required="true"  
                      requiredMessage="You must enter a keyword"  
                      id="keyword"/></td>  
      <td><h:message for="keyword" styleClass="error"/></td></tr>  
  </table>  
</h:form>
```

32

Input Form: Continued (enter-bid2.xhtml)

```
<tr>
  <td>Bid Amount:
    $<h:inputText value="#{bidBean2.bidAmount}"
      required="true"
      requiredMessage="You must enter an amount"
      converterMessage="Amount must be a number"
      id="amount"/></td>
  <td><h:message for="amount" styleClass="error"/></td></tr>
<tr>
  <td>Duration:
    <h:inputText value="#{bidBean2.bidDuration}"
      required="true"
      requiredMessage="You must enter a duration"
      converterMessage="Duration must be a whole number"
      id="duration"/></td>
  <td><h:message for="duration" styleClass="error"/></td></tr>
```

33

Input Form: Continued (enter-bid2.xhtml)

```
<tr><th colspan="2">
  <h:commandButton value="Send Bid!"
    action="#{bidBean2.doBid}"/></th></tr>
</table>
</h:form>
```

34

Results Page (show-bid2.xhtml)

```
...
<h2>You have bid successfully.</h2>
<ul>
  <li>User ID: #{bidBean2.userID}</li>
  <li>Keyword: #{bidBean2.keyword}</li>
  <li>Bid Amount: $#{bidBean2.bidAmount}</li>
  <li>Duration: #{bidBean2.bidDuration}</li>
</ul>
...
```

This results page is shared by this example and the next two examples.

35

Results (Initial Form)

The screenshot shows a web browser window titled "Enter Bid - Mozilla Firefox". The address bar displays "http://localhost/validation/enter-bid2.jsf". The page content features a prominent "Enter Bid" button at the top. Below it, a section titled "Parameters for Your Bid" contains four input fields: "User ID:", "Keyword:", "Bid Amount: \$", and "Duration:". A "Send Bid!" button is positioned at the bottom of these fields. Two blue arrows point from the text below to the "Bid Amount" and "Duration" input fields.

Because we use Double and Integer for the getter methods, the fields can be initially blank (because value is null). If you use double or int, getter method will never return null, and textfield will never be empty.

36

Results (Bad Input)

Enter Bid - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/validation/enter-bid2.jsf

Enter Bid

Parameters for Your Bid

User ID: You must enter a user ID

Keyword: You must enter a keyword

Bid Amount: \$ You must enter an amount

Duration: You must enter a duration

Send Bid!

Done

Since tests for required attributes take precedence over tests for proper types, the requiredMessage attribute is displayed here.

37

Results (Bad Input)

Enter Bid - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/validation/enter-bid2.jsf

Enter Bid

Parameters for Your Bid

User ID: a1234

Keyword: You must enter a keyword

Bid Amount: \$ 0.5

Duration: blah Duration must be a whole number

Send Bid!

Done

Since the value passes the required test, the type test is applied.

38

Results (Good Input)



39

© 2012 Marty Hall



Validation using the JSF Validator Tags

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

Customized Java EE Training: <http://courses.coreservlets.com/>

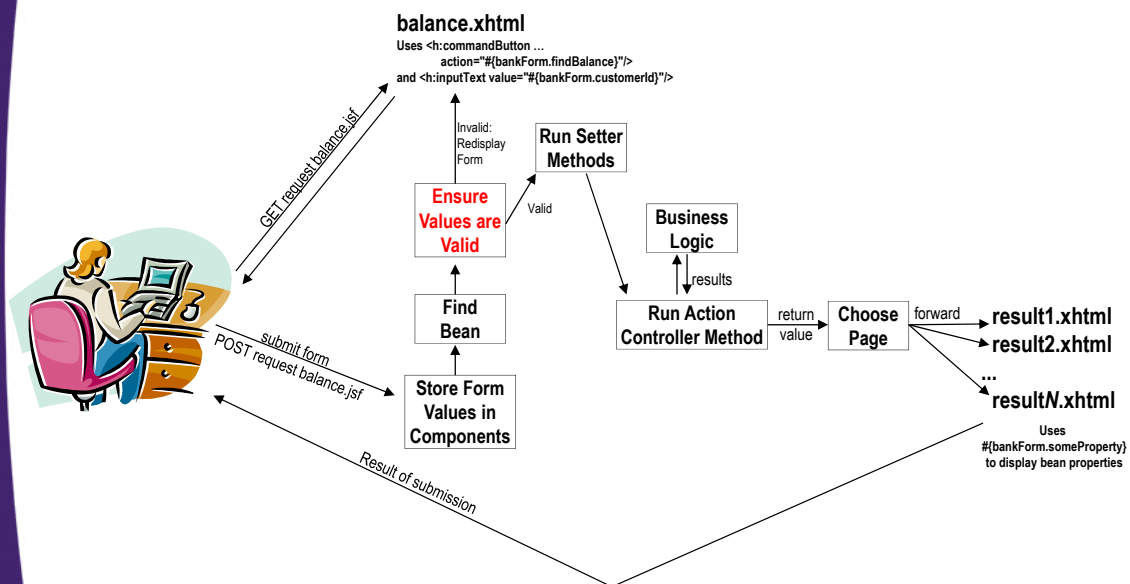
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Explicit Automatic Validation

- **Define bean properties to be simple types**
 - int/Integer, double/Double, boolean/Boolean, etc.
- **Add `f:validateBlah` or `f:convertBlah`**
 - `<f:validateLength .../>`
 - `<f:validateLongRange .../>`
 - `<f:validateDoubleRange .../>`
 - `<f:validateRegex .../>`
- **System checks validity**
 - After checking required and types, sees if value passes rule of the validator
 - If not, stores error message and redisplay the form
 - Use `validatorMessage` attribute of input element to customize the error message that is created

41

JSF Flow of Control (Simplified)



"Valid" now means non-empty (if "required" is set), able to be converted to the type that the setter method expects, and passing the rules of the explicit validator tags.

42

General Format for Validator Tags

- **Example (value must be 5 or 6 characters)**

