# CS 416
# Web Programming

Java Server Faces cont.

Dr. Williams
Central Connecticut State University

## Managed beans

- The crux of how JSF works is driven by **managed beans**
- A managed bean is just like any other Java bean but has the annotation @ManagedBean before the declaration of the class
- Managed beans mean they can be used within JSF pages and bound to fields, functionality, events

## Binding values

- A JSF page is made dynamic by adding **value binding expressions**
- A binding expression is of the form:

```
<h:inputText label="email"
  value="#{customer.email}">
```

- This syntax says that the managed bean with class name Customer will have its value email tied to this input text field. Note when referencing bean first letter is lower case

confirmation.xhtml

## Binding example

- carDataEntry.xhtml
- carSaved.xhtml

- Key things bind of input and output the same

- Bound objects page to page no need to specify population of values – bindings specified in original page already populated on target page

## Validation framework

- Upon form submit, JSF’s validation framework executes checking validity of fields
  - If a field value is invalid the framework writes an error message to the page and prompts the user to fix their data
  - Form will not redirect or process until the validation criteria has been met
- Result: provides the ability for the developer to specify allowed values and after that the developer can assume valid values received

## Creating validation

- Validation can be created in three different ways
    - Built in validation
    - Custom validator class (implements Validator interface)
    - Custom validation function
- Once developed and added to the code, framework takes care of enforcing validation

## Validator review: Standard validation

| Validation tag | Description |
|---|---|
| <f:validateLength minimum="2" maximum="5"> | Verifies input length is between the tag's minimum and maximum values |
| <f:validateDoubleRange minimum="0.2" maximum="75.5"> | Verifies input is valid Double within the specified range inclusive |
| <f:validateLongRange minimum="-56" maximum="1234"> | Verifies input is valid Long within the specified range inclusive |
| <f:validateRequired> | Validates that the tag is not empty (same as adding required="true" in the input text tag) |
| <f:validateRegex pattern="[a-z]+"> | Validates that the value is in the format specified by the regular expression in the pattern attribute |
| <f:validateBean> | Allows validation to be done by annotations on the managed bean without having to add validators to the JSF tags |

## Example of basic validation

```
<h:inputText label="First Name"
 value="#{customer.firstName}" required="true">
 <f:validateLength minimum="2"
     maximum="30">
 </f:validateLength>
</h:inputText>
```

## Custom validation – Validator class

- One of the ways custom validation can be implemented is by creating a class that implements javax.faces.validator.Validator
- The class also must be annotated with @FacesValidator(value="emailValidator")
- This interface has one method validate which throws the ValidatorException
- Essentially what your class does is check the passed input, if it is fine nothing is done otherwise a ValidatorException is thrown

EmailValidator.java

## Using a faces validator

- A faces validator is used by adding:

```
<h:inputText label="Email"
    value="#{customer.email}" >
 <f:validator
 validatorId="emailValidator" />
</h:inputText>
```

- Upon submitting the form, the contents of the Email textbox will then be sent to the faces validator we registered with the name “emailValidator” with our annotation
- The JSF framework will then take over to ensure the validation criteria is met before the form is submitted, messaging the user if it is not

## Custom validation - Validator methods (sometimes logic part of MVC-Model)

- Another way validation can be created is with validator methods
- Validator methods can be added to any managed bean
- To support validation 1-many functions may be added to the class that accept the same parameters as the validate method on the validator interface
- Useful for adding custom bean logic to the managed entity bean or for creating utility classes that contain many forms of validation

ValidationUtils.java

## Using validator methods

```
<h:inputText label="Last Name"
  value="#{customer.lastName}" required="true"
  validator="#{validationUtils.validateAlphaSpace}">
  <f:validateLength minimum="2"
  maximum="30"></f:validateLength>
</h:inputText>
```

Example of all:  CarDataEntrySoln.java

## MVC – Model
## Integration with JPA

- One of the main benefits of JSF is its seamless integration with the data model and the controller logic
- Since there is nothing special about Managed beans other than the @ManagedBean annotation a managed bean can also be an entity bean
- Binding values on page enables easy persistence and reading from DB

## MVC - Controller

- The last component of MVC is the controller which is responsible for the **flow of the application** and **persistence**
- The JSF framework does this by being able to direct control to managed beans
- Managed bean provides method that command button (<h:command>) can bind to its action

## Controller cont.

Controller is responsible for

- Enhancing object model values if needed
- Persisting data if needed
- Directing the application to the next page

## Controller setup

- Like the other components in the JSF framework, the controller is also a managed bean
- Functions that can be bound to the pages action need to take no parameters and return a String
- String returned will be the page the application should be directed to

## Controller and data model

- For the controller to access a bean an attribute for that bean must be added as a property of the controller bean
- Property is then marked as

```
@ManagedProperty(value = "#{customer}")
private Customer customer;
```

Which binds the customer bean to the customer attribute

- The controller can then persist the bean by creating an instance of the entityManager and calling persist on the bean

## MVC demo

- customerDataEntry.xhtml
- CustomerController.java
- confirmationSaved.xhtml

## Creating your own MVC

- Modify controller *CarController* to control submission of the car info page that persists the car before directing the browser to the display page *CarSaved*

