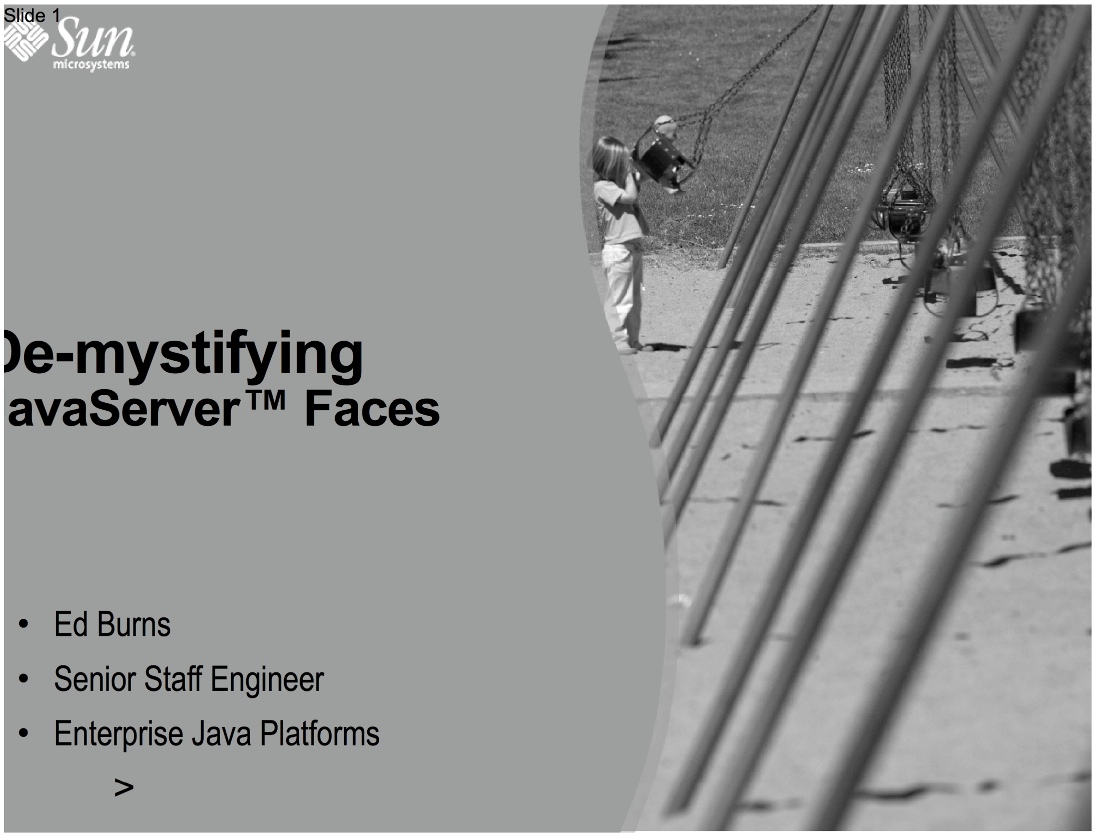
Slide 1

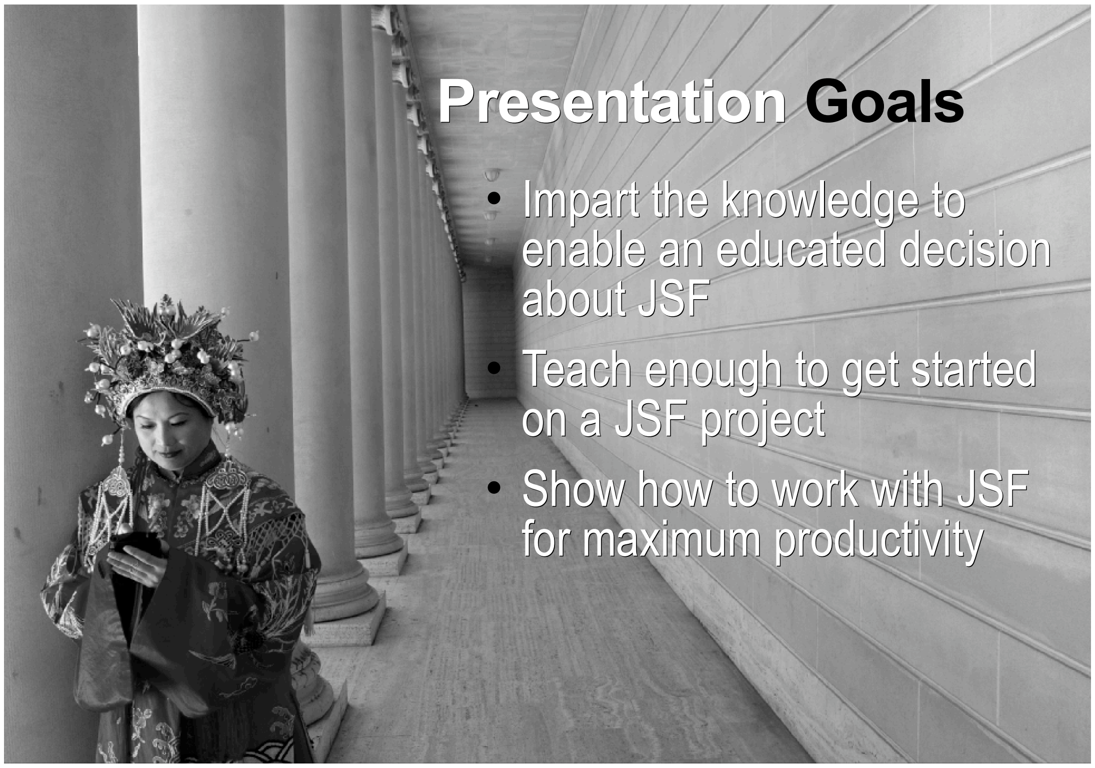


De-mystifying
avaServer™ Faces

- Ed Burns
- Senior Staff Engineer
- Enterprise Java Platforms

>

Slide 2



Presentation Goals

- Impart the knowledge to enable an educated decision about JSF
- Teach enough to get started on a JSF project
- Show how to work with JSF for maximum productivity

