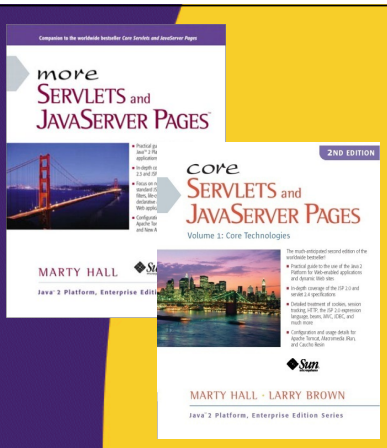
JSF: Handling Events

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/

This somewhat old tutorial covers JSF 1, and is left online for those maintaining existing projects. All new projects should use JSF 2, which is both simpler and more powerful. See http://www.coreservlets.com/JSF-Tutorial/jsf2/.

© 2012 Marty Hall



Topics in This Section

- **Comparing action controllers to event listeners**
- **Action listeners**
- **Value change listeners**
- **Using JavaScript to submit form**
  - Browser incompatibilities
- **Combining action listeners and action controllers in the same GUI element**

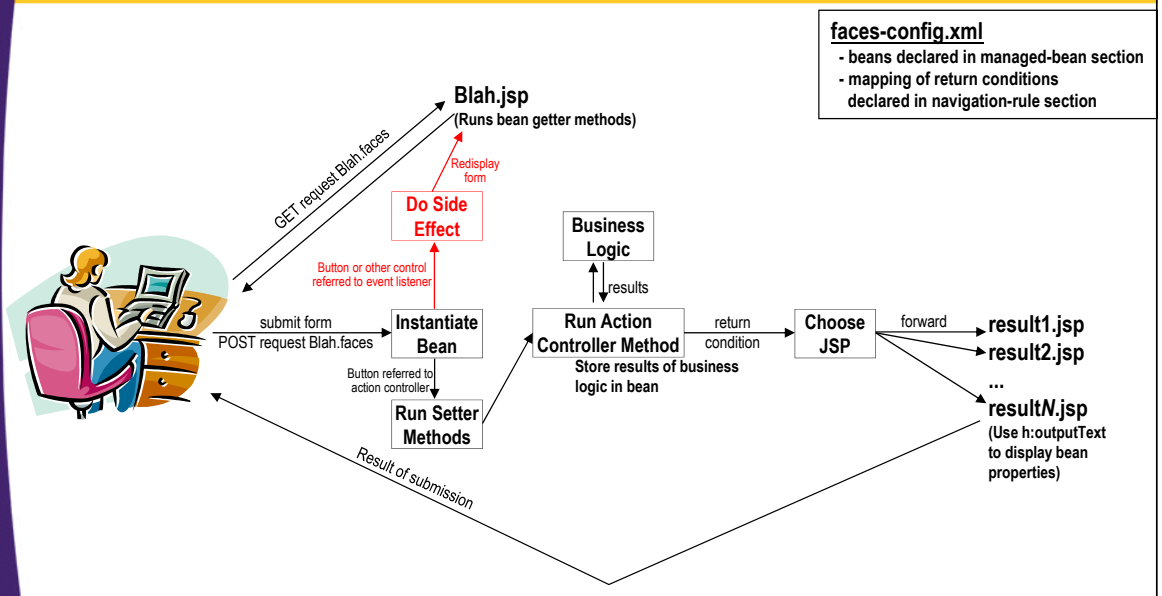
Motivation

- **There are two varieties of user interface events**
  - Events that start back-end processing
  - Events that affect only the format of the user interface
- **JSF categorizes code that handles these as *action controllers* and *event listeners***
  - Action controllers handle main form submission
    - Fire after bean has been populated (see previous section)
    - Fire after validation logic (see upcoming section)
    - Return strings that directly affect page navigation
  - Event listeners handle UI events
    - Often fire before bean has been populated
    - Often bypass validation logic
    - Never directly affect page navigation

Event Handlers vs. Ajax

- **Event handlers are one of the most convenient features of JSF**
  - Especially compared to Struts or regular MVC
- **However, Ajax is often even better**
  - Many situations where event handlers apply result in only a small number of elements changing
  - In that case, Ajax yields a more interactive user experience
    - Although it fails in older browsers and if JavaScript is disabled
  - See later lecture on Ajax4jsf

JSF Flow of Control (Updated)