```
<h:inputText value="#{someBean.someProperty}"
             required="true"
             requiredMessage="..."
             converterMessage="..."
             validatorMessage="..."
             id="someID">
```

```
<f:validateLength minimum="5" maximum="6"/>
```

```
</h:inputText>
```

- **Precedence of tests**

- Required
- Type conversion
- Validator rules

43

Conversion vs. Validation

- **Both `f:convertBlah` and `f:validateBlah` check format and redisplay form if field is invalid**

- But `f:convertBlah` does more
 - Can change the format in which the field is displayed
 - Can change the way String is converted to number
- `f:convertBlah` meaningful for `outputText`

- **Examples**

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

- Displays 0.75 or something similar (not 0.749)

```
<h:outputText value="#{orderBean.discountCode}">
  <f:convertNumber type="percentage"/>
</h:outputText>
```

- Displays 75% or something similar

44

Main Validator and Converter Attributes

- **f:validateLength**
 - minimum
 - maximum
- **f:validateLongRange**
 - minimum
 - maximum
- **f:validateDoubleRange**
 - minimum
 - maximum
- **f:validateRegex**
 - pattern
- **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

45

Explicit Validation: Example

- **Leave BidBean unchanged**
 - No changes from last example
 - bidAmount is Double
 - duration is Integer
- **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

46

Input Form: User ID (enter-bid3.xhtml)

```
<tr>
  <td>User ID:
  <h:inputText value="#{bidBean2.userID}"
               required="true"
               requiredMessage="You must enter a user ID"
               validatorMessage="ID must be 5 or 6 chars"
               id="userID">
    <f:validateLength minimum="5" maximum="6"/>
  </h:inputText></td>
  <td><h:message for="userID" styleClass="error"/></td>
</tr>
```

No converterMessage because property type is String, so conversion cannot fail.

Input Form: Keyword (enter-bid3.xhtml)

```
<tr>
  <td>Keyword:
  <h:inputText value="#{bidBean2.keyword}"
               required="true"
               requiredMessage="You must enter a keyword"
               validatorMessage="Keyword must be at least 3 chars"
               id="keyword">
    <f:validateLength minimum="3"/>
  </h:inputText></td>
  <td><h:message for="keyword" styleClass="error"/></td>
</tr>
```

No converterMessage because property type is String, so conversion cannot fail.

Input Form: Bid Amount (enter-bid3.xhtml)

```
<tr>
  <td>Bid Amount:
    $<h:inputText value="#{bidBean2.bidAmount}"
      required="true"
      requiredMessage="You must enter an amount"
      converterMessage="Amount must be a number"
      validatorMessage="Amount must be 0.10 or greater"
      id="amount">
      <f:validateDoubleRange minimum="0.10"/>
    </h:inputText></td>
  <td><h:message for="amount" styleClass="error"/></td>
</tr>
```

Input Form: Bid Duration (enter-bid3.xhtml)

