



Java Server Faces (JSF)

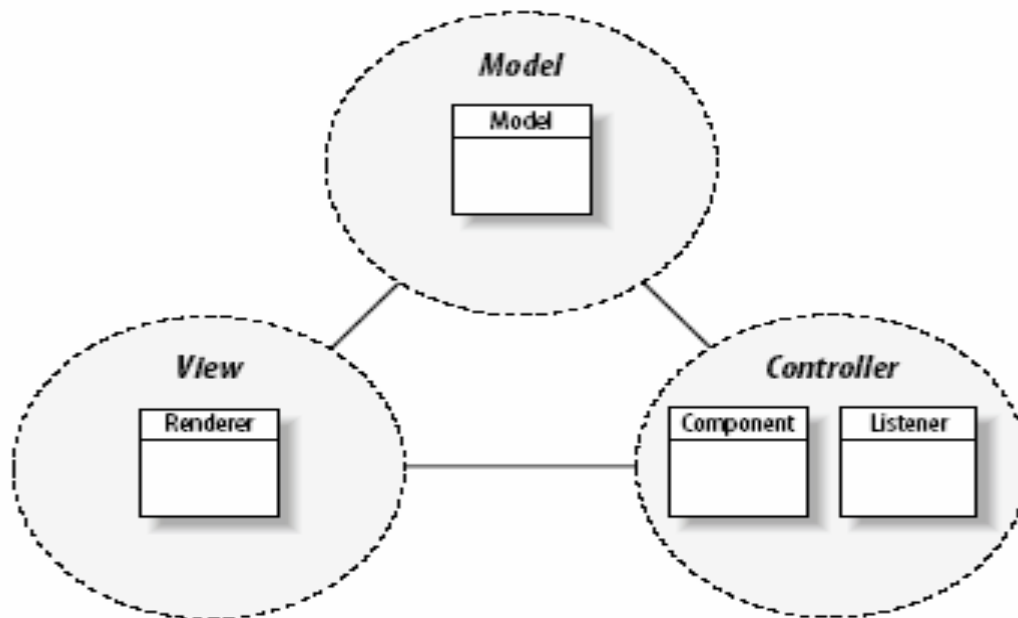


Java Server Faces (JSF)

- JSF is used for building Java Web application interfaces.
- Like Swing and AWT, JSF is a development framework that provides a set of standard, reusable GUI components.
- JSF provides the following development advantages:
 - Clean separation of behavior and presentation
 - Component-level control over statefulness
 - Events easily tied to server-side code
 - Leverages familiar UI-component and Web-tier concepts
 - Offers multiple, standardized vendor implementations
 - JSF's fine-tuned event model allows your applications to be less tied to HTTP details and simplifies your development effort.

Java Server Faces (JSF)

- A typical JSF application consists of the following parts:
 - JavaBeans components for managing application state and behavior
 - Event-driven development (via listeners as in traditional GUI development)
 - Pages that represent MVC-style views; pages reference view roots via the JSF component tree



JSF MVC Design

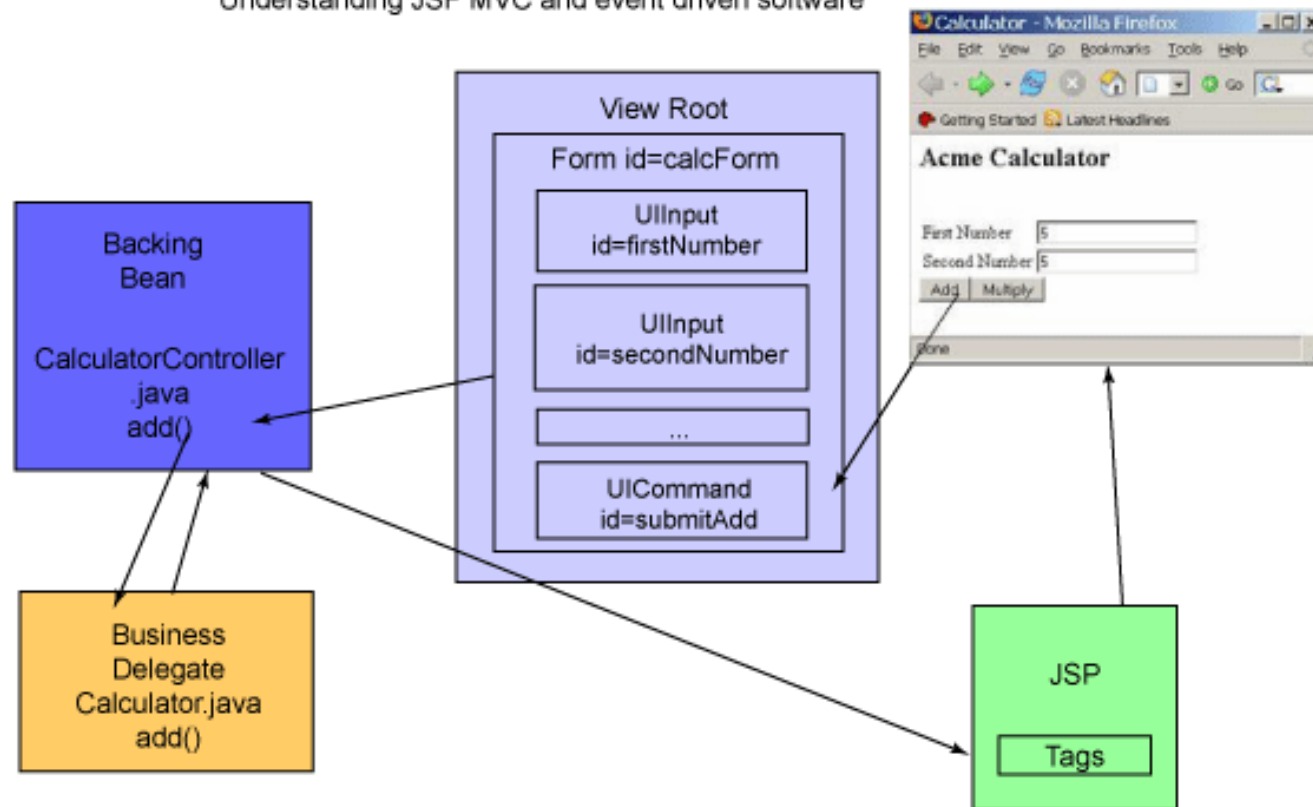
- The model-view-controller (MVC) architecture provides a set of design patterns that help you separate the areas of concern involved in building and running a GUI-based application:
 - The *model* encapsulates the business logic and persistence code for the application. The model should be as view-technology-agnostic as possible. For example, the same model should be usable with a Swing application, a Struts application, or a JSF application.
 - The *view* should display model objects and contain presentation logic only. There should be no business logic or controller logic in the view.
 - The *controller* (along with its attending logic) acts as the mediator between the view and the model. The controller talks to the model and delivers model objects to the view to display. In an MVC architecture the controller always selects the next view.