Types of Event Listeners

- **ActionListener**
  - Fired by submit buttons, image maps, and hypertext links with attached JavaScript
    - <h:commandButton value="..." .../>
    - <h:commandButton image="..." .../>
    - <h:commandLink .../>
  - Automatically submit the form
- **ValueChangeListener**
  - Fired by combo boxes, checkboxes, radio buttons, textfields, and others
    - <h:selectOneMenu .../>
    - <h:selectBooleanCheckbox.../>
    - <h:selectOneRadio .../>
    - <h:inputText .../>
  - Do *not* automatically submit the form

Action Listeners

Using ActionListener in JSF

- **Some buttons submit the form and start backend processing**
  - Use <h:commandButton action="..." ...>
- **Other buttons affect only the UI**
  - Use <h:commandButton actionListener="..." .../>
  - You usually want this process to occur before beans are populated and especially before validation occurs
    - Since forms are often incomplete when the UI is adjusted
  - Use "immediate" to designate that listener fires before beans are populated or validation is performed

```
<h:commandButton actionListener="..."
                 immediate="true" .../>
```


Implementing ActionListener in the Java Code

- **Listener is usually in the form bean class**
  - But can be in separate class if you use FacesContext to get the request or session object and look up the form bean explicitly
- **Takes an ActionEvent as an argument**
  - Void return type (*not* String as in action controllers)
  - ActionEvent is in javax.faces.event
  - ActionEvent has a getComponent method that lets you obtain the UIComponent reference
    - From the UIComponent, you can get the component ID, renderer, and other low-level information
  - Sample code

```
public void someMethod(ActionEvent event) {
  doSomeSideEffects();
}
```

Example: Letting the User Disable Certain UI Controls

- **Every UI control has a "disabled" property that is false by default**
- **Use button with attached ActionListener to disable or reenable certain controls**
- **Specific example**
  - Collect name and job title to display a sample resume
  - Create comboboxes that let user select foreground and background colors

```
<h:selectOneMenu value="..." disabled="..." ...>
  <f:selectItems value="..."/>
</h:selectOneMenu>
```

    - Value for f:selectItems should be a SelectItem array
  - Have button that disables or reenables the comboboxes

Steps in Using JSF

1) Create a bean

A) Properties for form data
B) Action controller and event listener methods
C) Placeholders for results data

2) Create an input form

A) Input fields refer to bean properties
B) Button specifies action controller method that will return condition
C) Button or other GUI control specifies event listener method

3) Edit faces-config.xml

A) Declare the bean
B) Specify navigation rules

4) Create results pages

- Output form data and results data with h:outputText

5) Prevent direct access to JSP pages

- Use a filter that redirects blah.jsp to blah.faces

Step 1: Create Bean

- (A) Properties for form data

```
package coreservlets;
import javax.faces.model.*;
import javax.faces.event.*;

public class ResumeBean implements Serializable {
  ...
  private String fgColor = "BLACK";
  private String bgColor = "WHITE";
  private SelectItem[] availableColorNames =
    { new SelectItem("BLACK"),
      new SelectItem("WHITE"),
      new SelectItem("SILVER"),
      new SelectItem("RED"),
      new SelectItem("GREEN"),
      new SelectItem("BLUE") };

  public String getFgColor() { return(fgColor); }

  public void setFgColor(String fgColor) {
    this.fgColor = fgColor;
  } ...

  public SelectItem[] getAvailableColors() {...}
```


Step 1: Create Bean

- (A) Properties for form data

```
  ...
  private String name = "";
  private String jobTitle = "";
  ...
  public String getName() { return(name); }

  public void setName(String name) {
    this.name = name;
  }

  public String getJobTitle() { return(jobTitle); }

  public void setJobTitle(String jobTitle) {
    this.jobTitle = jobTitle;
  }
```

Step 1: Create Bean

- (B) Event listener and action controller methods

```
public boolean isColorSupported() {
  return(isColorSupported);
}

public void toggleColorSupport(ActionEvent event) {
  isColorSupported = !isColorSupported;
}

public String showPreview() {
  if (isColorSupported && fgColor.equals(bgColor)) {
    return("same-color");
  } else {
    return("success");
  }
}
```