```
<tr>
  <td>Duration:
    <h:inputText value="#{bidBean2.bidDuration}"
      required="true"
      requiredMessage="You must enter a duration"
      converterMessage="Duration must be a whole number"
      validatorMessage="Duration must be 15 days or more"
      id="duration">
      <f:validateLongRange minimum="15"/>
    </h:inputText></td>
  <td><h:message for="duration" styleClass="error"/></td>
</tr>
```

Results (Initial Form)

Enter Bid - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/validation/enter-bid3.jsf

Enter Bid

Parameters for Your Bid

User ID:

Keyword:

Bid Amount: \$

Duration:

Send Bid!

Done

51

Results (Missing Data)

Enter Bid - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost/validation/enter-bid3.jsf

Enter Bid

Parameters for Your Bid

User ID: You must enter a user ID

Keyword: You must enter a keyword

Bid Amount: \$ You must enter an amount

Duration: You must enter a duration

Send Bid!

Done

The "required" rule has first precedence.

52

Results (Type Conversion Errors)

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost/validation/enter-bid3.jsf`. The page has a title bar 'Enter Bid - Mozilla Firefox' and a menu bar with 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Tools', and 'Help'. The main content area is titled 'Enter Bid' in a large, bold, black font. Below this is a section titled 'Parameters for Your Bid' containing four input fields: 'User ID: a1234', 'Keyword: mortgages', 'Bid Amount: \$ foo', and 'Duration: bar'. To the right of the 'Bid Amount' and 'Duration' fields, there are two red error messages: 'Amount must be a number' and 'Duration must be a whole number'. A 'Send Bid!' button is located below the input fields. The status bar at the bottom of the browser window shows 'Done'.

Second precedence is conversion to the types expected by the setter methods.

53

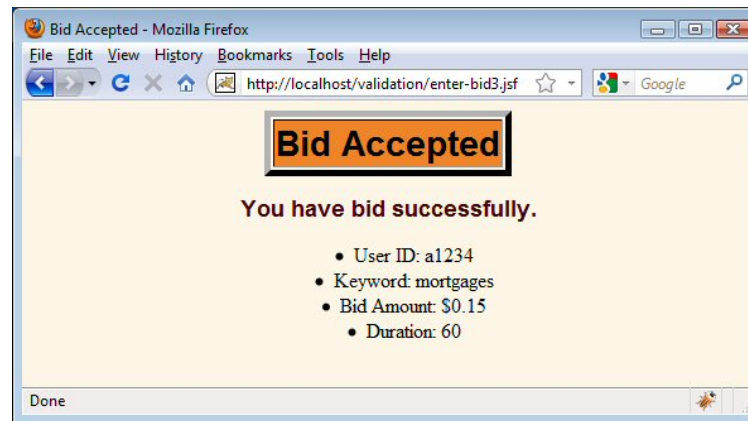
Results (Explicit Validation Errors)

The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://localhost/validation/enter-bid3.jsf`. The page has a title bar 'Enter Bid - Mozilla Firefox' and a menu bar with 'File', 'Edit', 'View', 'History', 'Bookmarks', 'Tools', and 'Help'. The main content area is titled 'Enter Bid' in a large, bold, black font. Below this is a section titled 'Parameters for Your Bid' containing four input fields: 'User ID: a123', 'Keyword: mo', 'Bid Amount: \$ 0.01', and 'Duration: 10'. To the right of the input fields, there are four red error messages: 'ID must be 5 or 6 chars', 'Keyword must be at least 3 chars', 'Amount must be 0.10 or greater', and 'Duration must be 15 days or more'. A 'Send Bid!' button is located below the input fields. The status bar at the bottom of the browser window shows 'Done'.

Third precedence is checking the rules of the explicit validator tags.

54

Results (Good Input)



55

© 2012 Marty Hall



Validation using Custom Validator Methods

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

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Writing Your Own Validator Methods

- **Facelets**

- For input component, specify method name explicitly
 - `<h:inputText id="someID" validator="#{someBean.someMethod}"/>`
- Use `h:message` as before
 - `<h:message for="someID"/>`

- **Java**

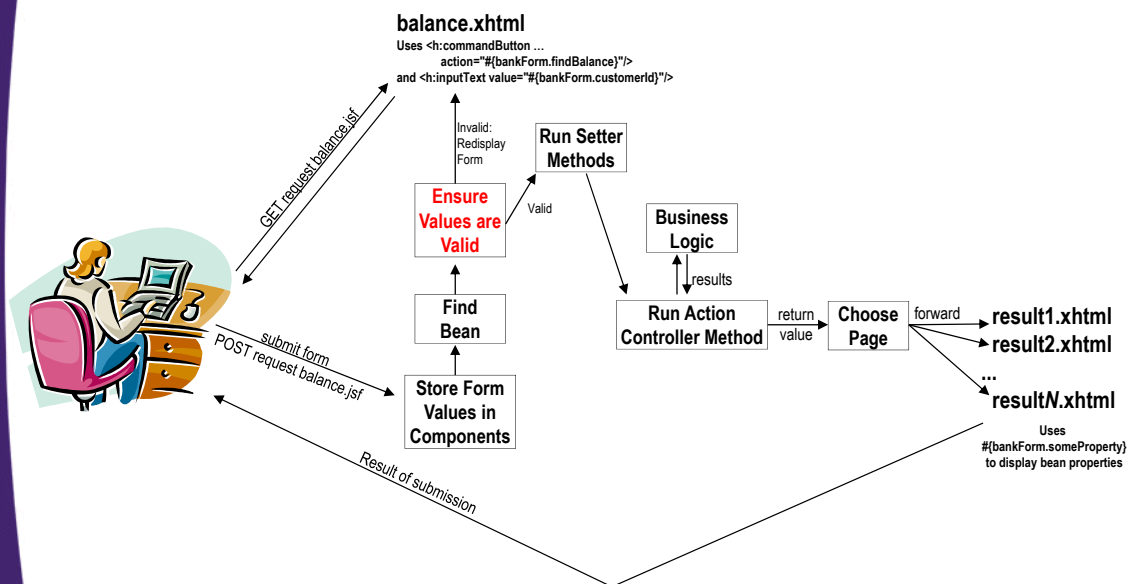
- Throw `ValidatorException` with a `FacesMessage` if validation fails. Do nothing if validation succeeds.
- Method arguments:
 - `FacesContext` (the context)
 - `UIComponent` (the component being validated)
 - `Object` (the submitted value, but primitives use wrappers)

- **Warning**

- Don't use `f:validateBlah` & custom method on same field

57

JSF Flow of Control (Simplified)



"Valid" now means non-empty (if "required" is set), able to be converted to the type that the setter method expects, passing the rules of the explicit validator tags, and not throwing exceptions in the custom validator methods.

58

Input Form: Bid Amount (enter-bid4.xhtml)

