JavaServer Faces (JSF) Basics
Topics

? What is and why JSF?
? Quick overview on JSF architecture, concepts, and features
? Developer roles (in Web app development)
? Request processing life cycle
? UI Component model
? Using JSF tag libraries
  - Core tags, HTML tags
? Backing beans (model beans)
? Page navigation
Advanced Topics (We will talk about these in Advanced JSF Session)

- Backing bean (model objects) management
- Navigations
- Event and listeners
- Validation
- Rendering
- Conversion
- Internationalization and Localization
- Custom UI component
What is & Why JSF?
JavaServer™ Faces (JSF) Framework Is…

A server side user interface component framework for Java™ technology-based web applications
What is JSF?

A specification and reference implementation for a web application development framework

- Components
- Events
- Validators
- Back-end-data integration

Designed to be leveraged by tools
- Sun Java Studio Creator
Why JSF? (page 1)

- MVC for web applications
- Clean separation of roles
- Easy to use
- Extensible Component and Rendering architecture
- Support for client device independence
- Standard
- Huge vendor and industry support
Why JSF? (page 2)

- Offers finer-grained separation of behavior and presentation than JSP
  - Component-specific event handling
  - UI elements as stateful objects on the server
- UI-component and Web-tier concepts without limiting you to a particular scripting technology or markup language
  - Can work with any presentation technology including JSP
Why JSF? (page 3)

? JSP and Servlet
  - No built-in UI component model

? Struts (I am not saying you should not use Struts)
  - No built-in UI component model
  - No built-in event model for UI components
  - No built-in state management for UI components
  - No built-in support of multiple renderers (Struts is more or less tied up with HTML)
  - Not a standard (despite its popularity)

? Struts and JSF can be used together
JSF is a UI Framework for Java Web Applications

Client

request (events)

Response (markup)

Server

UI
JSF Design Goals
JavaServer Faces Must Be ...

- Tool friendly
- Client device / protocol neutral
- Usable with JavaServer Pages (JSP)
- Usable without JSP
- Useful in the context of HTML and today's browsers
- Scalable
Our Requirements

- MVC
- UI components
- State management
- Events
- Validation and error handling
- Lightning fast performance
How the JSF Specification Fits In

- JSF App
- JSF Tags
- JSP (1.2)
- Servlets (2.3)
- JSF API
Quick Overview on JSF Architecture, Concept, & Features
JSF Architecture

Server

JSF Page

HTML RenderKit

App Backend

JSF Page

WML RenderKit

Front ctrl

Desktop Browser

HTML

Phone

WML
Important Basic Capabilities

- Extensible UI component model
- Flexible rendering model
- Event handling model
- Validation framework
- Basic page navigation support
- Internationalization
- Accessibility
Key JSF Concepts

- **UIComponent**
  - Render-independent characteristics
  - Base class with standard behaviors
- **Standard UIComponent Subclasses:**
  - UICommand, UIForm, UIGraphic, UIInput, UIOutput, UIPanel, UISelectBoolean, UISelectMany, UISelectOne
- **FacesEvent** – Base class for request and application events
- **Validator** – Base class for standard and application defined validators
Key JSF Concepts

- **Converter** – Plug-in for String-Object conversion
- **FacesContext** – Per-Request State:
  - Servlet request, response, session
  - JSF request, response trees
  - Model reference expression evaluators
    - Syntax similar to the expression language of the JSP Standard Tag Library (JSTL) 1.0
    - Primary interface between components and the data provided by (or to) the application
Key JSF Concepts

- **Renderer** – Converts components to and from a specific markup language
  - Supports render-dependent attributes on components
  - May support more than one component type

- **RenderKit** – Library of Renderers
  - Extensible at runtime
  - Basic HTML RenderKit is part of the specification
Relationship to Other JSRs

- JSF is based on:
  - Servlet 2.3 (JSR-53)
  - JSP 1.2 (JSR-53)

- JSF must be synergistic with:
  - JSTL 1.0 (JSR-52)
  - Portals (JSR-168)

- JSF is not part of J2EE 1.4 standard yet
  - Will be considered for J2EE 1.5
  - It is included in J2EE 1.4 SDK, however
Developer Roles
JSF Developer Roles