Action listener

Action controller

Step 2: Create Input Form

- **New features**
  - Use h:selectOneMenu to create combobox
  - Use f:selectItems to populate list
  - Use h:commandButton
    - With actionListener and immediate
      - Designates event listener code to fire when button clicked
  - Change the label (value) of the button depending on what the button will do
    - In this case, button is a toggle
- **Example code**

```
<h:commandButton
    value="#{someBean.buttonLabel}"
    actionListener="#{someBean.doSideEffect}"
    immediate="true"/>
```

Step 2: Create Input Form

- **(A) Input fields**

```
<h:form>
  ...
  Foreground color:
  <h:selectOneMenu value="#{resumeBean.fgColor}"
                   disabled="#{!resumeBean.colorSupported}">
    <f:selectItems value="#{resumeBean.availableColors}"/>
  </h:selectOneMenu>
  <BR>
  Background color:
  <h:selectOneMenu value="#{resumeBean.bgColor}"
                   disabled="#{!resumeBean.colorSupported}">
    <f:selectItems value="#{resumeBean.availableColors}"/>
  </h:selectOneMenu><BR>
  ...
  Name:
  <h:inputText value="#{resumeBean.name}"/><BR>
  Job Title:
  <h:inputText value="#{resumeBean.jobTitle}"/><P>
</h:form>
```


Step 2: Create Input Form

- **(B) Action controller**
  - Even when you have event listeners, you almost always have action controllers
    - As before, these invoke business logic and participate in navigation flow (via navigation-rule entries in config file)
    - Setter methods and validation fire before action controller

```
<h:form>
  ...
  <h:commandButton value="Show Preview"
                   action="#{resumeBean.showPreview}"/>
</h:form>
```

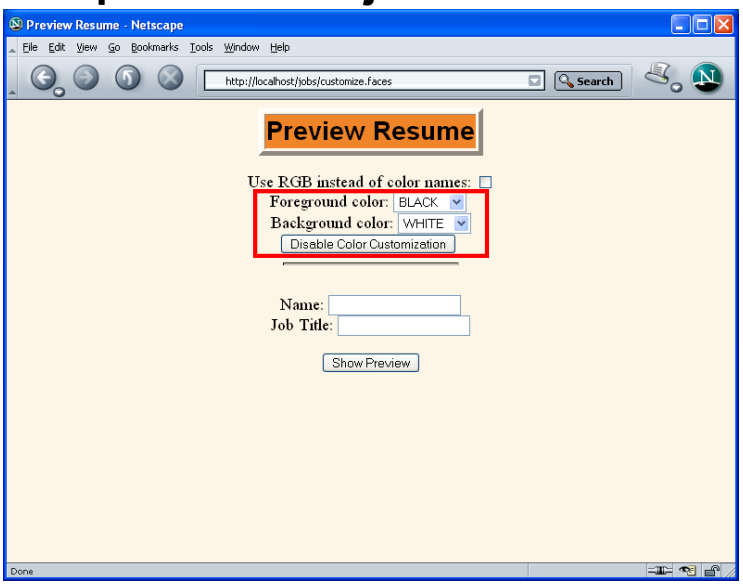
Step 2: Create Input Form

- **(C) Event listener**
  - Event listeners perform side effects, and then form is redisplayed
    - They do not usually invoke normal business logic, setter methods are not usually fired, and they never participate in navigation flow (navigation-rule entries do not apply)
    - You usually use "immediate" to say that setter methods do not fire

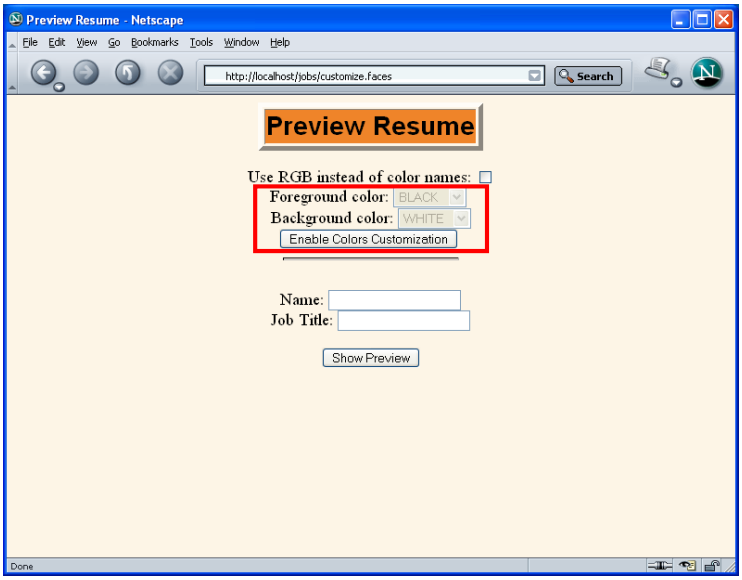
```
<h:form>
  ...
  <h:commandButton
     value="#{resumeBean.colorSupportLabel}"
     actionListener="#{resumeBean.toggleColorSupport}"
     immediate="true"/>
</h:form>
```