Agenda

- Foundations
  - > Web Framework Classifications
  - > Four Pillars of JSF
  - > Design Patterns used in JSF
- Getting Stuff Done
  - > Getting Started
  - > Data Tables
  - > Spring and JSF
- Extending JSF
  - > UI Components
  - > non UI components

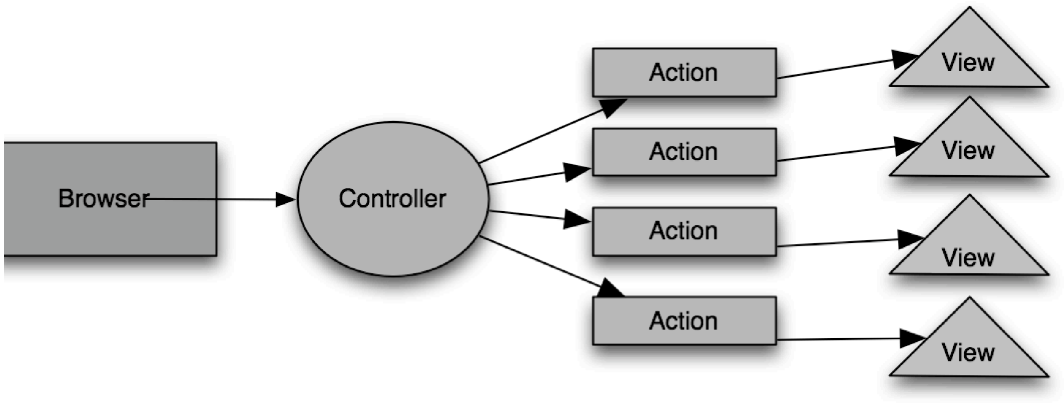
Too Many Frameworks!

Web Framework Classification

- Action Frameworks
  - > Struts, Struts2, Rails
- Hybrid Frameworks
  - > Tapestry, Wicket
- UI Component Frameworks
  - > JSF, Rife, Echo2

Web Framework Classification

Action Frameworks



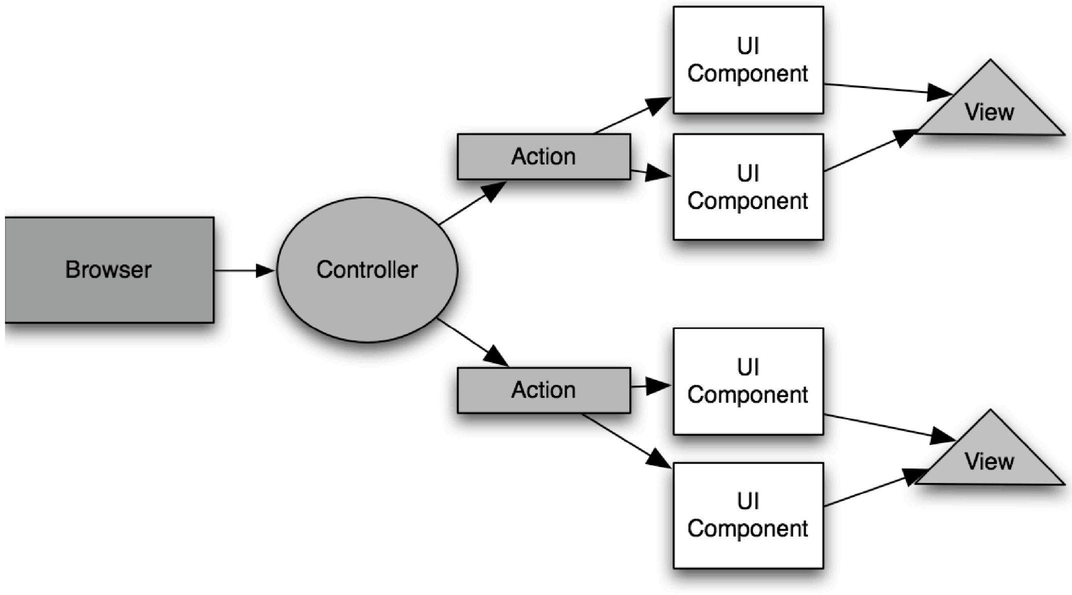
Web Framework Classification

Action Frameworks

- You're all familiar with this kind of framework
- Write an action for each kind of interaction you offer to the user
- Usually have to write a view for each action
- Sometimes have a direct mapping from URL segments to actions and views
  - `http://server/controller/action/view`
- Many implementations require subclassing framework classes for actions and forms

Web Framework Classification

Hybrid Component/Action Frameworks



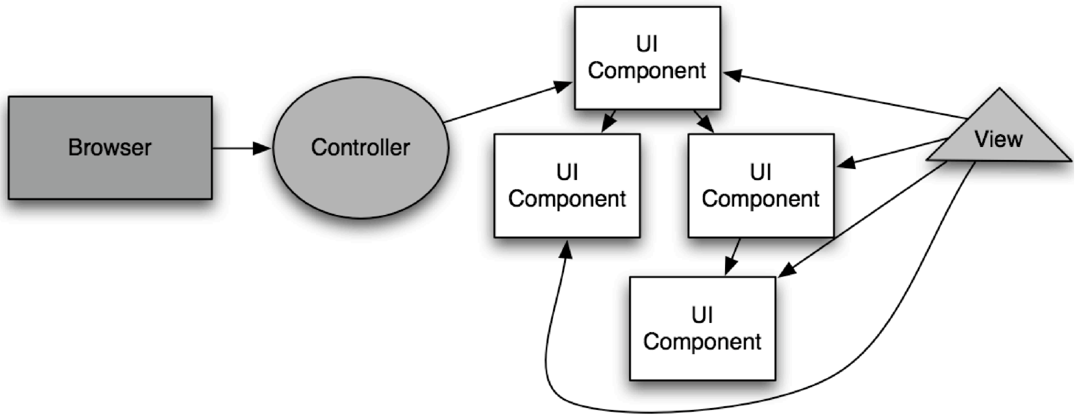
Web Framework Classification

Hybrid Component/Action Frameworks

- Can be said to offer the best of both worlds
  > Tight mapping to HTTP in actions
  > Reuse of components
- Still need to code up 1:1 relationship between actions and views
- Some overhead for component state, but not for controllers

Web Framework Classification

True UI Component Frameworks



Web Framework Classification

True UI Component Frameworks

- Framework interacts directly with Components that comprise the view
- Presence of view tree allows flexible traversal and lifecycle
- Lots of overhead for stateful components
- Model dynamically referenced from components in the view.