Application

Page Author
Application Developer

Component Developer

Extensions

Tools Developer
JSF Implementor/Extender
Roles Definition

- **Page Author** – Creates the user interface of a web application
  - Familiar with markup language(s) to be used
  - Assembler of prebuilt components
  - Uses “Drag and drop” IDE like Sun Java Studio Creator

- **Component Writer** – Creates reusable components, renderers, and libraries
  - Components – Render-independent properties
  - Renderers – Render-dependent properties
Roles Definition

Application Developer – Creates the server-side functionality of a web application not directly related to the user interface

- Business logic components implemented in standard J2EE ways (EJBs, JavaBeans, Connectors)
- Persistence tier components implemented in standard J2EE ways (EJBs, JDBC, Connectors)
- Model data exposed to user interface via JavaBean programming model
- Validator, Convertor, Event handler
Roles Definition

(tool provider) Tool Provider – Creates tools to assist page authors, component writers, and application developers
- GUI-oriented page development tools
- IDEs to facilitate creation of components
- Application generators (from high level description)
- Web application frameworks that utilize JSF components for their user interface
- Example: Sun Java Studio Creator

(JSF implementor) JSF Implementor – Provides runtime environment to execute JSF webapps
- J2EE SDK 1.4
Request Processing
Life Cycle
Lifecycle of JSF Page

- A JSF page is represented by a tree of UI components, called a view.
- When a client makes a request for the page, the lifecycle starts.
- During the lifecycle, JSF implementation must build the view while considering state saved from the previous postback.
- When the client performs a postback of the page, JSF implementation must perform lifecycle steps:
  - validation
  - conversion
Request Processing Lifecycle

1. Reconstitute Component Tree
2. Apply Request Values
3. Process Events
4. Process Validations
5. Process Events
6. Update Model Values
7. Invoke Application
8. Process Events
9. Validation / Conversion Errors / Render Response
10. Response Complete
11. Render Response
12. Response Complete
13. Render Responder
14. Process Events
15. Response Complete
16. Faces Response
17. Faces Request
Request Processing

- Life-cycle handles two types of requests
  - Initial request & Postback

- Initial request
  - A user requests the page for the first time
  - Lifecycle only executes the restore view and render response phases

- Postback
  - A user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request
  - Lifecycle executes all phases
Request Processing Lifecycle Phases

1. Reconstitute component tree phase
2. Apply request values phase
3. Process validations phase
4. Update model values phase
5. Invoke application phase
6. Render response phase
greeting.jsp (from guessNumber)

<HTML>
<HEAD> <title>Hello</title> </HEAD>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<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:graphicImage id="waveImg" url="/wave.med.gif" />
<h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
<f:validateLongRange minimum="0" maximum="10" />
</h:inputText>
<h:commandButton id="submit" action="success" value="Submit" />
<p>
<h:message style="color: red; font-family: 'New Century Schoolbook', serif;
font-style: oblique; text-decoration: overline" id="errors1" for="userNo"/>
</h:form>
</f:view>
</HTML>
Phase 1: Reconstitute Component Tree (View) Phase

When a request for a JavaServer Faces page is made, such as when clicking on a link or a button, the JSF implementation begins the Restore view phase

- if it is the first time, an empty view is created

JSF implementation performs

- builds the view of the JavaServer Faces page
- wires up event handlers and validators
- saves the view in the FacesContext
Example: Component Tree (View) of greeting.jsp page of GuessNumber

- UIView
  - helloForm
    - userNo
    - submit
    - errors1
Phase 2: Apply Request Values

Phase

Each component in the tree extracts its new value from the request parameters with its built-in `decode method`

Within the decode method, the value is then converted to right type then stored locally on the component

- For `userNo` component in `greeting.jsp` page, type gets converted from String to Integer
- Conversion errors are queued on the `FaceContext`
Phase 3: Process Validation Phase

- JSF implementation processes all input validations registered on the components in the tree
  - This is input validation (not business logic validation)
- In case of validation errors
  - Error messages are queued in FacesContext
  - Lifecycle advances directly to the Render Response phase
- Example
  - userNo has be be between 1 and 10
Phase 4: Update Model Values Phase

JSF implementation walks the component tree and sets the corresponding server side object properties to the components' local values

- Update the bean properties pointed at by an input component's value attribute
- Type conversion from local value to model property type

Example in greeting.jsp

- `userNumber` property of the `UserNumberBean` is set to the local value of the `userNo` component
Phase 5: Invoke Application Phase