JSF MVC Implementation

- In JSF's MVC implementation, backing beans mediate between the view and the model. Because of this, it's important to limit the business logic and persistence logic in the backing beans. One common alternative is to delegate business logic to a façade that acts as the model.
- Unlike JSP technology, JSF's view implementation is a stateful component model. The JSF view is comprised of two pieces: the view root and JSP pages.
 - The view root is a collection of UI components that maintain the state of the UI
 - The JSP page binds UI components to JSP pages and allow you to bind field components to properties of backing beans

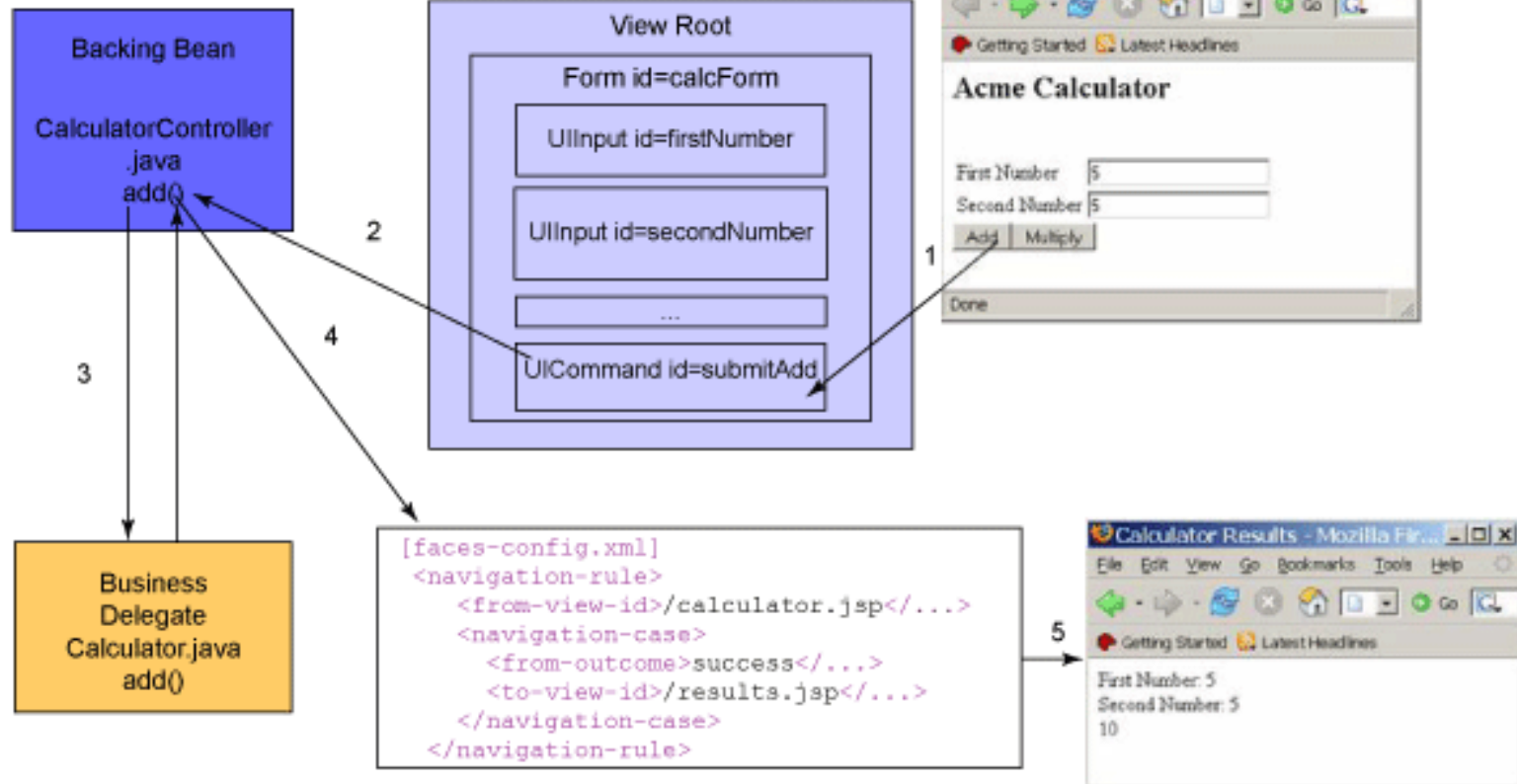
JSF MVC Implementation

Understanding JSF MVC and event driven software



JSF Example

MVC Calculator App





JSF Example

- To build the Calculator application in JSF you'll need to do the following:
 1. Declare the Faces Servlet, and Faces Servlet mapping in the web.xml file.
 2. Declare what beans get managed by JSF in the faces-config.xml file.
 3. Declare the navigation rules in the faces-config.xml file.
 4. Develop the model object Calculator.
 5. Use the CalculatorController to talk to the Calculator model.
 6. Create the index.jsp page.
 7. Create the calculator.jsp page.
 8. Create the results.jsp page.

Declare the Faces Servlet and Servlet mapping

```
<!-- Faces Servlet -->  
<servlet>  
  <servlet-name>Faces Servlet</servlet-name>  
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class> <load-on-startup> 1 </load-on-startup>  
</servlet>
```

```
<!-- Faces Servlet Mapping -->  
<servlet-mapping>  
  <servlet-name>Faces Servlet</servlet-name>  
  <url-pattern>/calc/*</url-pattern>  
</servlet-mapping>
```

The above tells the Faces Servlet container to send all requests that map to /calc/ to the Faces Servlet for processing.

Declare bean management

Next, you will want to declare which beans get used by JSF GUI components. The example application only has one managed bean. It is configured in faces-config.xml as follows:

```
<faces-config>
...
<managed-bean>
<description> The "backing file" bean that backs up the calculator webapp </description>
<managed-bean-name>CalcBean</managed-bean-name>
<managed-bean-class>com.arcmind.jsfquickstart.controller.CalculatorController</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope> </managed-bean>
</faces-config>
```

The above config tells JSF that you want to add a bean to the JSF context called CalcBean. You can call your managed bean anything you want.

Declare navigation rules

For this simple application you need only to establish the navigation path from the calculator.jsp page to the results.jsp page, as shown below.

```
<navigation-rule>  
  <from-view-id>/calculator.jsp</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/results.jsp</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

Develop the model object

```
package com.arcmind.jsfquickstart.model;
```

```
public class Calculator {  
    //add numbers.  
    public int add(int a, int b)  
    {  
        return a + b;  