Strengths and Weaknesses of JSF

Strengths

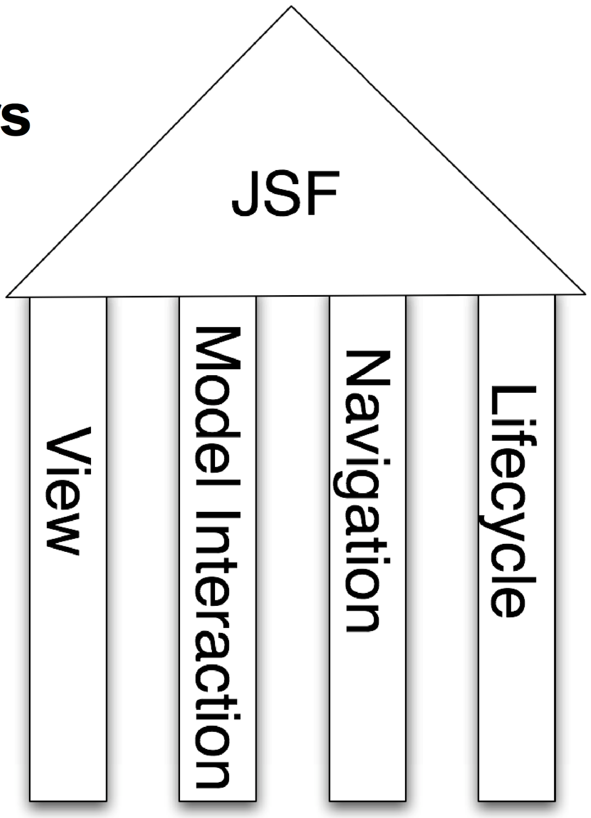
- Powerful
- Flexible
- Abstraction
- Tool support
- “Black Box” components for the web
- I18N,L10N,A11Y
- Industry Standard

Weaknesses

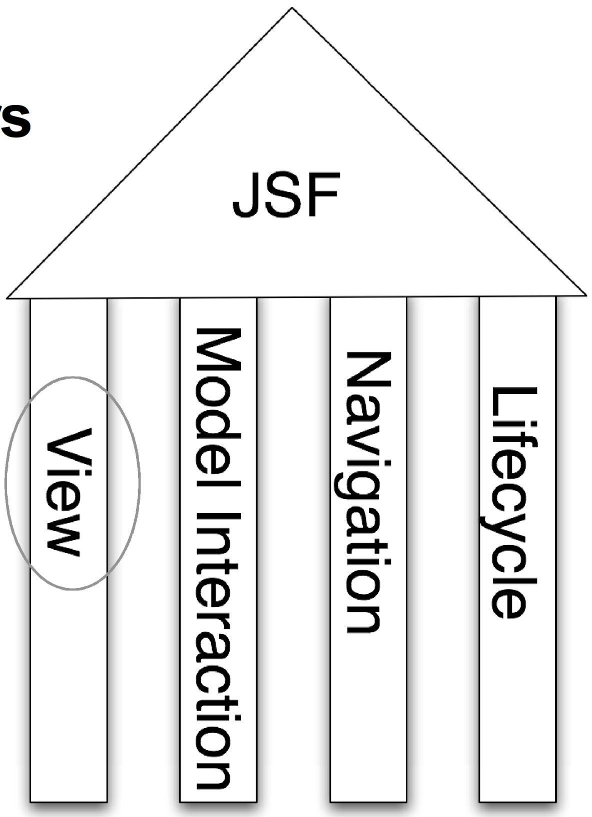
- Complex, overkill in some cases
- Different mind-set from Action based frameworks
- Conceptually divorced from HTTP
- JSP layer had problems prior to JSF 1.2
- All stateful by default- performance problem

Framework Classification Discussion

The Four Pillars of JSF



The Four Pillars of JSF



The Four Pillars of JSF

Views

- How to author them?
  - > JSP: HTML or XML syntax
  - > Facelets: XHTML syntax
- What are they comprised of?
  - > UI Components exposed as markup tags
  - > Nesting of tags defines UI Component Hierarchy
    - >similar to DOM, but definitely not 1:1 with DOM

Slide 17

The Four Pillars of JSF

Views

- What are they comprised of?
  - > Static HTML Markup
  - > Template composition components
- UI Components grouped together into taglibs
- Can use JSTL with JSP based pages
- Can use JSP custom taglibs with JSP based pages
- How does JSF use views?
  - > one thing: build the UI Component tree
- Show Code of JSP and Facelet Views

Slide 18

The Four Pillars of JSF: View

JSP Standard Syntax: not necessarily well formed XML

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


f:view>
html>
  <head><title><h:outputText value=”#{storeFront.title}” />
  </title></head>
  <body><h:form>
  <h1>Items from JSTL</h1>
  <c:forEach var="item" items="#{input}">
    <h:inputText value="#{item}"
         valueChangeListener="#{forEachBean1.valueChange1}"/> <br>
  </c:forEach>
  <p>
  <h1>Items from JSF DataTable</h1>
  <h:dataTable var=”item” value=”#{input}”>
    <h:column>
      <h:inputText value="#{item}"
         valueChangeListener="#{forEachBean1.valueChange1}"/> <br>
    </h:column>
  </h:dataTable>
  </h:form></body>
!-- no closing html tag -->
/f:view>
```

Slide 19

The Four Pillars of JSF: View

JSP XML Syntax: well formed XML

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xml:lang="en" lang="en">
<jsp:output doctype-root-element="html"
            doctype-public="-//W3C//DTD XHTML 1.0 Strict//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd"/>
<jsp:directive.page contentType="application/xhtml+xml;
charset=UTF-8"/>
<f:view>

<html><head><title>Hello</title></head>
<body bgcolor="white"><h:form>
<h2>Hi. My name is Duke. I'm thinking of a number from
<h:outputText lang="en_US" value="#{UserNumberBean.minimum}"/>
to <h:outputText value="#{UserNumberBean.maximum}"/>. Can you
guess it?</h2>
<p><h:inputText id="userNo" label="User Number"
              value="#{UserNumberBean.userNumber}"
              validator="#{UserNumberBean.validate}"/>
<h:commandButton id="submit" action="success" value="Submit"/></p>
</body>
</html>

</f:view>
```