JSF implementation handles any application-level events, such as submitting a form or linking to another page.
Phase 5: Example in GuessNumber

- The `greeting.jsp` page from the guessNumber example has one application-level event associated with the UICommand component.
- A default `ActionListener` implementation retrieves the outcome, "success", from the component's action attribute.
- The listener passes the outcome to the default `NavigationHandler`.
- The `NavigationHandler` matches the outcome to the proper navigation rule defined in the application's application configuration file to determine what page needs to be displayed next.
- JSF implementation then sets the response view to that of the new page.
Phase 6: Render Response Phase

- JSF implementation invokes the components' built-in encode method and renders the components from the component tree saved in the FacesContext
  - Create appropriate markup from component tree
  - If errors in previous phases, original page is rendered with any queued error messages
- State of the response is saved so that subsequent requests can access it and it is available to the Restore View phase
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, for example
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: 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

```
<table>
<thead>
<tr>
<th>Tag</th>
<th>Rendered as</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_button</td>
<td>Figure 20-4</td>
</tr>
<tr>
<td></td>
<td>Login</td>
</tr>
<tr>
<td></td>
<td>Login Button</td>
</tr>
<tr>
<td>command_link</td>
<td>Figure 20-5</td>
</tr>
<tr>
<td></td>
<td>hyperlink</td>
</tr>
<tr>
<td></td>
<td>A Hyperlink</td>
</tr>
</tbody>
</table>
```
greeting.jsp

<f:view>
  <f:form id="helloForm">
    <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"/>
      </p>
    </h:form>
  </f:form>
</f:view>
</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>
</f:view>
</HTML>
Application Configuration
Application Configuration File

- XML file for configuring resources required at application startup time
  - navigation rules, converters, validators, render kits
- Usually named as faces-config.xml
- A <faces-config> tag must enclose all of the other declarations
  <faces-config>
    ....
  </faces-config>
faces-config.xml of guessNumber

<?xml version="1.0"?>

<!--
    Copyright 2003 Sun Microsystems, Inc. All rights reserved.
    SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 -->

<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN" "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>

<application>
    <locale-config>
        <default-locale>en</default-locale>
        <supported-locale>de</supported-locale>
        <supported-locale>fr</supported-locale>
        <supported-locale>es</supported-locale>
    </locale-config>
</application>

</faces-config>
faces-config.xml of guessNumber

<navigation-rule>
    ...
    <from-view-id>/greeting.jsp</from-view-id>
    ...
</navigation-rule>

<navigation-rule>
    ...
    <from-view-id>/response.jsp</from-view-id>
    ...
</navigation-rule>

<managed-bean>
    ...
    <managed-bean-name>UserNumberBean</managed-bean-name>
    ...
</managed-bean>

</faces-config>
Application Configuration File

? You can have more than one application configuration file

? There are three ways that you can make these files available to the application:

- A resource named /META-INF/faces-config.xml in any of the JAR files in the Web application's /WEB-INF/lib directory
- A context init parameter, javax.faces.application
- A resource named faces-config.xml in the /WEB-INF/ directory of your application (most common)
Application Class

- When an application starts up, the JSF implementation creates a single instance of the Application class
- Is automatically created for each application
- FacesContext.getApplication()
Using JSF Tag Libraries
Two Tag Libraries

- **html_basic**
  - Defines tags for representing common HTML user interface components

- **jsf_core**
  - Defines other JSF related tags
  - Independent of any rendering technology

- JSP page need to declare them
  
  `<%@ taglib uri="http://java.sun.com/jsf/html/" prefix="h" %>`
  `<%@ taglib uri="http://java.sun.com/jsf/core/" prefix="f" %>`
<f:view> element

- Represents UIViewRoot component
- All component tags on the page must be enclosed in the view tag

  <f:view>
  ... other faces tags, possibly mixed with other content ...
  </f:view>

- Optional locale attribute
  - Overrides the Locale stored in the UIViewRoot
Nested View's

Use `<f:subview>` element in order to include a JSF page inside another JSP page

```jsp
<f:subview>
  jsp:include page="theNestedPage.jsp"
</f:subview>
```
Using Core Tags
Event Handling Tags and Attributes

? `<f:actionListener>` or actionListener attribute
  - Registers an action listener on a component