    }  
  
    // multiply numbers  
    public int multiply(int a, int b)  
    {  
        return a * b;  
    }  
}
```

With that, the business logic is all set up. Your next step is to glue it to the Web application interface.

Gluing the model and the view through the controller

```
package com.arcmind.jsfquickstart.controller;
import com.arcmind.jsfquickstart.model.Calculator;
public class CalculatorController {
    private Calculator calculator = new Calculator();
    private int firstNumber = 0;
    private int result = 0;
    private int secondNumber = 0;

    public CalculatorController() {super(); }
    public void setCalculator(Calculator aCalculator) { this.calculator = aCalculator; }
    public void setFirstNumber(int aFirstNumber) { this.firstNumber = aFirstNumber; }
    public int getFirstNumber() { return firstNumber; }
    public int getResult() { return result; }
    public void setSecondNumber(int aSecondNumber) { this.secondNumber = aSecondNumber; }
    public int getSecondNumber() { return secondNumber; }

    public String add() {
        result = calculator.add(firstNumber, secondNumber);
        return "success";
    }
    public String multiply() {
        result = calculator.multiply(firstNumber, secondNumber);
        return "success";
    }
}
```

Gluing the model and the view through the controller

```
package com.arcmind.jsfquickstart.controller;
import com.arcmind.jsfquickstart.model.Calculator;
public class CalculatorController {
    private Calculator calculator = new Calculator();
    private int firstNumber = 0;
    private int result = 0;
    private int secondNumber = 0;

    public CalculatorController() {super(); }
    public void setCalculator(Calculator aCalculator) { this.calculator = aCalculator; }
    public void setFirstNumber(int aFirstNumber) { this.firstNumber = aFirstNumber; }
    public int getFirstNumber() { return firstNumber; }
    public int getResult() { return result; }
    public void setSecondNumber(int aSecondNumber) { this.secondNumber = aSecondNumber; }
    public int getSecondNumber() { return secondNumber; }

    public String add() {
        result = calculator.add(firstNumber, secondNumber);
        return "success";
    }
    public String multiply() {
        result = calculator.multiply(firstNumber, secondNumber);
        return "success";
    }
}
```

- Notice that the multiply and add methods return "success." The string success signifies a logical outcome. Note that it is not a keyword. You used the string success when specifying navigation rules in faces-config.xml; therefore, after the add or multiply operation is executed the application will forward the user to the results.jsp page.
- With that, you're done with the backing code. Next you'll specify the JSP pages and component trees that represent the application view.

Create the index.jsp page

- The purpose of the index.jsp page in this application is to ensure that the /calculator.jsp page loads in the JSF context so that the page can find the corresponding view root. The index.jsp page looks as follows:

```
<jsp:forward page="/calc/calculator.jsp" />
```

- All this page does is redirect the user to calculator.jsp under the "calc" Web context. This puts the calculator.jsp page under the JSF context, where it can find its view root.

Create the calculator.jsp page

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>

<f:view>

<h:form id="calcForm">

<h:panelGrid columns="3">
    <h:outputLabel value="First Number" for="firstNumber" />
    <h:inputText id="firstNumber" value="#{CalcBean.firstNumber}" required="true" />
    <h:message for="firstNumber" /> <h:outputLabel value="Second Number" for="secondNumber" />
    <h:inputText id="secondNumber" value="#{CalcBean.secondNumber}" required="true" />
    <h:message for="secondNumber" />
</h:panelGrid>

<h:panelGroup>
    <h:commandButton id="submitAdd" action="#{CalcBean.add}" value="Add" />
    <h:commandButton id="submitMultiply" action="#{CalcBean.multiply}"
        value="Multiply" />
</h:panelGroup>

</h:form>

</f:view>
```


Create the results.jsp page

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

```
<f:view>
```

```
First Number: <h:outputText id="firstNumber" value="#{CalcBean.firstNumber}"/>
```

```
<br />
```

```
Second Number: <h:outputText id="secondNumber" value="#{CalcBean.secondNumber}"/>
```

```
<br />
```

```
Result: <h:outputText id="result" value="#{CalcBean.result}"/>
```

```
<br />
```

```
</f:view>
```

- This results.jsp file is a relatively simplistic page that displays the addition results to the user. It accomplishes this through the <outputText> tag.
- The <outputText> tag takes an id and value attribute. The value attribute outputs the bean value as a string when rendered.
- The value attribute uses JSF to bind the output value to your backing bean properties (namely, firstNumber, secondNumber, and result).

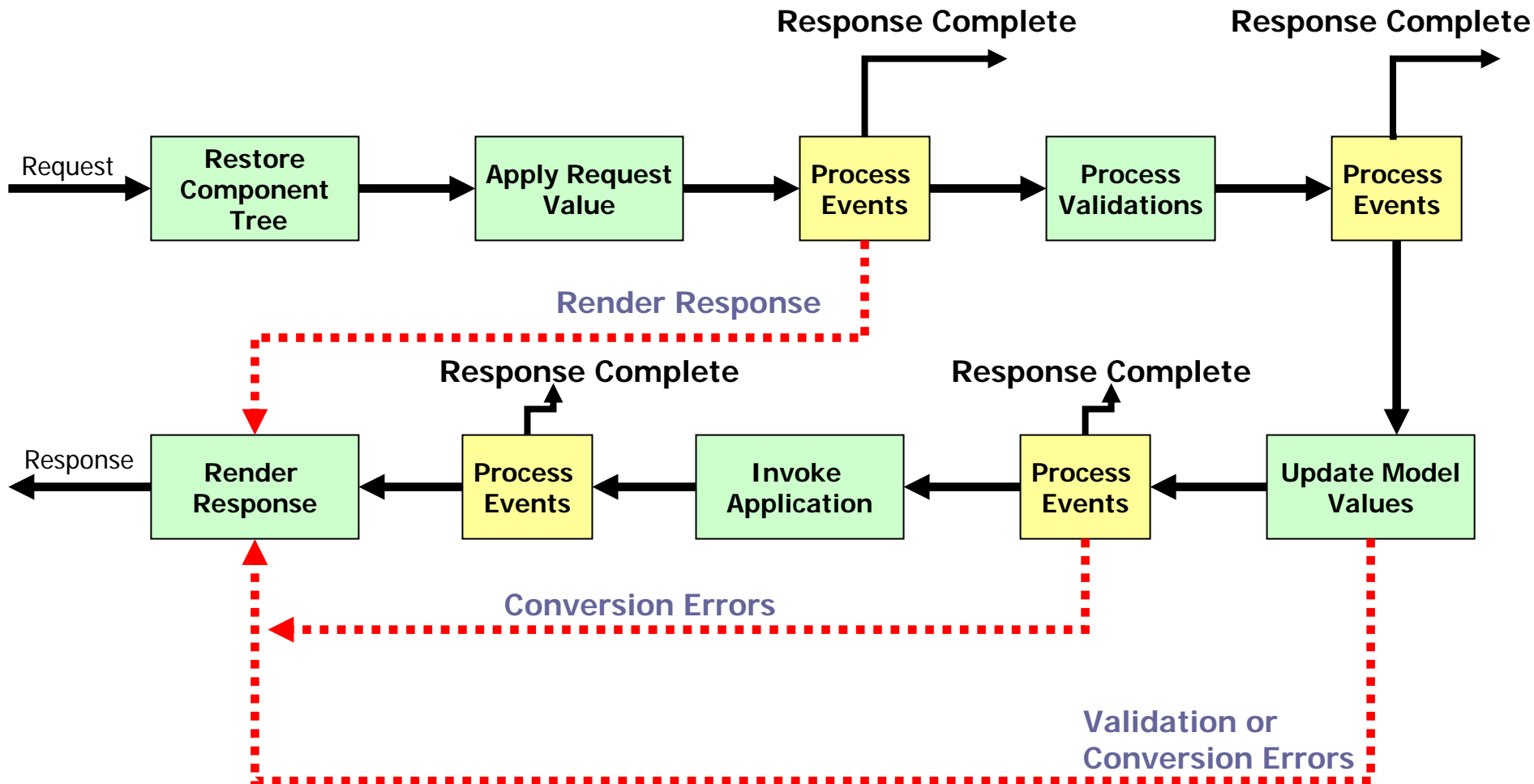


Demo with NetBeans 5.5 and the Visual Web Pack

Read Visual Web Pack documentation for details with more emphasis of the following:

- Layout: Property Sheets, JSF Fragments
- Data Binding: **CachedRowSet,**
CachedRowSetDataProvider,
ObjectListDataProvider

JSF Request Processing Lifecycle



JSF - Request Processing Lifecycle

- Restore component tree
 - The controller examines the request and extracts the view ID, which is determined by the name of the JSP page. If the view doesn't already exist, the JSF controller creates it. If the view already exists, the JSF controller uses it. The view contains all the GUI components.
- Apply Request Values
 - The purpose of the *apply request values* phase is for each component to retrieve its current state. Component values are typically retrieved from the request parameters.
- Process Validations
 - At this stage, each component will have its values validated against the application's validation rules.
- Update Model
 - updates the actual values of the server-side model -- namely, by updating the properties of your backing beans.



JSF - Request Processing Lifecycle

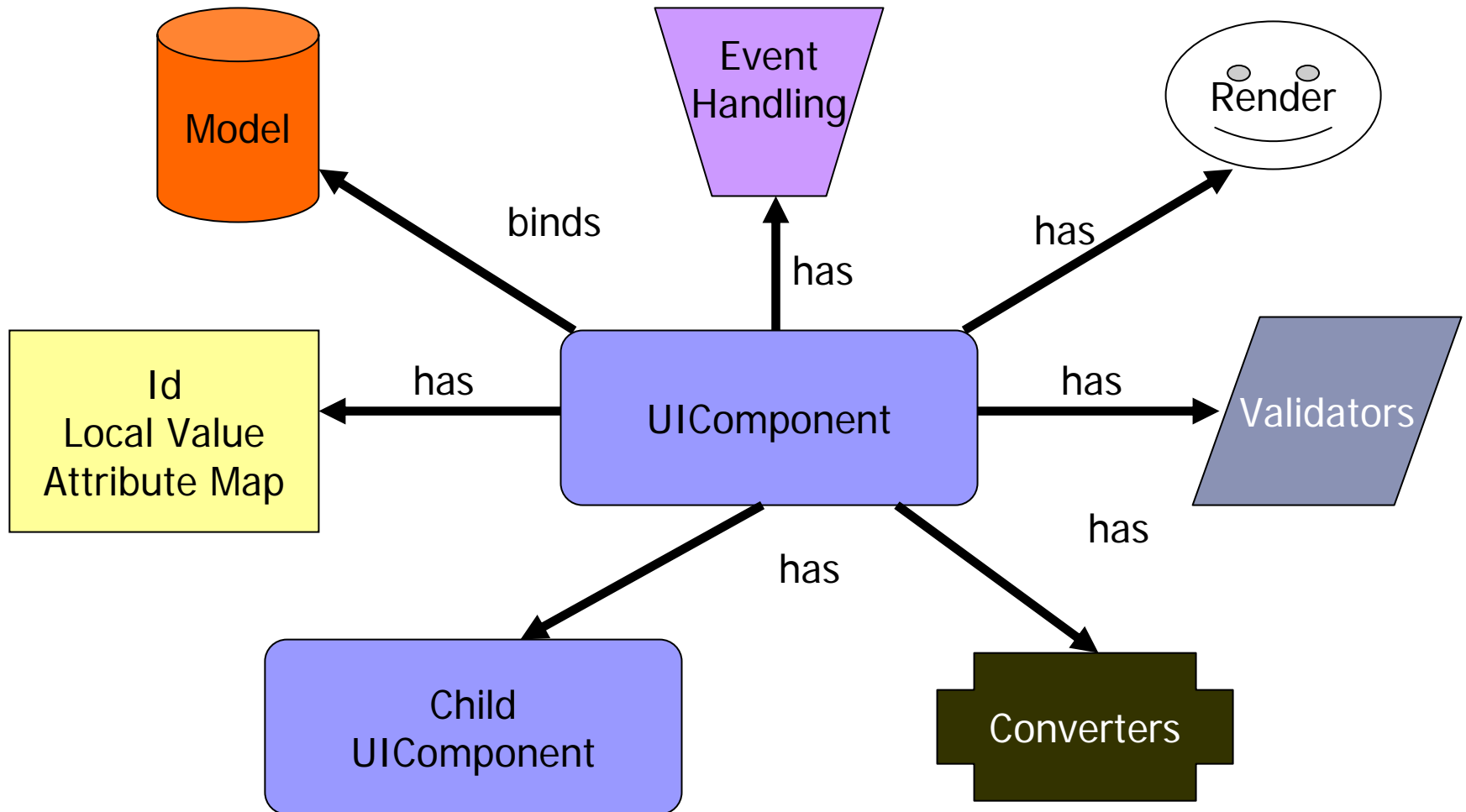
■ Invoke Application


- ☐ The JSF controller invokes the application to handle Form submissions. The component values will have been converted, validated, and applied to the model objects, so you can now use them to execute the application's business logic.

