13

Handling Application Events

Objectives

After completing this lesson, you should be able to do the following:

- Define the types of JavaServer Faces (JSF) events
- Create event listeners for a JSF application
- Describe how the JSF life cycle handles validation
- List the types of validation provided by JSF

JSF Event Model

- The JSF Event Model is based on the event model defined by the JavaBeans specification where the event is represented by a specific class.
- An event source object fires an event by calling an event notification method on event listener objects registered to receive the event, passing a reference to the event object as a notification method argument.
- Developers can write their own event listener implementations, or reference a backing bean method by using Expression Language (EL).

Types of Events

JSF supports:

- Action events:
  - Occur when a command component is activated—for example, when a user clicks a button or a link
  - Return a navigation case
- Value change events:
  - Occur when the local value of an input component changes—for example, when a user selects a check box
  - Are used for managing UI elements
- Phase events:
  - Execute as part of the JSF life cycle
  - Can be used to augment standard behavior

Action Events

- Command components raise action events.
- Code can be in the backing bean or external class.
- Stub code is generated by JDeveloper on demand.
- The action is registered in the page source.
- Action events are called in the application phase of the life cycle.
- Action events are the last to fire after other listeners (for example, value change events).

Creating Action Events

- Two ways:
  - Double-click a command component in the Visual Editor.
  - Enter a method name (including the `()`) in the Action property of the component.
- JDeveloper:
  - Creates a method in the backing bean
  - Adds the method to the Action property in the page source
- Add your custom code.
- Action methods return a string outcome.

Value Change Events

- Input components raise value change events.
- Code can be in the backing bean or external class.
- Stub code is generated by JDeveloper.
- The value change event is registered in the page source.
- Value change events are called in the application phase of the life cycle.
- Value change events fire before action events.

Creating Value Change Events

- Select the input component in the Visual Editor.
- Set the `valueChangeListener` property to a method name that includes `().`
- JDeveloper:
  - Creates the method in the backing bean
  - Adds the method to the `valueChangeListener` property in the page source
- Add your custom code.

Event Listener Classes

- Action listener:
  - Is a class that wants to be notified when a command component fires an action event
  - Implements `javax.faces.event.ActionListener`
- Value change listener:
  - Is a class that wants to be notified when an input component fires a value change event
  - Implements `javax.faces.event.ValueChangeListener`

Handling Action Events

To listen for an action event, you need to:

- Create a class that implements a `javax.faces.event.ActionListener` interface
- Register the action listener instance on the component's `actionListener` attribute

➢ To register the action listener, drag an action listener from JSF > Core to the component.

➢ You can create multiple listeners on each component.

Handling Value Change Events

To handle a value change event for an input component, you need to:

- Implement the `ValueChangeListener` interface
- Register the value change listener instance on the component (using the `f:valueChangeListener` tag)

➢ To register the action listener, drag a value change listener from JSF > Core to the component.