? `<f:valueChangeListener>` or valueChangeListener attribute
  - Registers a value-change listener on a parent component
Example: `<f:valueChangeListener>` in `customerInfo.jsp` (carstore)

```
<h:inputText id="firstName" value="#{customer.firstName}" required="true">
    <f:valueChangeListener type="carstore.FirstNameChanged" />
</h:inputText>
```
Example: actionListener attribute in chooseLocale.jsp (carstore)

```xml
<h:commandButton id="NAmerica" action="storeFront"
    value="#{bundle.english}"
    actionListener="#{carstore.chooseLocaleFromLink}"
>
</h:commandButton>
```
Attribute Configuration Tags

? <f:attribute>
  - Adds configurable attributes to a parent components
Data Conversion Tags

? <f:converter>
  - Registers an arbitrary converter on the parent component

? <f:convertDateTime>
  - Registers a DateTime converter instance on the parent component

? <f:convertNumber>
  - Registers a Number converter instance on the parent component
Facet Tag

? <f:facet>
- Signifies a nested component that has a special relationship to its enclosing tag
Parameter Substitution Tag

? <f:parameter>
  - Substitutes parameters into a MessageFormat instance and to add query string name/value pairs to a URL
Tags for Representing Items in a List

- `<f:selectItem>`
  - Represents one item in a list of items in a UISelectOne or UISelectMany component

- `<f:selectItems>`
  - Represents a set of items in a UISelectOne or UISelectMany component
Container Tags

`<f:subview>`

- Contains all JavaServer Faces tags in a page that is included in another JavaServer Faces page
Validator Tags

- `<f:validateDoubleRange>`
  - Registers a DoubleRangeValidator on a component

- `<f:validateLength>`
  - Registers a LengthValidator on a component

- `<f:validateLongRange>`
  - Registers a LongRangeValidator on a component

- `<f:validator>`
  - Registers a custom Validator on a component
Output Tags

- `<f:verbatim>`
  - Generates a UIOutput component that gets its content from the body of this tag
Using HTML Tags
HTML Tags

- Used to control display data or accept data from the user

- Common attributes
  - id: uniquely identifies the component
  - value: identifies an external data source mapped to the component's value
  - binding: identifies a bean property mapped to the component instance
Using HTML Tags
UIForm & <h:form>
UIForm & <h:form> tag

? UIForm UI component
   - An input form with child components representing data that is either presented to the user or submitted with the form

? Encloses all of the controls that display or collect data from the user

? Include HTML markup to layout the controls on the page
   - <h:form> tag itself does not perform any layout
Using HTML Tags

UICommand &

<h:commandButton>
UICommand &<h:commandButton>

- UICommand component performs an action when it is activated
  - Most common renderers are Button and Link
UICommand &
<h:commandButton>

? Additional attributes

- **action:**
  
  ? is either a logical outcome String or a JSF EL expression that points to a bean method that returns a logical outcome String
  
  ? In either case, the logical outcome String is used by the navigation system to determine what page to access when the UICommand component is activated

- **actionListener:**
  
  ? is a JSF EL expression that points to a bean method that processes an ActionEvent fired by the UICommand component
Example 1: `<h:commandButton>` from `carDetail.jsp`

```html
<h:commandButton action="#{carstore.buyCurrentCar}"
    value="#{bundle.buy}" />
```

- **action attribute**
  - references a method on the `CarStore` backing bean that performs some processing and returns an outcome
  - outcome is passed to the default NavigationHandler, which matches the outcome against a set of navigation rules defined in the application configuration file.

- **value attribute**
  - references the localized message for the button's label
  - bundle part of the expression refers to the `ResourceBundle` that contains a set of localized messages

