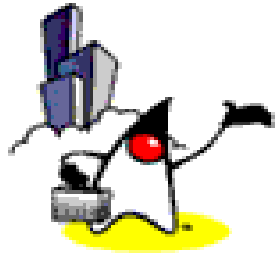
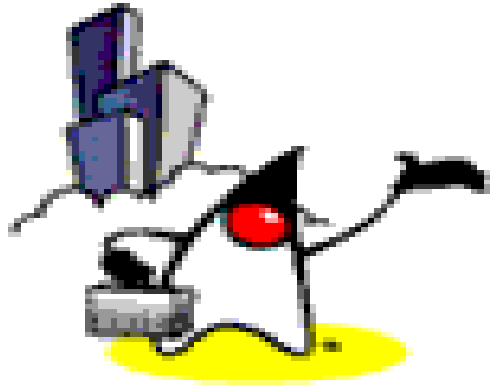
Managed Beans & Backing Beans



Topics

- Writing bean methods
- Referencing bean methods
- Binding component values and instances with external data sources
- Managed beans vs. Backing beans



Writing Bean Methods

Types of Bean Methods

- Getter and setter methods of properties
  - Follows JavaBeans convention
- JSF related methods
  - Validation methods
  - Action event handler methods
  - Value change event handler methods
  - Navigation handling methods (Action methods)

Validation Methods

- A validation method must accept a *FacesContext* and a *UIInput* component as parameters
  - just like the *validate* method of the *Validator* interface does
- Only values of *UIInput* components or values of components that extend *UIInput* can be validated
- A component refers to this method with its *validate* attribute

Example: ValidateEmail() Method CheckoutFormBean in the Coffee Break

```
public void validateEmail(FacesContext context, UIInput toValidate) {
  String message = "";
  String email = (String) toValidate.getValue();
  if (email.indexOf('@') == -1) {
    toValidate.setValid(false);
    message = CoffeeBreakBean.loadErrorMessage(context,
      CoffeeBreakBean.CB_RESOURCE_BUNDLE_NAME,
        "EMailError");
    context.addMessage(toValidate.getClientId(context),
      new FacesMessage(message, ""));
  } else {
      toValidate.setValid(true);
  }
}
```

Action Event Handler Method

- Handles an *ActionEvent*
  - Accepts an *ActionEvent* and returns void
- Referenced with the component's *actionListener* attribute
- Only UI components that implement *ActionSource* interface can refer to this method
  - *UICommand*
  - *UIButton*

Example: chooseLocaleFromLink() Method of CarStore bean (cardemo)

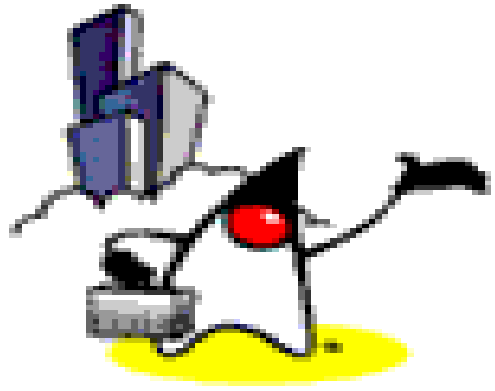
```
public void chooseLocaleFromLink(ActionEvent event) {
  String current = event.getComponent().getId();
  FacesContext context = FacesContext.getCurrentInstance();
  context.getViewRoot().
    setLocale((Locale) locales.get(current));
    resetMaps();
}
```

Action Handler (Navigation Handler) Method

- Takes no parameters and returns an outcome String
- Is referenced with the component's *action* attribute

Example: buyCurrentCar() Method of CarStore bean (cardemo)

```
public String buyCurrentCar() {
  getCurrentModel().getCurrentPrice();
  return "confirmChoices";
}
```



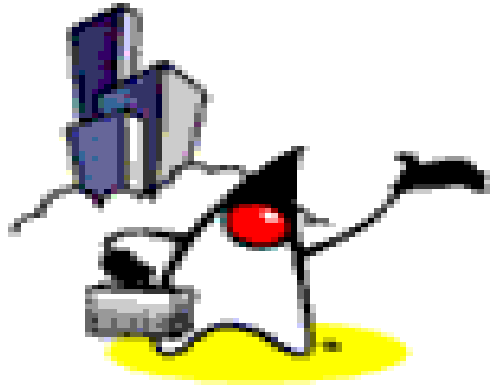
Referencing bean Methods

How to reference bean methods in Custom Tags?