## JSF and Ajax

- Ajax can be added to JSF facelets with very little code
- All of the client side JavaScript needed is generated
  - Asynchronous call
  - Waiting for readystate
  - Parsing and injecting results in the page
- Result is developer can focus on **content** rather than details

## Ajax tag

- Within a JSF page the Ajax tag of the JSF standard components:

<f:ajax

- The tag is used as a child of another tag
  - Respond to event related to tag
  - Respond to action of parent element

## Tag structure

```
<h:inputText id="txtInput" value="#{controller.text}">

  <f:ajax render="myTargetId" event="keyup" />

</h:inputText>
```

- Basic flow is when the keyup event is fired the value of the parent tag (txtInput) will be set to the bound variable (controller.text)
- The ***render*** attribute specifies one or more target id’s that should be updated (if more than one, separated by spaces)
  - Render specifies the ids of the tags that should be updated with their new bound values

## Events

- Ajax can be tied to any parent element’s actions
  - Blur, change, click, dblclick, focus, keydown, keypress, keyup, mouseup, mousedown, mousemove, mouseout, mouseover, select, valueChange
- Alternatively can be tied to action listener (covered later)

## Design of update

- When bound attribute is updated through the *set* method application takes action and updates relevant information so *get* on rendered elements return the results
- Generally:
  - If updates are based on consistency of bean, Ajax is tied to bean element
  - Otherwise *set*er is called on a controller which executes some logic then updates relevant fields

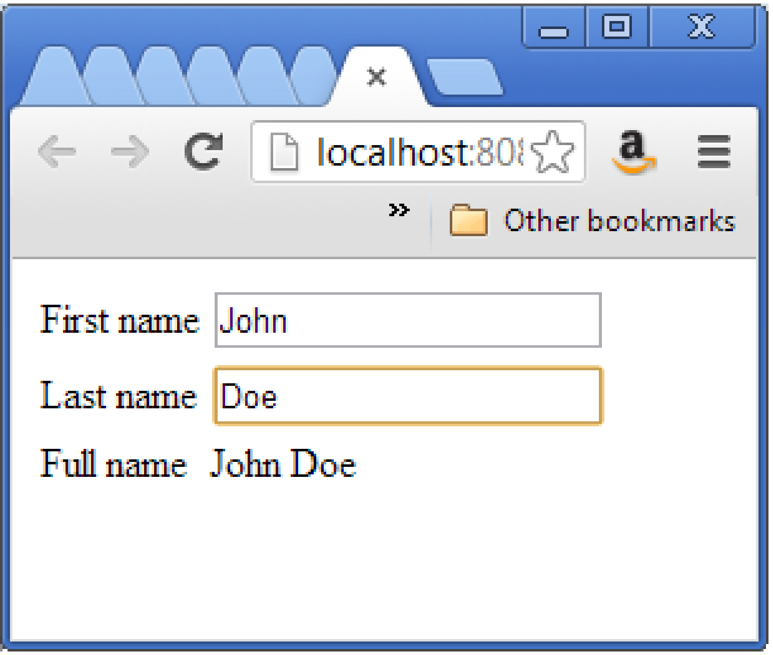
## Multiple arguments

- By default the only value updated is parent of Ajax tag
- To specify multiple bound attributes should be updated can be specified with the *execute* attribute

<f:ajax execute="firstName lastName" render="fullName suggestedLogin"...

- Result is parent and both firstName and lastName bound attributes updated and then render update on updated bound attributes

## Name example (nameExample.xhtml)



First name John

Last name Doe

Full name John Doe

- Add ajax to input fields
- Respond to keyup
- Tag's field will be updated automatically – need to execute any other related fields
- Render full name which will take new bound values

## Ajax on actions

- Alternative to responding to events is to respond to actions
- Action is user clicking on a command button (button) or command link (appears as URL)
- Adding Ajax to command is typically done to update the page without actually changing the page Ex.
  - Calculations
  - Search

## Actions cont.

- Rather than acting on setters a command can be bound to a method as an Action Listener

```
<h:commandButton
 actionListener="#{controller.calculateTotal}"
 value="Calculate Sum">
 <f:ajax execute="first second"
 render="sum"/>
</h:commandButton>
```

## Action listeners

- Action listeners generally a method on a controller designed to perform functionality similar to submit
- When combined with Ajax form processing result is page update without reload of submit
- To implement action listener, any method that returns void and takes as parameters a javax.faces.event.ActionEvent as its argument