Slide 20

The Four Pillars of JSF: View

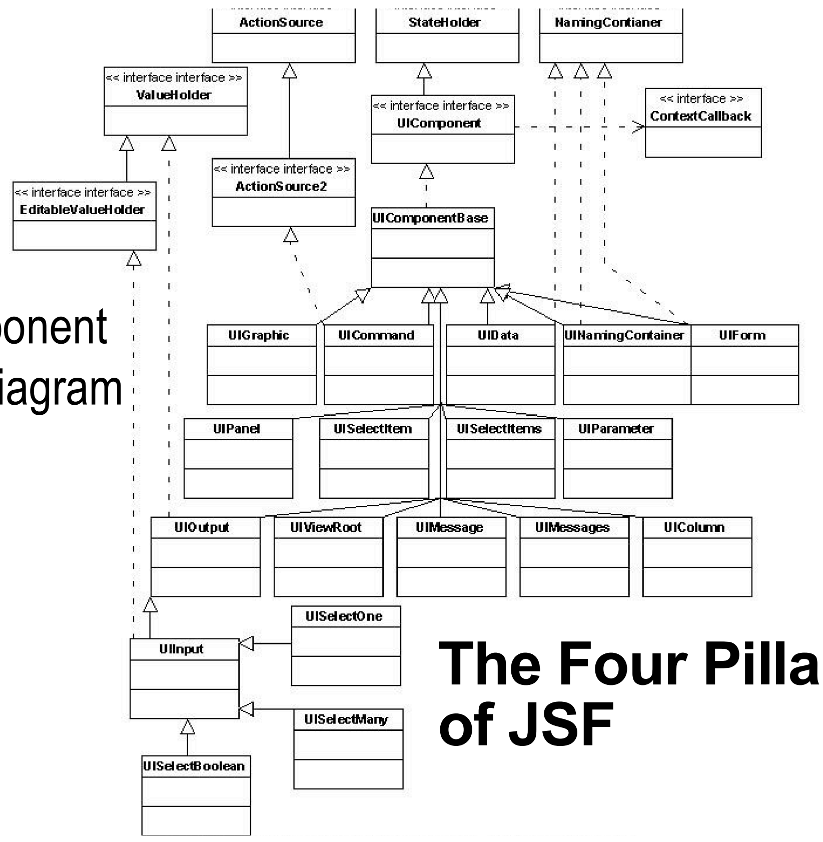
Facelets: well formed XML

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xml:lang="en" lang="en">
<f:view>

<html><head><ui:insert name="head"/></head>
<body bgcolor="white"><h:form>
<h2>Hi. My name is Duke. I'm thinking of a number from
<h:outputText lang="en_US" value="#{UserNumberBean.minimum}"/>
to <h:outputText value="#{UserNumberBean.maximum}"/>. Can you
guess it?</h2>
<p><h:inputText id="userNo" label="User Number"
              value="#{UserNumberBean.userNumber}"
              validator="#{UserNumberBean.validate}"/>
<h:commandButton id="submit" action="success" value="Submit"/></p>
</body>
</html>

