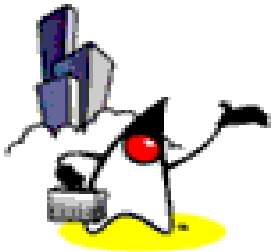
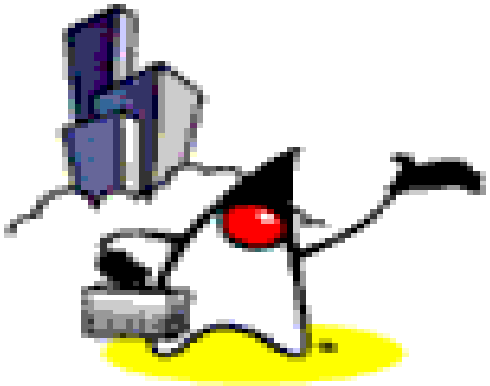
UI Component Model



Sub Topics

- What is a UI component?
- UI component classes
- UI component rendering model
- Conversion model
- Event and listener model
- Validation model



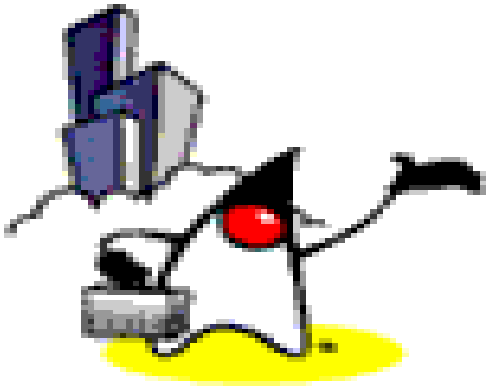
UI Component Model:
What is a UI Component?

What is a UI Component?

- A well defined, familiar idiom for UI design
- Are configurable, reusable elements that compose the user interfaces of JSF applications
- Can be simple, like a button, or compound, like a table, which can be composed of multiple components
- Extensible through composition, adding new components
- Accessible via JSF custom tags in JSP page

JSF UI Component Model

- A set of UIComponent classes for specifying the state and behavior of UI components
- A rendering model that defines how to render the components in different ways.
- An event and listener model that defines how to handle component events
- A conversion model that defines how to plug in data converters onto a component
- A validation model that defines how to register validators onto a component



UI Component Model:
UI Component Classes

UI Component Classes

- UI Component classes specify all of the UI component functionality
  - Retrieving values from input form (decoding)
  - Holding component state
  - Maintaining a reference to model objects
  - Driving event-handling
  - Rendering – creating markup (encoding)

UI Component Classes

- JSF implementation provides a set of UI component classes
  - Developers can extend these UI component classes to create custom UI components
- All JSF UI component classes extend from UIComponentBase
  - UIComponentBase defines the default state and behavior of a UIComponent

How UI Component classes are used by Page authors?

- Most page authors and application developers will not have to use these classes directly
  - They will instead include the components on a page by using the component's corresponding tags
- Most of these component tags can be rendered in different ways
  - For example, a UICommand component can be rendered as a button or a hyperlink using different tags

Built-in UI Component Classes

- UICommand:
  - Represents a control that fires actions when activated.
- UIForm:
  - Encapsulates a group of controls that submit data to the application. This component is analogous to the form tag in HTML.
- UIGraphic:
  - Displays an image.

Built-in UI Component Classes

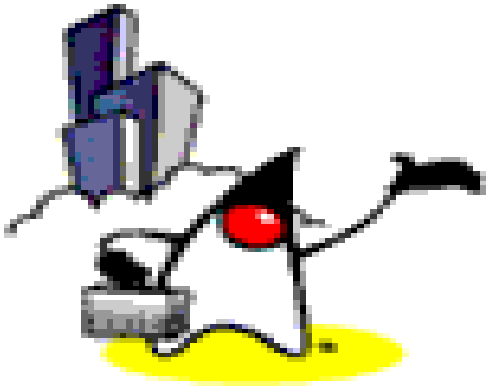
- UIInput:
  - Takes data input from a user
  - is a subclass of UIOutput
- UIOutput:
  - Displays data output on a page
- UIPanel
  - Displays a table
- UIParameter:
  - Represents substitution parameters

Built-in UI Component Classes

- UISelectItem:
  - Represents a single item in a set of items.
- UISelectItems:
  - Represents an entire set of items.
- UISelectBoolean:
  - Allows a user to set a boolean value on a control by selecting or de-selecting it. This class is a subclass of UIInput.
- UISelectMany:
  - Allows a user to select multiple items from a group of items. This class is a subclass of UIInput.