Steps 2-4: Initial Result

- **File is *tomcat_dir*/webapps/jobs/customize.jsp**
- **URL is http://localhost/jobs/customize.faces**



Step 2: Result of Clicking Button (Independent of Action Controller)



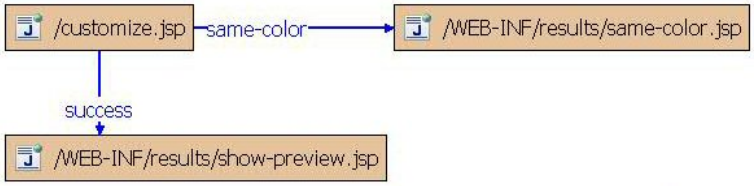
Step 3: Edit faces-config.xml

- **(A) Declaring bean**
  - Note session scope so changes persist

```
...
<faces-config>
  <managed-bean>
    <managed-bean-name>resumeBean</managed-bean-name>
    <managed-bean-class>
      coreservlets.ResumeBean
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
...
</faces-config>
```

Step 3: Edit faces-config.xml

- **(B) Specifying navigation rules**




```
...
<faces-config>
  ...
  <navigation-rule>
    <from-view-id>/customize.jsp</from-view-id>
    <navigation-case>
      <from-outcome>same-color</from-outcome>
      <to-view-id>/WEB-INF/results/same-color.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>/WEB-INF/results/show-preview.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```


Step 4: Create Results Pages

- **WEB-INF/results/same-color.jsp**
  - Results pages apply to action controller, *not* event listener

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE …>
<HTML>
...
You chose
"<h:outputText value="#{resumeBean.fgColor}"/>"
as both the foreground and background color.
<P>
You don't deserve to get a job, but I suppose
we will let you <A HREF="customize.faces">try again</A>.
...
</HTML>
</f:view>
```