■ Render Response

- ☐ you display the view with all of its components in their current state.
- ☐ Render the page and send it back to client

JSF Component





JSF Standard UI Components

- UIInput
- UIOutput
- UISelectBoolean
- UISelectItem
- UISelectMany
- UISelectOne
- UISelectMany
- UIGraphic
- UICommand
- UIForm
- UIColumn
- UIData
- UIPanel



JSF HTML Tag Library

- JSF Core Tag Library

- Validator, Event Listeners, and Converters

- JSF Standard Library

- Express UI components in JSP

JSF HTML Tag Library

<f:view>

```
<h:form id="logonForm">
```

```
  <h:panelGrid columns="2">
```

```
    <h:outputLabel for="username">
```

```
      <h:outputText value="Username:"/>
```

```
    </h:outputLabel>
```

```
    <h:inputText id="username" value="#{logonBean.username}"/>
```

```
    <h:outputLabel for="password">
```

```
      <h:outputText value="Password:"/>
```

```
    </h:outputLabel>
```

```
    <h:inputSecret id="password" value="#{logonBean.password}"/>
```

```
    <h:commandButton id="submitButton" type="SUBMIT"
```

```
      action="#{logonBean.logon}"/>
```

```
    <h:commandButton id="resetButton" type="RESET"/>
```

```
  </h:panelGrid>
```

```
</h:form>
```

```
</f:view>
```

JSF HTML Tag Library

```
<table>
```

```
  <tr><td>Food Selection:</td></tr>
```

```
  <tr><td>
```

```
    <h:selectManyCheckBox value="#{order.foodSelections}">
```

```
      <f:selectItem itemValue="z" itemLabel="Pizza" />
```

```
      <f:selectItem itemValue="fc" itemLabel="Fried Chicken" />
```

```
      <f:selectItem itemValue="h" itemLabel="Hamburger" />
```

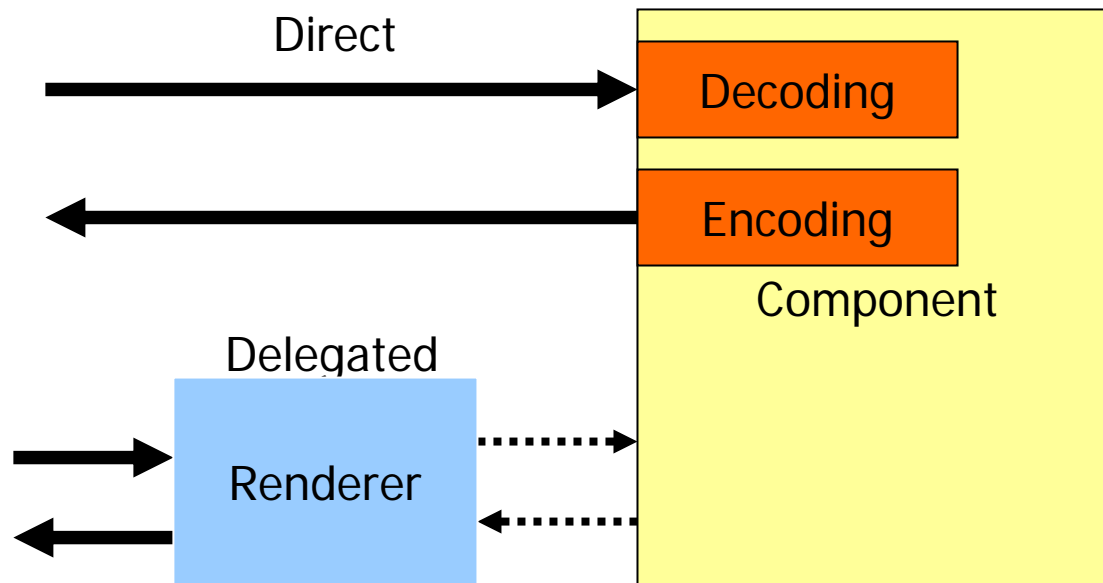
```
    </h:selectManyCheckBox>
```

```
  </td>
```

```
</table>
```

JSF Rendering Model

Two Rendering Models (direct or delegated)



Pluggable look and feel



JSF Rendering Model

- Render kit consists of a set of renders
- JSF reference implement must provide a render kit for all the standard UI components to generate HTML 4.01
- Custom render kit can be provided to render UI components into a specific markup language



JSF – Managed Bean

- Use to separate presentation from business logic
- Based on JavaBeans
- Use the declarative model
- Entry point into the model and event handlers

JSF – Value Binding

- Bind component value and attribute to model objects

Literal:

```
<h:outputText rendered="true" value="$1000.00"/>
```

Value Binding:

```
<h:outputText rendered="#{user.manager}"  
  value="#{employee.salary}"/>
```

JSF – Value Binding

- Value binding expression
 - Bean properties
 - List
 - Array
 - Map
 - Predefine objects- header, header values, request parameters, cookie, request/session/application scope attributes, initial parameters

JSF – Value Binding Expression

```
<h:outputText value="#{user.name}" />
```

```
<h:outputText value="Hello There #{user.name}" />
```

`#{!user.manager}` – operator (+, -, *, /, % ...)

Faces-config.xml

```
<managed-bean>
```

```
  <managed-bean-name>user</managed-bean-name>
```

```
  <managed-bean-class>org.User</managed-bean-class>
```

```
  <managed-bean-scope>session</managed-bean-scope>
```

```
</managed-bean>
```

Support bean property initialization for primitive data type as well as List and Map.