</f:view>
```

iews -
lComponent
lass Diagram



The Four Pillars of JSF

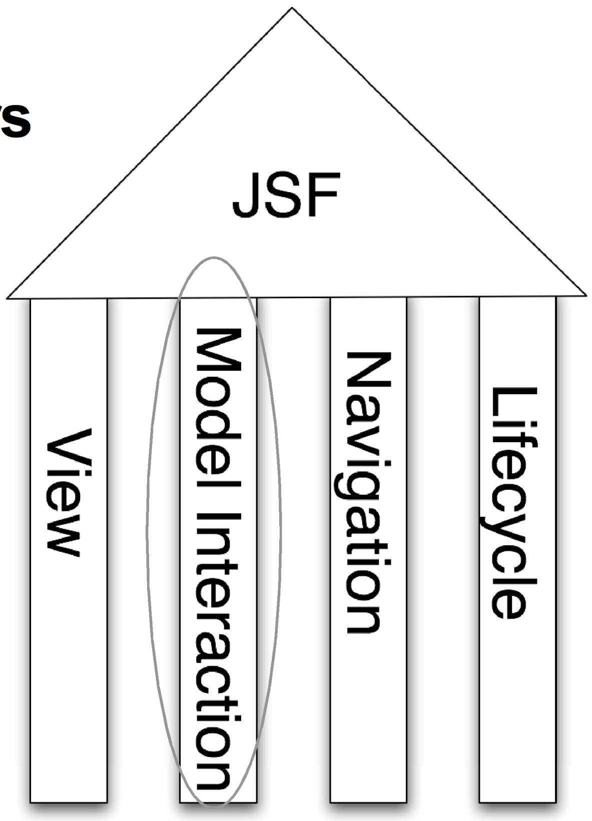
The Four Pillars of JSF

Views

- `UIComponent` defines semantics of a UI component
- `Renderer` defines appearance
- Each tag in the page is really a declaration of a pairing of a `UIComponent` with a `Renderer`
- The choice to use the delegated rendering approach is up to the component author

Slide 23

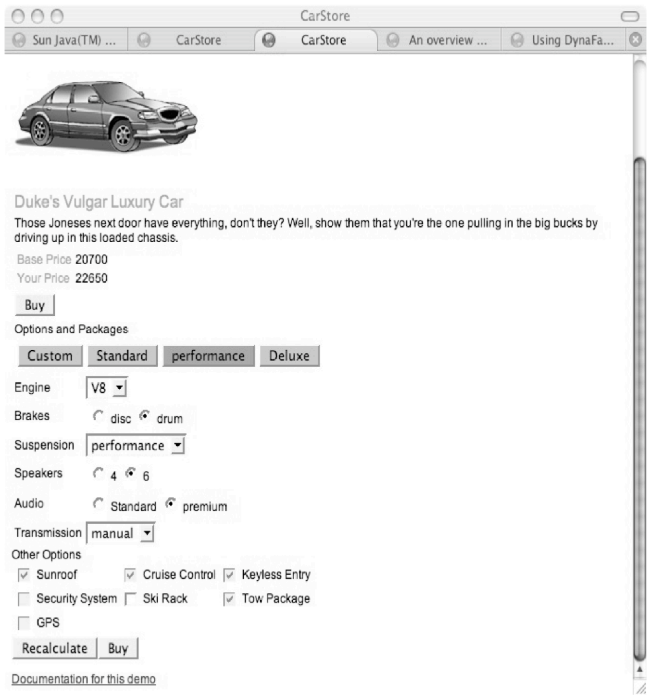
The Four Pillars of JSF



Slide 24

The Four Pillars of JSF

Model Interaction



Duke's Vulgar Luxury Car

Those Joneses next door have everything, don't they? Well, show them that you're the one pulling in the big bucks by driving up in this loaded chassis.

Base Price 20700
Your Price 22650

Buy

Options and Packages

Custom | Standard | performance | Deluxe

Engine: V8

Brakes: disc [✓ drum]

Suspension: performance

Speakers: 4 [✓ 6]

Audio: Standard [✓ premium]

Transmission: manual

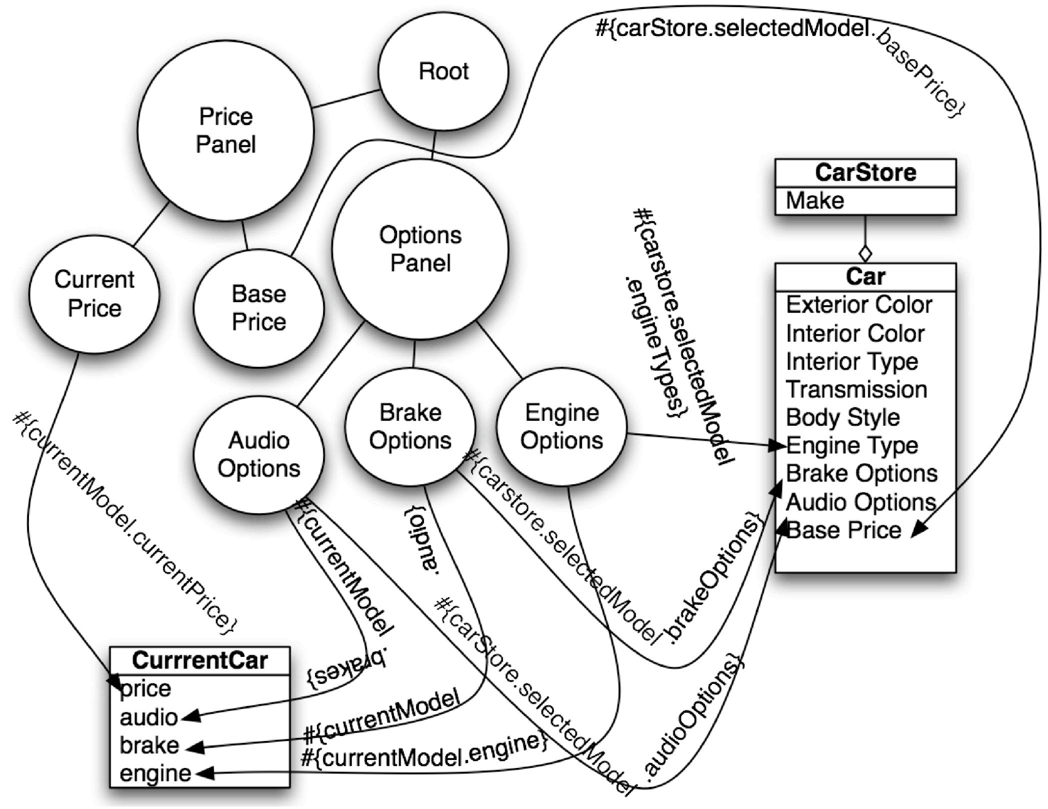
Other Options

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

Recalculate | Buy

Documentation for this demo

Slide 25



Slide 26

The Four Pillars of JSF

Model Interaction - EL

- Use EL to “point to” properties of model objects from UI Components
- UI Component will get its “value” from the associated model property.
- Type safe conversion happens automatically
- Server side validation can happen if desired
- Model objects should only store converted and validated data
- You are responsible for persisting model objects

The Four Pillars of JSF

Model Interaction - EL

- Model objects are POJOs, usually managed-beans
- Model objects can be “backing beans”: one per-page, but are not required to be used in this way
- EL Expressions operate bi-directionally
  > “read” or “get” when rendering
  > “write” or “set” when posting back
- EL allows easy integration to technologies outside of JSF: Spring, Seam, Hibernate, JNDI, etc
- UIData provides access to iterating over data

Spring Integration Example

Model Interaction - EL

- Model objects are POJOs, usually managed-beans
- Model objects can be “backing beans”: one per-page, but are not required to be used in this way
- EL Expressions operate bi-directionally
  > “read” or “get” when rendering
  > “write” or “set” when posting back
- EL allows easy integration to technologies outside of JSF: Spring, Seam, Hibernate, JNDI, etc
- UIData provides access to iterating over data

The Four Pillars of JSF