04/29/2004
Example1: buyCurrentCar() method of CarStore.java

```java
public class CarStore extends Object {
    ...
    public String buyCurrentCar() {
        getCurrentModel().getCurrentPrice();
        return "confirmChoices";
    }
    ...
}
```
Example 1: Navigation rule for “confirmChoices” in faces-config.xml

```xml
<navigation-rule>
  <from-view-id>/carDetail.jsp</from-view-id>
  <navigation-case>
    <description>
      Any action that returns "confirmChoices" on carDetail.jsp should cause navigation to confirmChoices.jsp
    </description>
    <from-outcome>confirmChoices</from-outcome>
    <to-view-id>/confirmChoices.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```
Example1: Resources.properties file of carstore

sunroofLabel=Sunroof
cruiseLabel=Cruise Control
driverLabel=Driver
keylessLabel=Keyless Entry
securityLabel=Security System
skiRackLabel=Ski Rack
towPkgLabel=Tow Package
gpsLabel=GPS
buy=Buy
back=Back
buyLabel=Thanks for stopping by!
Duke's Stripped-Down Jalopy

If you're the type who doesn't care what anyone thinks, this is the car for you. Strictly for point-a-to-point-b types.

Base Price: 10750
Your Price: 10750

Buy

Options and Packages

Custom  Standard  performance  Deluxe

Engine: V4
Brakes: disc  drum
Suspension: regular
Speakers: 4  6
Audio: Standard  premium
Transmission: manual
Other Options

- Sunroof
- Cruise Control
- Keyless Entry
- Security System
- Ski Rack
- Tow Package
- GPS

Recalculate  Buy
Welcome to CarStore - Netscape

Car Demo

Duke's Stripped-Down Jalopy

You have chosen the following options:

- Engine: V4
- Brakes: Disc
- Suspension: Regular
- Speakers: 4
- Audio: Standard
- Transmission: Manual
- Sunroof: false
- Cruise Control: false
- Keyless Entry: false
- Security System: false
- Ski Rack: false
- Tow Package: false
- GPS: false

Your Price 11500

Buy Back

Documentation for this demo
Example2: `<h:commandButton>
    id="Custom"
    value="#{bundle.Custom}"
    styleClass="#{carstore.customizers.Custom.buttonStyle}"
    actionListener="#{carstore.choosePackage}" />

optionsPanel.jsp is “include'd” inside carDetails.jsp
Duke's Stripped-Down Jalopy

If you're the type who doesn't care what anyone thinks, this is the car for you. Strictly for point-a-to-point-b types.

Base Price 10750
Your Price 11500

Buy

Options and Packages

- Custom
- Standard
- Performance
- Deluxe

Engine

Brakes
- disc
- drum

Suspension
- regular
Example 2: choosePackage() method of CarStore.java

public class CarStore extends Object {
    ...
    public void choosePackage(ActionEvent event) {
        String packageName = event.getComponent().getId();
        choosePackage(packageName);
    }
    public void choosePackage(String packageName) {
        // Business logic processing
    }
    ...
}
Using HTML Tags

UIInput & UIOutput
UIInput & UIOutput Components

- UIInput component displays a value to a user and allows the user to modify this data
  - The most common example is a text field
- UIOutput component displays data that cannot be modified
  - The most common example is a label
- Conversions can occur
- Both UIInput and UIOutput components can be rendered in several different ways
UIInput Component and Renderer Combinations

- inputHidden
  - Allows a page author to include a hidden variable in a page

- inputSecret
  - Accepts one line of text with no spaces and displays it as a set of asterisks as it is typed

- inputText
  - Accepts a text string of one line

- inputTextarea
  - Accepts multiple lines of text
UIOutput Component and Renderer Combinations

- **outputLabel**
  - Displays a nested component as a label for a specified input field

- **outputLink**
  - Display an `<a href>` tag that links to another page without generating an ActionEvent

- **outputMessage**
  - Displays a localized message

- **outputText**
  - Displays a text string of one line
Attributes of `<h:inputText>` and `<h:outputText>`

- id
- value
- converter
- validator
  - JSF EL expression pointing to a backing-bean method that performs validation on the component's data
- valueChangeListener
  - a JSF EL expression that points to a backing-bean method that handles the event of entering a value in this component
Example: `<h:inputText>` in `customerInfo.jsp`

```java
<h:inputText value="#{customer.lastName}" />
```
Please fill in your name and address.

<table>
<thead>
<tr>
<th>Title</th>
<th>Mr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Name</td>
<td>Sang</td>
</tr>
<tr>
<td>Middle Initial</td>
<td>C</td>
</tr>
<tr>
<td>Last Name</td>
<td>Shin</td>
</tr>
<tr>
<td>Mailing Address</td>
<td>123 Wonderland</td>
</tr>
<tr>
<td>City</td>
<td>Wondercity</td>
</tr>
<tr>
<td>State</td>
<td>AL</td>
</tr>
<tr>
<td>Zip Code</td>
<td>12345</td>
</tr>
<tr>
<td>Credit Card Number</td>
<td>1234567812345678</td>
</tr>
<tr>
<td>Expiry Date</td>
<td>01 2005</td>
</tr>
</tbody>
</table>

Thank you for stopping by!
Example: `<h:outputText> in finish.jsp (old: need to be updated)`