Binding expression can be used at the bean configuration as well

JSF – Predefined Objects

Variable	Meaning
header	A Map of HTTP headers, only the first value of for each name
headerValues	A Map of HTTP headers, String[] of all values for each name
param	A Map of HTTP request parameters, first value of for each name
paramValues	A Map of HTTP headers, String[] of all values for each name
cookie	A Map of cookie name and values
initParam	A Map of initialization parameters
requestScope	A Map of all request scope attributes
sessionScope	A Map of all session scope attributes
applicationScope	A Map of all request scope attributes
facesContext	FacesContext instance of this request
view	The UIViewRoot instance of this request

JSF – Method Binding

- Binding an event handler to a method

`<h:commandButton action="#{user.login}" />`

- Four component attributes:

- ☐ Action
- ☐ Action listener
- ☐ Value change listener
- ☐ Validator



JSF Events

- Events are fired by each UI component
- Event handlers are registered with each component

JSF Events – Value Change Event

Value Changed Listener:

```
<h:inputText id="maxUsers"  
    valueChangeListener="#{user.checkMaxUser}" />
```

```
public void checkMaxUser(ValueChangeEvent evt) {  
    evt.getNewValue();    // new value  
    evt.getOldValue();    // old value  
}
```

JSF Events – Action Event

Action Listener:

```
<h:commandButton value="Login"  
    actionListener="#{customer.loginActionListener}"  
    action="#{customer.login}" />
```

```
public void loginActionListener(ActionEvent e) {  
  
}
```

```
public String login() {  
    return "OK";  
    // return "FAIL";  
}
```



JSF Events – Listener vs. Action

■ Listener Handlers

- ☐ Implement UI logic
- ☐ Have access to event source
- ☐ Do not participate in navigation handling

■ Action Handlers

- ☐ Implement business logic
- ☐ Don't have access to action source
- ☐ Returned outcome affects the navigation handling

JSF – Multiple Event Handlers

```
<h:selectOneMenu value="#{customer.country}"  
    <f:valueChangeListener type="com.comp.CntrListener"  
    <f:valueChangeListener type="com.comp.CCListener"  
</h:selectionOneMenu>
```

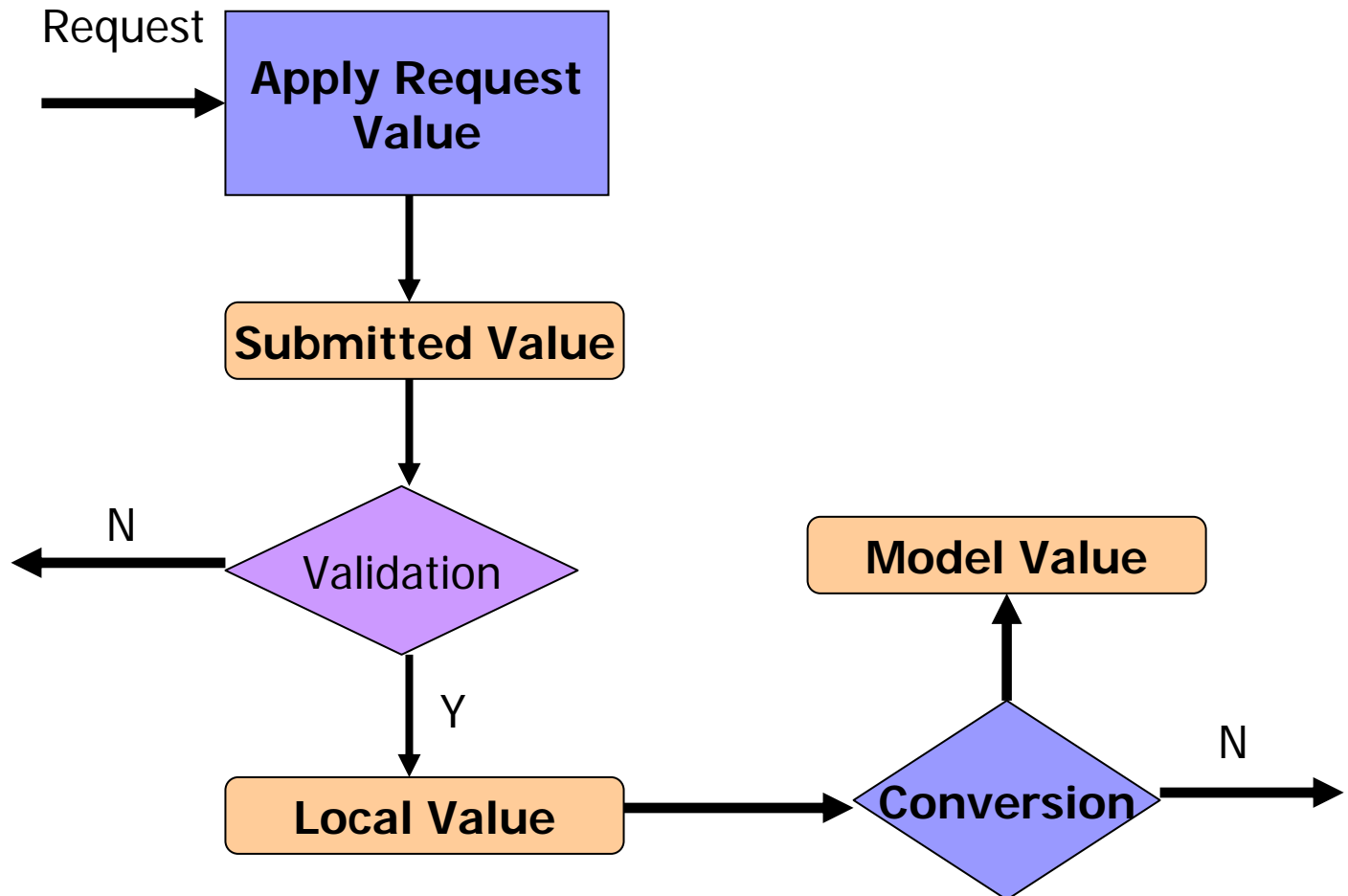
```
<h:commandButton action="#{search.doSearch()}">  
    <f:actionListener type="com.comp.AAciontListener" />  
    <f:actionListener type="com.comp.BActionListener" />  
</h:commandButton>
```