Model Interaction - EL

- EL implicit objects give access to web scopes and more: `cookie, facesContext, header, headerValues, param, paramValues, request, requestScope, view, application, applicationScope, initParam, session, sessionScope`
- Examples: `#{requestScope.user.name}, #{prop}`
- Widening scope lookup happens automatically
- Can introduce new implicit objects easily
- Takes full advantage of JavaBeans naming conventions for getters, setters, arrays, lists.

The Four Pillars of JSF

Model Interaction - EL

- Can use EL in View markup
- Can use EL programmatically
- EL Can also point to component instances
  - > value style vs. component style
  - > Show code to illustrate use of EL in both styles
- EL can also point to methods

Slide 31

The Four Pillars of JSF

Model Interaction – Managed Beans

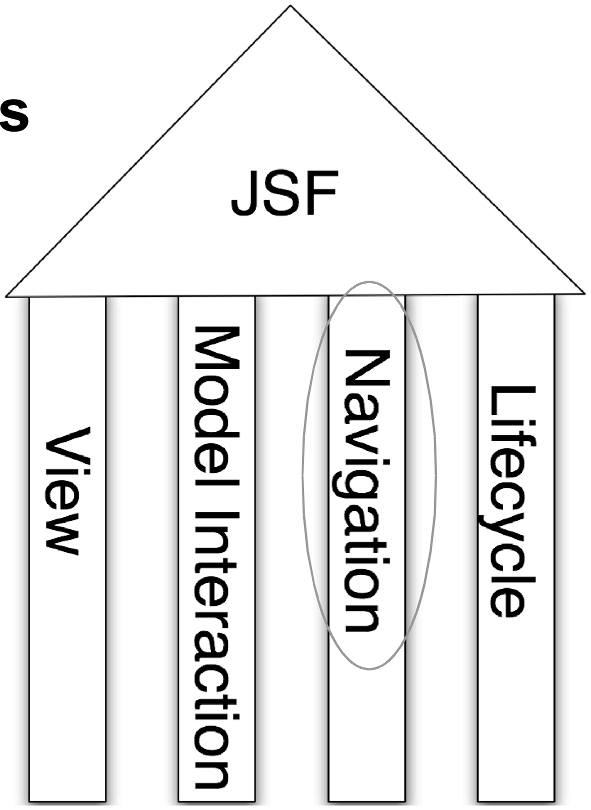
- The built in Inversion of Control framework for JSF
- Can replace or supplement JSF managed beans with Spring Framework
- Managed beans accessed via EL are created lazily by the framework and placed into the proper "scope": none, request, session, application
- Can use Java EE 5 Resource Injection into Managed Beans
- Show code for managed-beans

Slide 32

View and Model Interaction Discussion

Slide 33

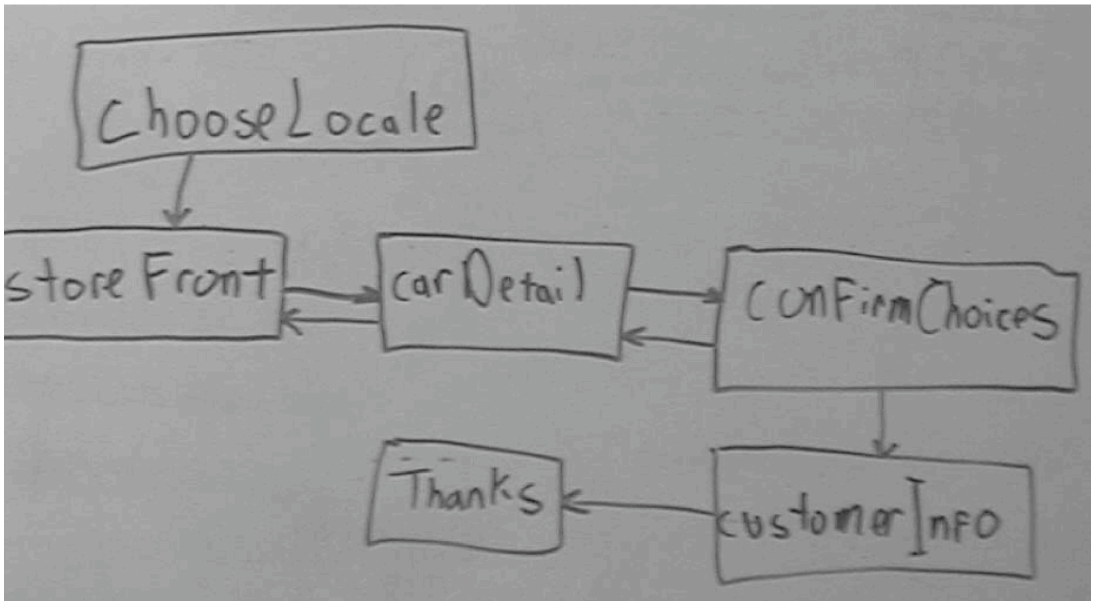
The Four Pillars of JSF



Slide 34

The Four Pillars of JSF

Navigation



The Four Pillars of JSF

Navigation

- Pre-Ajax (and even post-Ajax) web apps are a modeled very well as a directed graph of web pages.
- The valid traversal paths for the graph can be modeled as page navigation
- JSF has a simple yet flexible navigation model that enables centralizing the navigation rules for the entire application in one place, yet allows easily changing these rules so the valid traversal paths are changed.

The Four Pillars of JSF

Navigation

- Every `ActionSource` component can cause a page transition to occur based on the “outcome”
  - > A literal string hard coded in the page
  - > Using the EL to point to a method that takes no arguments and returns an Object that has a reasonable `toString()`
- No limit on the number of `ActionSource` components in a page.
- Each one can return a different outcome
- A `null` outcome means, “stay on the current view”

Slide 37

The Four Pillars of JSF

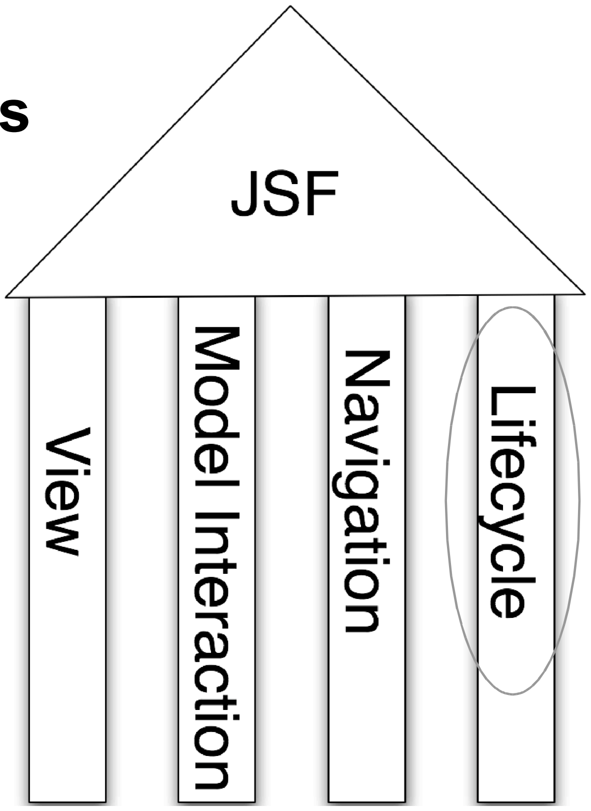
Navigation

- The outcome is fed into a rule base and combined with the current viewId to derive where to go next, and how to get there. Example:

```
<navigation-rule>
     <from-view-id>/chooseLocale.jsp</from-view-id>
     <navigation-case>
         <description>
             Any action on chooseLocale should cause navigation to
             storeFront.jsp
         </description>
         <from-outcome>storeFront</from-outcome>
         <to-view-id>/storeFront.jsp</to-view-id>
         <!-- <redirect /> -->
     </navigation-case>
 </navigation-rule>
```

Slide 38

The Four Pillars of JSF



The Four Pillars of JSF

Lifecycle

- The Lifecycle dictates how an incoming request is handled and how a response is generated.
- Primarily concerned with
  > Finding the View on which to operate
  > Allowing components to get their values
  > Ensuring the values are converted and validated
  > Ensuring any event listeners are called
  > Updating the model
  > Selecting and rendering the new view

The Four Pillars of JSF

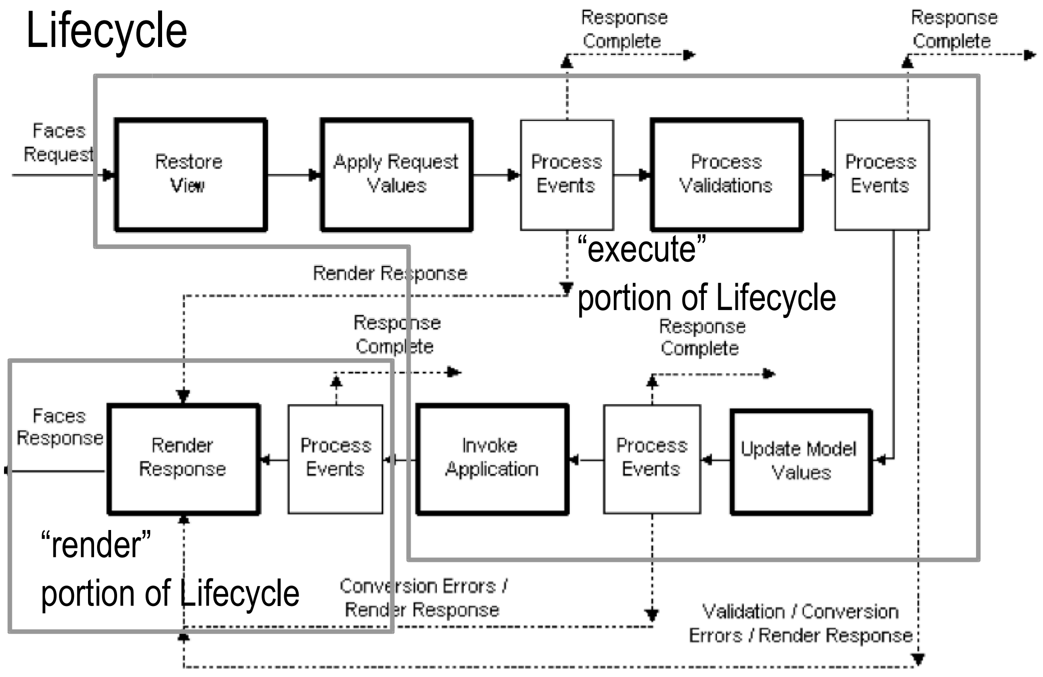
Lifecycle

- You don't need to invoke the Lifecycle yourself
- Two methods
  > execute
    Handles the postback processing for the request
  > render
    Renders the view
- Each method uses a number of “phases” to accomplish its task.

Slide 41

The Four Pillars of JSF

Lifecycle



Slide 42

The Four Pillars of JSF

Lifecycle

- You can install `PhaseListener`s into the Lifecycle to do whatever you want before or after each or every phase.
- A `PhaseListener` has full access to the entire JSF framework, including manipulation of the view.
- You can use decoration to provide your own Lifecycle if you want.

Slide 43

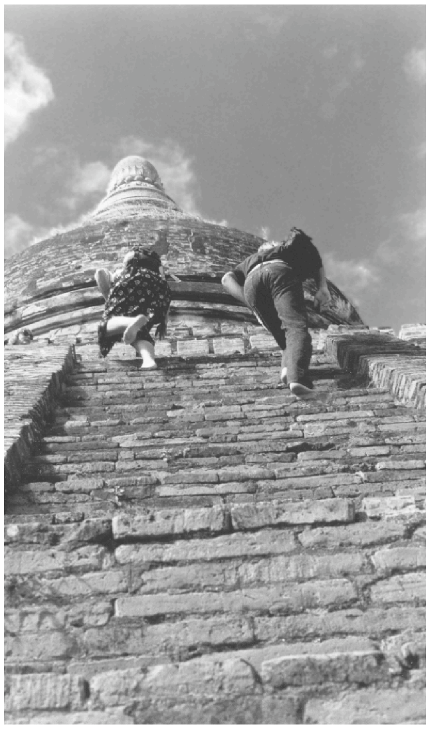
Design Patterns at work in JSF

- Decorator
  - > Used throughout to allow customization
- Singleton
  - > FacesServlet, Lifecycle, ViewHandler, RenderKit,
- Strategy
  - > Flexible Rendering model
- Template Method
  - > PhaseListeners
- Observer
  - > java.util.EventListener

Slide 44

Navigation, Lifecycle and Design Patterns Discussion

Slide 45



Gettings Things Done

- Getting Started
  - How to “get in” to JSF
  - Config files you need
  - Really Simple App
- Getting Real (thanks DHH)
  - Listeners
    - ActionListener
    - ValueChangeListener
  - Tables
  - Spring Integration

Slide 46

How to “get in” to JSF

- Map the FacesServlet to whatever
- Browser makes a GET that triggers the mapping
- FacesServlet fires up the Lifecycle and hands control to it
- Lifecycle hits Render Response phase, renders view
- Browser makes one or more POST requests which, by virture of having been rendered by JSF, will automatically hit the FacesServlet.

Config Files you need

- WEB-INF/web.xml
  - > Required for declaring FacesServlet
  - > Optionally map FacesServlet
  - > Optionally pass init params to JSF runtime
    - > State saving method
    - > Alternate Lifecycle
    - > XML validation of faces-config files
    - > Implementation specific options
  - > Anything else for which you normally use web.xml
    - > Resource declarations
    - > Security options

Config Files you need

- WEB-INF/faces-config.xml
  - > Don't need it for really simple apps without:

    managed-beans, non-jarred {components, converters, validators, renderers, etc}, navigation
  - > Declares

    any non-jarred JSF artifacts, Example: j1 app

    navigation-rules
- Let's look at the DTD (schemas are not human readable!)
  - > For regular apps that don't need to extend, all you really need to know is managed beans
  - > We'll cover the rest later.

Simple Toy App

helloDuke - files

web/WEB-INF/web.xml

web/WEB-INF/faces-config.xml

web/index.html

web/greeting.jsp

web/response.jsp

web/wave.med.gif

src/helloDuke/UserNameBean.java

Getting Real

Listeners - ActionListener

- General use:
  > MethodExpression in view