Built-in UI Component Classes

- UISelectOne:
  - Allows a user to select one item out of a group of items.This class is a subclass of UIInput.



UI Component Model:
ValueHolder Type

ValueHolder Type

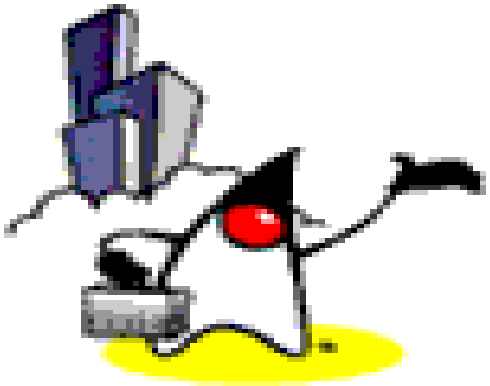
- An interface that may be implemented by any concrete UIComponent that wishes to support a local value, as well as access data in the model tier via a value expression
- Support conversion between String and the model tier data's native data type

Example: ValueHolder

```
public void printComponentValue(UIComponent comp){

    // Display the value of the component only when
    // the component is ValueHolder type.
    if (comp instanceof ValueHolder) {
        System.out.println("      value of the component = " +
                                    ((ValueHolder)comp).getValue());
    }

}
```

EditableValueHolder Type

- An extension of ValueHolder that describes additional features supported by editable components, including ValueChangeEvents and Validators
- Methods
  - *addValueChangeListener(ValueChangeListener listener)*
  - *addValidator(Validator validator)*



UI Component Model: Component Rendering Model

Component Rendering

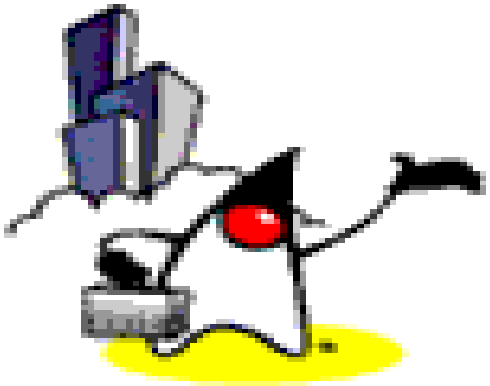
- Rendering is handled by Render kit not by component classes
  - Component writers can define the behavior of a component once, but create multiple renderers
- Page authors and application developers can change the appearance of a component on the page by selecting the tag that represents the appropriate component/renderer combination
  - <h:commandButton>
  - <h:commandLink>

RenderKit

- Defines how component classes map to component tags appropriate for a particular client
- JSF implementation includes a built-in RenderKit for rendering to an HTML client
- For every UI component that a RenderKit supports, the RenderKit defines a set of Renderer objects

Renderer Object

- Defines a different way to render the particular component to the output defined by the RenderKit
- Example
  - UISelectOne component has three different renderers
    - One of them renders the component as a set of radio buttons
    - Another renders the component as a combo box.
    - The third one renders the component as a list box.



UI Component Model: JSP Custom Tags in HTML Renderer Kit

Tags in HTML Renderer Kit

- Each JSP custom tag defined in the standard HTML RenderKit class is composed of
  - component functionality, defined in the UIComponent class
  - rendering attributes, defined by the Renderer

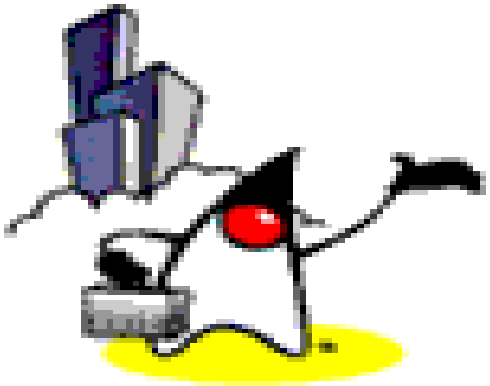
Example Tags

- <commandButton> & <commandLink> tags
  - “command” defines UI component
  - “Button” and “Link” defines rendering attribute