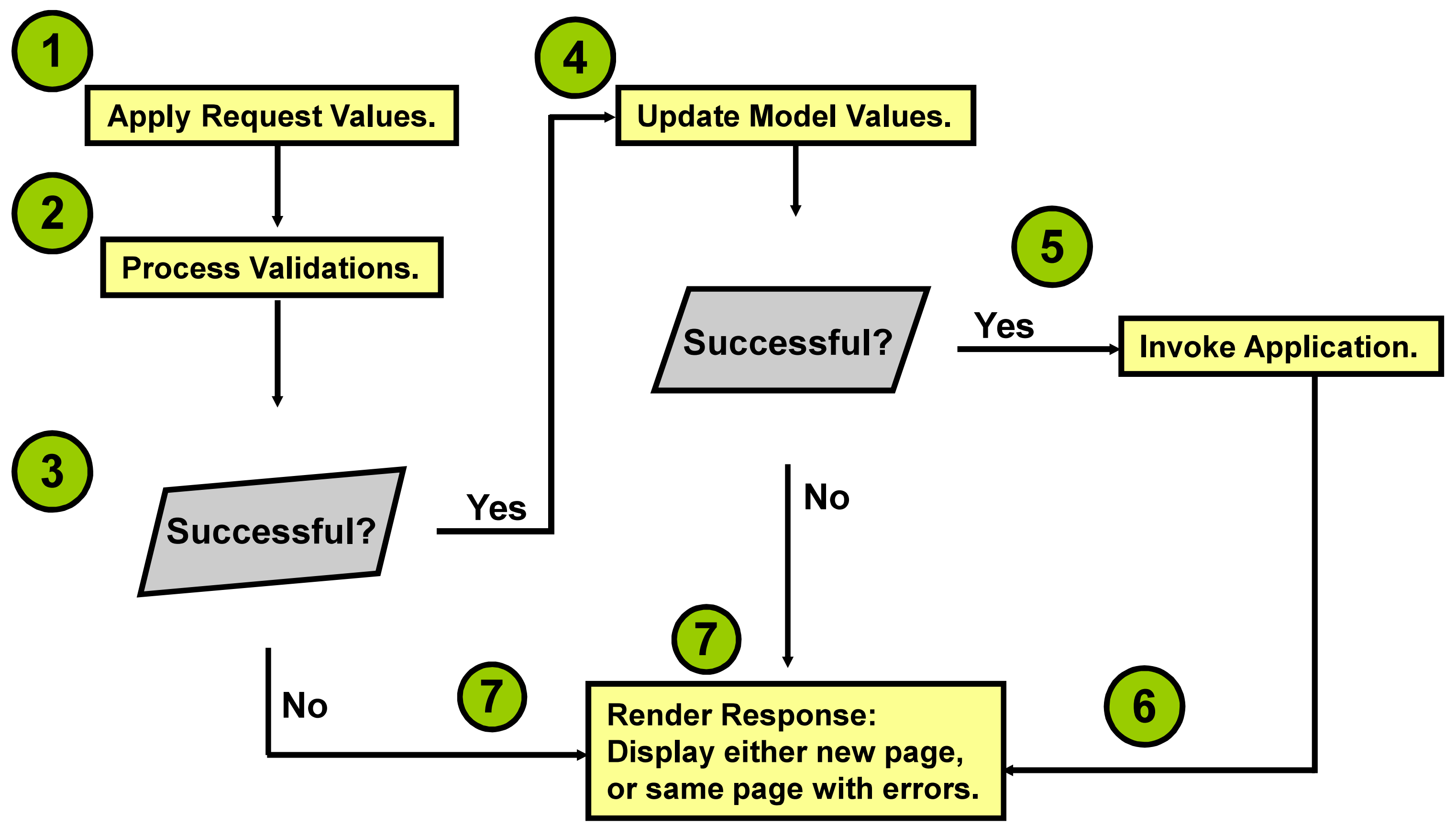
➢ You can create multiple value change listeners per component.

Event and Listener Execution Order

Events and listeners fire in the following order:

- Conversions and Then Validators
- Value change listeners
- Action listeners
- Action events

Validation in the JSF Life Cycle

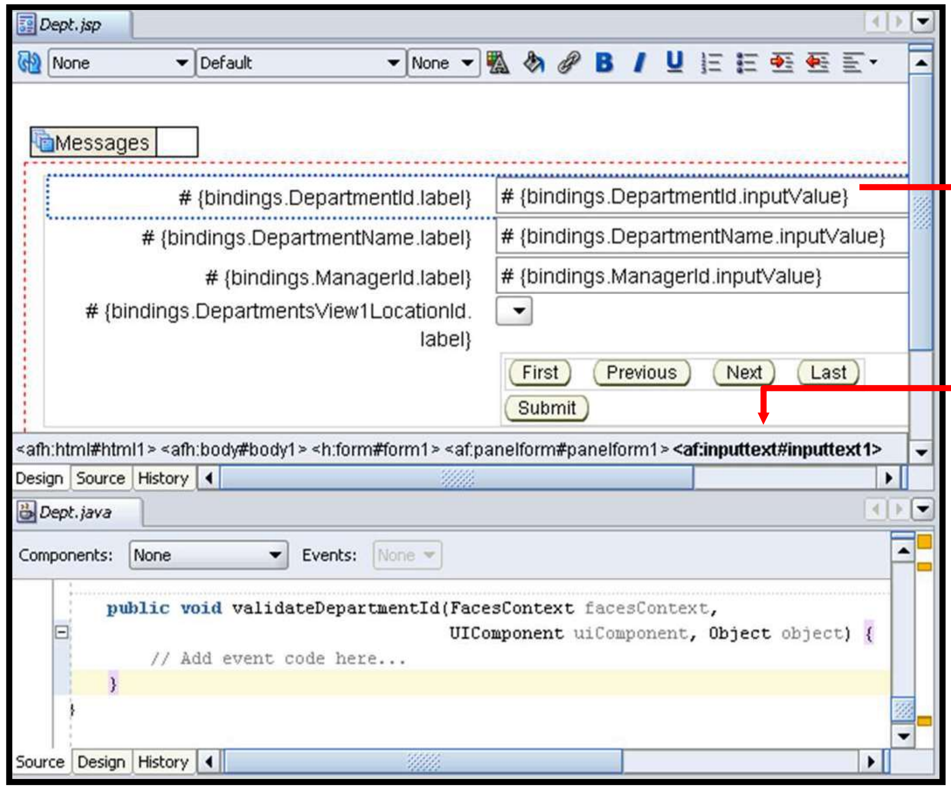


JavaServer Faces Validators

Validation can be performed through:

- Separate validator components (classes), which provide a high degree of reusability
- Backing bean code
  - Simpler to understand because code is co-located with the subject
  - Not reusable
- JSF Reference Implementation (RI). It provides default validator components.
  - No client-side validation code, such as JavaScript, is generated.
  - Multiple validators can be used on one input item.

Creating Backing Bean Validation



Dept.jsp

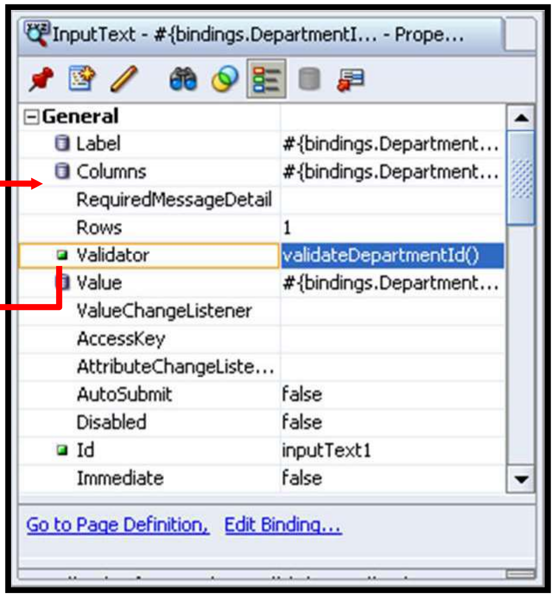
# {bindings.DepartmentId.label}	# {bindings.DepartmentId.inputValue}
# {bindings.DepartmentName.label}	# {bindings.DepartmentName.inputValue}
# {bindings.ManagerId.label}	# {bindings.ManagerId.inputValue}
# {bindings.DepartmentsView1LocationId.label}	

<afh:html#html1> <afh:body#body1> <h:form#form1> <af:panelform#panelform1> **<af:inputtext#inputtext1>**

Dept.java

Components: None Events: None

```
public void validateDepartmentId(FacesContext facesContext,
                                 UIComponent uiComponent, Object object) {
    // Add event code here...
}
}
```



InputText - #{bindings.DepartmentI... - Prope...

General	
Label	#{bindings.Department...
Columns	#{bindings.Department...
RequiredMessageDetail	
Rows	1
Validator	validateDepartmentId()
Value	#{bindings.Department...
ValueChangeListener	
AccessKey	
AttributeChangeListe...	
AutoSubmit	false
Disabled	false
Id	inputText1
Immediate	false

MENTORLABS℠

Backing Bean Validator: Code Example

```
public      void      validateDepartmentId(FacesContext
   facesContext, UIComponent uiComponent,Object object)
   {
  String expr = "[^0-9]{1,10}";
  Pattern p = Pattern.compile(expr);
  Matcher m = p.matcher((String)object);
//write     message     if     input     is     invalid
   if (!m.find()){
     facesContext.addMessage("RegExError",new
                          FacesMessage(
      FacesMessage.SEVERITY_ERROR,          "InvalidValue
   Provided for Deptno",null));
      facesContext.getApplication().getViewHandler().
     restoreView(facesContext,"/Dept.jsp");
   }
 }
```

Input Validation

- Use input validation to:
  - Avoid “garbage in, garbage out” applications
  - Block bad user input
  - Avoid attacks
  - Avoid misuse of form fields
- JavaServer Faces and ADF provide the following options for validation:
  - ADF binding validation
  - JSF validator components
  - Backing bean “coded” validation

Summary

In this lesson, you should have learned how to:

- Describe how the JSF event model enables developers to write their own event listener implementations
- Use action events to fire when a component is activated and value change events when the local value of an input component has changed
- Use managed beans as the state holder of user input and the component's data
- Describe how the JSF life cycle handles validation

Practice 1: Overview

This practice covers the following topics:

- Adding a SelectionListener for the Category Tree
- Coordinating the values of two components
- Connecting multiple pages with shared values
- Adding Validators
- Creating an Edit page