JSF Validators

- For validating user input
- 0 or more validators can be registered with a UIInput component
- Validators are invoked during the *Process Validations* request processing phase
- Standard validators and custom validator

JSF – Two Step Approach





JSF Validators

- DoubleRangeValidator

- Any numeric type, between specified maximum and minimum values

- LongRangeValidator

- Any numeric type convertible to long, between specified maximum and minimum values

- LengthValidator

- String type, between specified maximum and minimum values

JSF Validators

Required Validation Example:

```
<h:inputText value="#{user.id}" required="true" />
```

Length Validation Example:

```
<h:inputText value="#{user.password}" >  
    <f:validateLength minimum="6" />  
    <f:validator validatorId="passwordValidator" />  
</h:inputText>
```



JSF Converters

- Type conversion between server-side objects and their representation in markup language
- Standard converter implementations
 - DateTime
 - Number

JSF Converters

Number converter example:

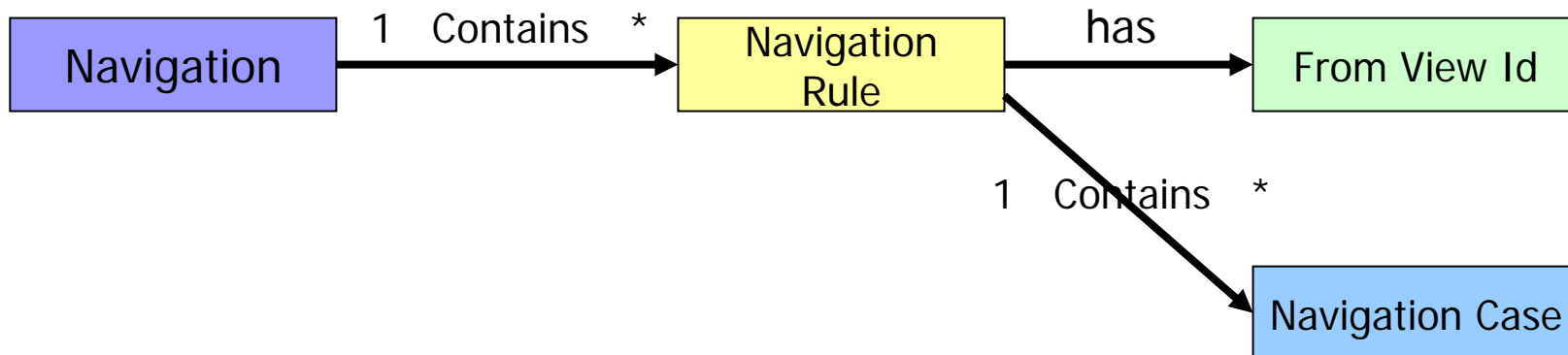
```
<h:inputText value="#{rent.amt}" converter="Number">  
  <f:attribute name="numberStyle" value="currency" />  
</h:inputText>
```

Date convert example:

```
<h:inputText value="#{rent.dueDate}" converter="DateFormat">  
  <f:attribute name="formatPattern" value="MM/DD" />  
</h:inputText>
```

JSF Navigation

- JSF provides a default navigational handler
- Behavior is configured in configuration file (faces-config.xml)



JSF Navigation - Example

```
<navigation-rule>
  <description>LOGIN PAGE NAVIGATION HANDLING</description>
  <from-view-id> /login.jsp </from-view-id>

  <navigation-case>
    <description>Handle case where login succeeded.</description>
    <display-name>Successful Login</display-name>
    <from-action>#{userBean.login}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/home.jsp</to-view-id>
  </navigation-case>

  <navigation-case>
    <description>User registration for a new user succeeded.</description>
    <display-name>Successful New User Registration</display-name>
    <from-action>#{userBean.register}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/welcome.jsp</to-view-id>
  </navigation-case>

</navigation-rule>
```

JSF – Error Handling

- `javax.faces.application.FacesMessage`
 - Information, Warning, Error, Fatal
- Contain summary and detail
- `<h:messages>` - to display all messages
- `<h:message>` - to display a single message for a particular component
- `javax.faces.context.FacesContext.addMessage(String clientId, FacesMessage)`

JSF - HTML & CSS Integration

■ HTML Integration

- Pass-through attributes

```
<h:inputText size="5" onblur="checkValue();" />
```

■ Stylesheets Integration

- Most HTML tags have one or more attributes (style, styleClass) for passing style information

```
<h:outputText styleClass="header" value="#{bundle.welcome}" />
```

- For data table

```
<h:dataTable rowClasses="odd, even",  
  columnClasses="columnOne, columnTwo" ..
```



References

- JavaServer Faces 1.0 (JSR-127) Tour, Hien Luu, Neoforma
- www.Netbeans.org, Visual Web Pack Documentation