```
<tr>
  <td>Bid Amount:
    <h:inputText value="#{bidBean2.bidAmount}"
      required="true"
      requiredMessage="You must enter an amount"
      converterMessage="Amount must be a number"
      validator="#{bidBean2.validateBidAmount}"
      id="amount"/>
  </td>
  <td><h:message for="amount" styleClass="error"/></td>
</tr>
```

There is no validatorMessage because the custom validation method produces the message.

Facelets code for user ID, keyword, and duration are the same as in the previous example.

59

BidBean2: Validation Method

```
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);
    }
}
```

60

Results: Failing Custom Validation



The screenshot shows a Mozilla Firefox browser window titled "Enter Bid - Mozilla Firefox". The address bar displays "http://localhost/validation/enter-bid4.jsf". The page content includes a header "Enter Bid" in a black box. Below it is a form titled "Parameters for Your Bid" with four input fields: "User ID: a1234", "Keyword: mortgages", "Bid Amount: \$ 0.15", and "Duration: 60". A red error message "Bid must be higher than current highest bid (\$0.23)." is displayed next to the Bid Amount field. A "Send Bid!" button is located at the bottom of the form. The status bar at the bottom of the browser window shows "Done".

Results for empty form, missing values, type conversion, validator tags, and successful submission same as previous example.

61

© 2012 Marty Hall



Wrap-Up

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Validation in action controller**
 - When validation linked to business logic, or when you are comparing values of multiple fields
 - But, the lowest-level and most tedious of the 4 approaches
- **Type conversion (and “required”)**
 - When it is enough to simply enforce types (counting non-empty as a type)
- **Explicit `f:validateBlah` tags**
 - When you are checking one field at a time, and want to enforce values are in certain ranges or satisfy a regex
- **Making your own validator methods**
 - When you are checking one field at a time, but want to perform complex tests not built into any of the tags

63

© 2012 Marty Hall



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.