Tag	Rendered as
command_button	Figure 20-4 Login Login Button
command_link	Figure 20-5 hyperlink *A Hyperlink*

greeting.jsp

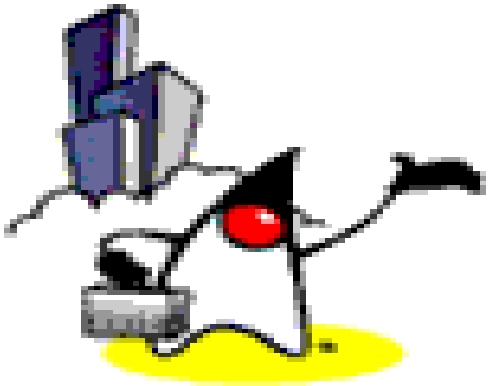
```
<f:view>
  <h:form id="helloForm" >
    <h2>Hi. My name is Duke.  I'm thinking of a number from
      <h:outputText value="#{UserNumberBean.minimum}"/> to
      <h:outputText value="#{UserNumberBean.maximum}"/>.  Can you guess it?
    </h2>

    <h:graphic_image id="waveImg" url="/wave.med.gif" />
    <h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
                 validator="#{UserNumberBean.validate}"/>
    <h:commandButton id="submit" action="success" value="Submit" />
    <p>
    <h:messages style="color: red; font-family: 'New Century Schoolbook', serif;
        font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>

  </h:form>
 </f:view>
</HTML>
```



UI Component Model: Conversion Model

Conversion Model

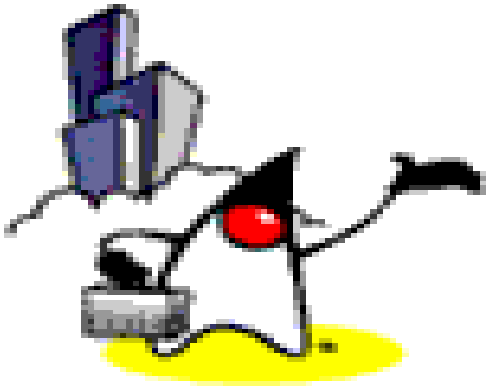
- A component can be associated with server-side model object data
- Two views of the component's data:
  - model view
  - presentation view
- Component's data can be converted between the model view and the presentation view
  - This conversion is usually performed automatically by the component's renderer
  - Custom conversion is supported via Converter



UI Component Model:
Event & Listener Model

JSF Event & Listener Model

- Similar to JavaBeans event model
  - Listener and Event classes that an application can use to handle events generated by UI components
  - An Event object identifies the component that generated the event and stores information about the event
  - To be notified of an event, an application must provide an implementation of the Listener class and register it on the component that generates the event
  - When the user activates a component, such as by clicking a button, an event is fired



UI Component Model:
Validation Model

Validation Model

- Like the conversion model, the validation model defines a set of standard classes for performing common data validation checks
- jsf-core tag library also defines a set of tags that correspond to the standard Validator implementations
- Most of the tags have a set of attributes for configuring the validator's properties
  - minimum and maximum

greeting.jsp

```
<f:view>
  <h:form id="helloForm" >
    <h2>Hi. My name is Duke.  I'm thinking of a number from
      <h:output_text value="#{UserNumberBean.minimum}"/> to
      <h:output_text value="#{UserNumberBean.maximum}"/>.  Can you guess it?
    </h2>

    <h:graphic_image id="waveImg" url="/wave.med.gif" />
    <h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
                 validator="#{UserNumberBean.validate}"/>
    <h:command_button id="submit" action="success" value="Submit" />
    <p>
    <h:messages style="color: red; font-family: 'New Century Schoolbook', serif;
        font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>

  </h:form>
 </f:view>
</HTML>
```

Passion!