- As values of the following attributes
  - *action*
  - *actionListener*
  - *validator*
  - *valueChangeListener*
- Value of attributes are in the form of JSF EL expression
- Only *UIInput* components or components that extend *UIInput* can use the *validator* or *valueChangeListener* attributes

Example:

```
<h:inputText ...
  validator="#{CarBean.validateInput}"
  valueChangeListener="#{CarBean.processValueChange}" />
```



Binding Component Values and Instances to External Data Sources

Types of Binding

- Binding a Component Value to a Property
- Binding a Component Instance to a Bean Property
  - This bean is called Backing bean
- Binding a Component Value to an Implicit Object (System Object)

Binding a Component Value to a Property

- To bind a component's value to a bean property
  - You must first specify the name of the bean and property as the value of the *value* attribute using a JSF EL expression #{X.Y}
    - X is value of <managed-bean-name>
    - Y is value of <property-name>
  - Bean and its properties are declared in Application Configuration File (*faces-config.xml*)

Example:

- Calling page

```
<h:outputText value="#{CarBean.carName}" />
```

- Bean declaration in App. Conf. File

```
<managed-bean>
  <managed-bean-name>CarBean</managed-bean-name>
  <managed-property>
    <property-name>carName</property-name>
    <value>Jalopy</value>
  </managed-property>
  ...
</managed-bean>
```

Reasons for Binding a Component Value to a Bean Property

- The page author has more control over the component attributes.
- The backing bean has no dependencies on the JavaServer Faces API (such as the UI component classes), allowing for greater separation of the presentation layer from the model layer.
- The JavaServer Faces implementation can perform conversions on the data based on the type of the bean property without the developer needing to apply a converter.

Binding a UI Component Instance to a Bean Property

- A component instance can be bound to a bean property using a JSF EL expression with the *binding* attribute of the component's tag
- When a component instance is bound to a backing bean property, the property holds the component's local value.
  - Conversely, when a component's value is bound to a backing bean property, the property holds its model value, which is updated with the local value during the update model values phase of the life cycle.

Reasons for Binding a UI Component Instance to a Bean Property

- The backing bean can programmatically modify component attributes
- The backing bean can instantiate components rather than let the page author do so

Example

- Calling page

```
<inputText binding="#{UserNumberBean.userNoComponent}" />
```

- Bean class

```
UIInput userNoComponent = null;
...
public void setUserNoComponent(UIInput userNoComponent) {
  this.userNoComponent = userNoComponent;
}
public UIInput getUserNoComponent() {
  return userNoComponent;
}
```

Example: from carstore application

- Calling page (bookcashier.jsp)

```
<h:selectBooleanCheckbox
    id="fanClub"
    rendered="false"
        binding="#{cashier.specialOffer}" />
```

- Bean class (CashierBean.java)

```
public class CashierBean extends AbstractBean {
    protected Date shipDate;

    // ------------------------------------------------- Component Properties
    protected String name = null;
    protected String shippingOption = "2";
    protected String[] newsletters = new String[0];
    UIOutput specialOfferText = null;
    UIOutput thankYou = null;
    UISelectBoolean specialOffer = null;
    private CreditCardConverter creditCard = null;
    private String stringProperty = "This is a String property";
```

Guideline

- In most situations, you will bind a component's value rather than its instance to a bean property. You'll need to use a component binding only when you need to change one of the component's attributes dynamically
- For example, if an application renders a component only under certain conditions, it can set the component's rendered property accordingly by accessing the property to which the component is bound.

Binding a Component Value to an Implicit Object

- Implicit objects
  - applicationScope
  - cookie
  - facesContext
  - header
  - headerValues
  - initParam
  - param
  - paramValues
  - requestScope
  - sessionScope
  - tree

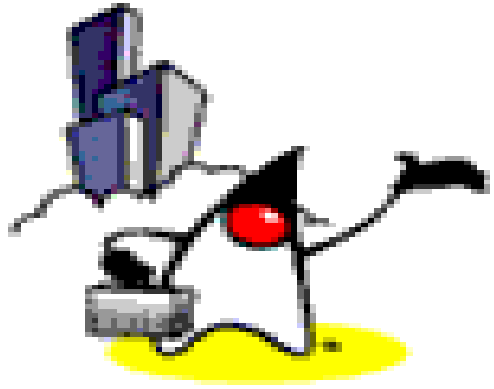
Example: Binding to Implicit Object

- Calling page

```
<h:outputText id=version value="#{initParam.versionNo}"
```

- web.xml file

```
<context-param>
  <param-name>versionNo</param-name>
  <param-value>1.05</param-name>
</context-param>
```



Managed Bean vs. Backing Bean

Managed Bean vs. Backing Bean

- Managed bean is a JavaBean registered in the faces-config file
  - These beans have properties that are bound to the values of UIComponents
  - Uses *value* attribute
  - <h:inputText value="#{ManagedBean,propertyName}" ... />

Managed Bean vs. Backing Bean

- Backing bean is a special type of managed-bean consisting of properties that are UIComponents
  - Instead of the bean properties being bound to the UIComponent values, they are instead bound to the UIComponents themselves
  - Uses *binding* attribute

```
<h:inputText
  binding="#{BackingBean.someUICommandInstance}
  " ... />
```

Passion!