```
<bp:scroller for="datagrid"
  actionListener="#{orderEnt.scrollDataGrid}">
```

  > Java code on managed bean

```
public void scrollDataGrid(ActionEvent e) {
  if (this.data != null) {
    int rows = this.data.getRows();
    if (rows < 1) return;
    if (currentRow < 0)
      this.data.setFirst(0);
    else if ...}
}
```

Getting Real

Listeners - ActionListener

- Also have `f:actionListener` tag
- Names an actual class that must implement `ActionListener`
- Example: J1 order entry scroller

Getting Real

Listeners - ValueChangeListener

- General use:
  > MethodExpression in view

```
<a:ajax name="dojo.fisheye"
  value="#{bean.selectedIndex}"
valueChangeListener="#{bean.valueChanged}"/>
```

  > Java code on managed bean

```
public void valueChanged(ValueChangeEvent e) {
  ... do something with the event
}
```

Getting Real

Listeners - ActionListener

- Also have `f:valueChangeListener` tag
- Names an actual class that must implement `ValueChangeListener`
- Example: Dojo Fisheye widget

Getting Real

Displaying Tabular Data

- Use `<h:dataTable value=”#{foo.dataModel}” />`
  - > `#{foo.dataModel}` points to an instance of `javax.faces.model.DataModel`
    - >Which has subclasses:
      ArrayDataModel, ListDataModel,
      ResultDataModel (JSTL Result),
      ResultSetDataModel (java.sql.ResultSet),
      ScalarDataModel (wraps Java Object)
  - > `#{foo.dataModel}` points to an array, List, Result (JSTL), ResultSet, Object. In this case, `DataModel` instance is constructed automatically

Getting Real

Displaying Tabular Data

- Example: Order Line system from J1 demo

Getting Real

Spring Integration Options

- Replace JSF's managed-bean facility with Spring via the SpringVariableResolver.
- Use JSF and Spring MVC together in the same application
- Replace JSF's navigation facility with Spring Web Flow via a custom Navigation Handler (prototype concept!)

Listeners, Tables, Spring Integration

xtending JSF

- UI Components
  - > Without delegated rendering
  - > With delegated rendering
- Non-UI Components
  ActionListener,ELResolver,
  {Variable,Property}Resolver,
  FacesContextFactory,
  RenderKit,
  NavigationHandler,
  ViewHandler
- Packaging components

Custom UI Components

Without Delegated Rendering

- Extend UIComponent or UIComponentBase
- Important methods
  > encode*(), decode(), saveState(), restoreState()
- Declare component in faces-config.xml

```
<component>
  <component-type>Chart</component-type>
  <component-class>com.foo.ChartComponent
    </component-class>
</component>
```

Custom UI Components

With Delegated Rendering

- Extend UIComponent or UIComponentBase
- Do not implement encode*() or decode*()
- Important methods
  > saveState(), restoreState()
- Component declaration same as before
- Extend Renderer

Custom UI Components

With Delegated Rendering

- Extend Renderer
- Show MapRenderer
- Declare in faces-config.xml

```
<render-kit>
  <renderer>
    <component-family>ProgressBar</component-family>
    <renderer-type>ProgressBar</renderer-type>
    <renderer-class>com.foo.ChartRenderer
</renderer-class>
</renderer>
</render-kit>
```

Custom Non-UI Components

- ActionListener,ELResolver,
  {Variable,Property}Resolver,
  FacesContextFactory,
  RenderKit,
  NavigationHandler,
  ViewHandler
- Back to the DTD

Packaging components in a JAR

- Key concept:
  JSF Runtime must look for
  `META-INF/faces-config.xml` file in every JAR file in
  `WEB-INF/lib`
  at web app deployment time
- Wide range of COTS components (both UI and non-UI)
  > see JSFCentral.com

Summary

- JSF is a component based framework
- JSF is easy to understand if you know the fundamentals
  > View
  > Model Interaction
  > Navigation
  > Lifecycle
- https://javaserverfaces.dev.java.net/

De-Mystifying JSF

Ed Burns
ed.burns@sun.com