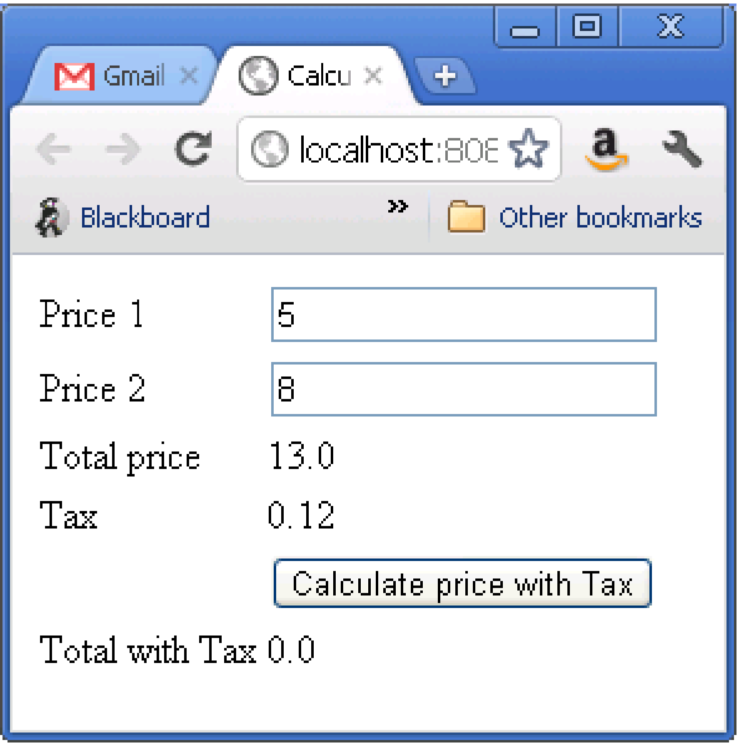
```
public void calculateTotal(ActionEvent
  event){
}
```

# Ajax in action

- Create a basic sales calculator
  1. Use Ajax event handler so that when price1 or price2 is updated the total price reflects the change
  2. Use action listener to update total with tax when clicked calls listener method:

priceCalculation.calcTotalWithTax

(this could also be just populated dynamically as above but this gives you an example of an action listener)

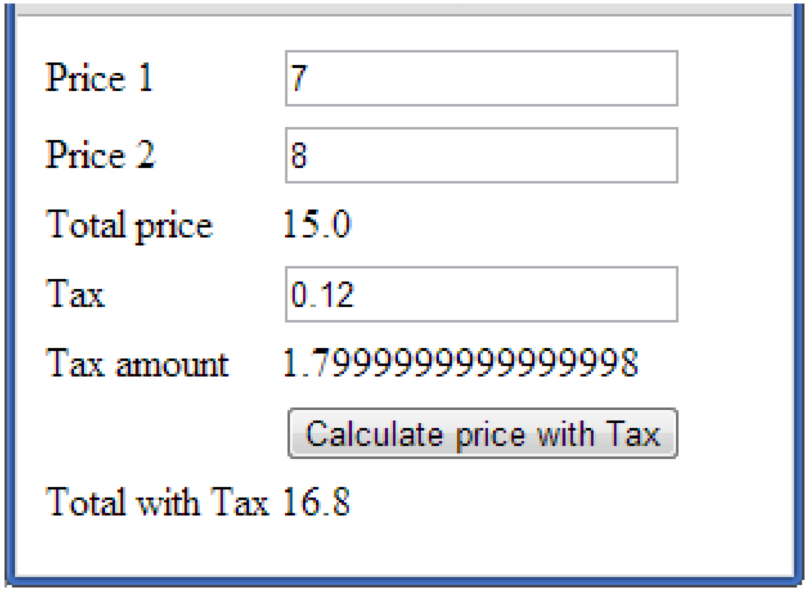


|  |  |
|---|---|
| Price 1 | 5 |
| Price 2 | 8 |
| Total price | 13.0 |
| Tax | 0.12 |

Total with Tax 0.0

## Your turn

- Modify ajaxInActionWithTax
- Update code so changing price1, price2 or tax displays the new tax amount



| | |
|---|---|
| Price 1 | 7 |
| Price 2 | 8 |
| Total price | 15.0 |
| Tax | 0.12 |
| Tax amount | 1.7999999999999998 |

Total with Tax 16.8

## Working with record sets

- To create a table of a list of items use

<h:dataTable

Element and bind contents to controller method to retrieve matching records

See customerSearch.xhtml and CustomerController.getMatchingCustomers

## Ajax in action

- Create a car search page

  User can enter part of car make and get results

  1. Add lookup method to controller
  2. Create jsf to call and display results

How would you modify it so you could enter part of make AND part of model?

## Project stages

- One of the features of JSF is for the application and application server to behave differently whether in development or production
  - SystemTest
  - UnitTest
  - Development
  - Production
- Specifically the application server will log differently depending on production stage – ie. More debugging information in development, no informational logging in production

## Project stages cont.

- In addition to having the application server automatically adjust its logging, your code can execute based on the property as well:

```
FacesContext facesContext =
           FacesContext.getCurrentInstance();
Application application =
           facesContext.getApplication();
if (application.getProjectStage().equals(
   ProjectStage.Production){
   //do something
}else if (application.getProjectStage().equals(
           ProjectStage.Development){

…
```

## Setting the project stage

- To set the project stage a custom JNDI resource must be defined
- In the GlassFish admin console
  - Go to JNDI|Custom resources
  - Click new
  - Enter:

  JNDI Name: java.faces.PROJECT_STAGE

  Resource type: java.lang.String

  Factory Class:
    com.sun.faces.application.ProjectStageJndiFactory
  - Add a new property “stage” set its value to “Development” and click Save