```jsp
<h:outputMessage value="#{bundle.thanksLabel}">
    <f:parameter value="#{sessionScope.firstName}"/>
</h:outputMessage>
```

- value attribute specifies the MessageFormat pattern
- parameter tag specifies the substitution parameters for the message

Thanks, {0}, for using carstore. Your car will ship soon.
Thanks, Sang, for using CarStore! Your car will ship soon.

Documentation for this demo
Backing Bean (Model Object) Management
What are Backing Beans?

- Server-side objects associated with UI components used in the page
- Define UI component properties, each of which is bound to
  - a component's value or
  - a component instance
- Can also define methods that perform functions associated with a component, which include validation, event handling, and navigation processing.
Why Backing Beans?

- Separation of Model from View (MVC)
  - Model handles application logic and data: Backing Beans are Model objects
  - View handles presentation: UI components
How to Specify Backing Beans in JSP page?

A page author uses the JavaServer Faces expression language (JSF EL) to bind a component's value or its instance to a backing bean property

- JSF EL is in the form of "#{...}"

A page author also uses the JSF EL to refer to the backing-bean methods that perform processing for the component
Example: Binding Component Value to Backing Bean in greeting.jsp

```
<h:inputText id="userNo" value="#{UserNumberBean.userNumber}"
validator="#{UserNumberBean.validate}"/>
```

userNo component's value is bound to the UserNumberBean.userNumber backing-bean property
UserNumberBean in faces-config.xml

<managed-bean>
  <description>
    The "backing file" bean that backs up the guessNumber webapp
  </description>
  <managed-bean-name>UserNumberBean</managed-bean-name>
  <managed-bean-class>guessNumber.UserNumberBean</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>minimum</property-name>
    <property-class>int</property-class>
    <value>0</value>
  </managed-property>
  <managed-property>
    <property-name>maximum</property-name>
    <property-class>int</property-class>
    <value>10</value>
  </managed-property>
</managed-bean>
Page Navigation
Define Page Navigation

? Application developer responsibility
  - Navigation rules are defined in the application configuration file

? Navigation rules
  - Determine which page to go to after the user clicks a button or a hyperlink
Navigation Rule 1 for guessNumber Example (V1)

<navigation-rule>
  <description>
    The decision rule used by the NavigationHandler to determine which view must be displayed after the current view, greeting.jsp is processed.
  </description>
  <from-view-id>/greeting.jsp</from-view-id>
  <navigation-case>
    <description>
      Indicates to the NavigationHandler that the response.jsp view must be displayed if the Action referenced by a UICommand component on the greeting.jsp view returns the outcome "success".
    </description>
    <from-outcome>success</from-outcome>
    <to-view-id>/response.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
Navigation Rule 2 for guessNumber Example (V1)

<navigation-rule>
  <description>
  The decision rules used by the NavigationHandler to determine which view must be displayed after the current view, response.jsp is processed.
  </description>
  <from-view-id>/response.jsp</from-view-id>
  <navigation-case>
    <description>
    Indicates to the NavigationHandler that the greeting.jsp view must be displayed if the Action referenced by a UICommand component on the response.jsp view returns the outcome "success".
    </description>
    <from-outcome>success</from-outcome>
    <to-view-id>/greeting.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
Navigation Rule

- `<navigation-rule>`
  - defines how to get from one page (specified in the from-tree-id element) to the other pages of the application
  - can contain any number of `<navigation-case>` elements

- `<navigation-case>`
  - defines the page to open next (defined by to-tree-id) based on a logical outcome (defined by from-outcome)
Where can Outcome come from?

 Outcome can be defined by the **action** attribute of the UICommand component that submits the form.

“action” attribute can be a string or action method (#{<BackingBean>.<Method>})

```xml
<h:commandButton id="submit"
    action="success" label="Submit" />
<h:commandButton
    action="#{carstore.buyCurrentCar}"
    value="#{bundle.buy}" />
```
Live your life with Passion!