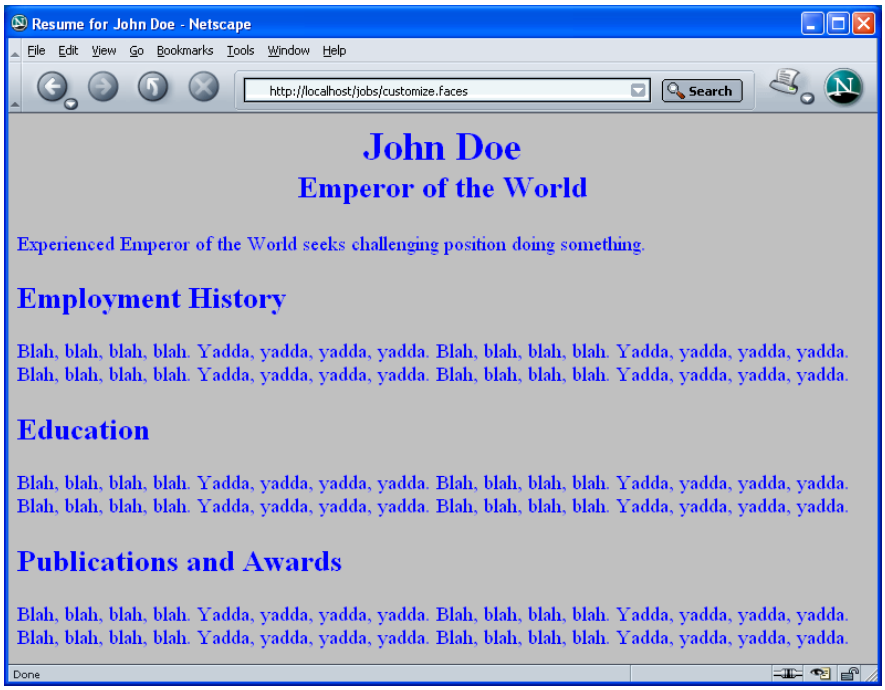
Step 4: Create Results Pages

- **WEB-INF/results/show-preview.jsp**

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE …>
<HTML>
…
<BODY TEXT="<h:outputText value="#{resumeBean.fgColor}"/>"
      BGCOLOR="<h:outputText value="#{resumeBean.bgColor}"/>">
<H1 ALIGN="CENTER">
<h:outputText value="#{resumeBean.name}"/><BR>
<SMALL><h:outputText value="#{resumeBean.jobTitle}"/>
</SMALL></H1>
Experienced <h:outputText value="#{resumeBean.jobTitle}"/>
seeks challenging position doing something.
<H2>Employment History</H2>
Blah, blah, blah, blah. Yadda, yadda, yadda, yadda.
…
</HTML>
</f:view>
```


Step 4: Example Result for Same Colors

Step 4: Example Result for Different Colors



Step 5: Prevent Direct Access to JSP Pages

- **Use filter that captures url-pattern *.jsp**
  - No changes from previous examples

Value Change Listeners

Using ValueChangeListener in JSP

- **ActionListener was attached to button**
  - Form was automatically submitted when clicked
- **ValueChangeListener is attached to combobox, listbox, radio button, checkbox, textfield, etc.**
  - Form not automatically submitted
  - Need to add JavaScript to submit the form
    `onclick="submit()"` or `onchange="submit()"`
  - Event incompatibility between Firefox and IE
    - Firefox, Netscape, and Opera fire the onchange events when the combobox selection changes, radio button is selected, or checkbox is checked/unchecked
    - Internet Explorer fires event after selection changes, but only when another GUI control receives the input focus
      - Older IE versions behave differently! Test on multiple browsers!

Implementing ValueChangeListener in Java Code

- **Listener is usually in the form bean class**
  - As discussed previously
- **Takes a ValueChangeEvent as an argument**
  - Useful ValueChangeEvent methods
    - getComponent (as mentioned for ActionEvent)
    - getOldValue (previous value of GUI element)
    - getNewValue (current value of GUI element)
      - Needed since bean has probably not been populated
      - Value for checkbox is of type Boolean
      - Value for radio button or textfield corresponds to request parameter
  - Sample code

```
public void someMethod(ValueChangeEvent event) {
  boolean flag =
    ((Boolean)event.getNewValue()).booleanValue();
  takeActionBasedOn(flag);
}
```


Example: Changing the Colors Listed for FG and BG Colors

- **h:selectOneMenu uses f:selectItems to get list of combobox entries**
- **Use checkbox to invoke ValueChangeListener**
- **Have listener toggle the definition of the lists**
  - Color names
  - RGB values

Steps in Using JSF

1) Create a bean
- A) Properties for form data
- B) Action controller and event listener methods
- C) Placeholders for results data

2) Create an input form
- A) Input fields refer to bean properties
- B) Button specifies action controller method that will return condition
- C) Button or other GUI control specifies event listener method

3) Edit faces-config.xml
- A) Declare the bean
- B) Specify navigation rules

4) Create results pages
- Output form data and results data with h:outputText

5) Prevent direct access to JSP pages
- Use a filter that redirects blah.jsp to blah.faces

Step 1: Create Bean (ResumeBean)

- **(A) Code for color choices**

```
private SelectItem[] availableColorNames =
  { new SelectItem("BLACK"),
    new SelectItem("WHITE"),
    new SelectItem("SILVER"),
    new SelectItem("RED"),
    new SelectItem("GREEN"),
    new SelectItem("BLUE") };
private SelectItem[] availableColorValues =
    { new SelectItem("#000000"),
      new SelectItem("#FFFFFF"),
      new SelectItem("#C0C0C0"),
      new SelectItem("#FF0000"),
      new SelectItem("#00FF00"),
      new SelectItem("#0000FF") };
private boolean isUsingColorNames = true;

public SelectItem[] getAvailableColors() {
  if (isUsingColorNames) {
    return(availableColorNames);
  } else {
    return(availableColorValues);
  }
}
```

Step 1: Create Bean

- **(B) Event listener**
  - When checkbox selected, change color choices to return RGB values instead of color names
  - Action controller, action listener, and other form data shown in previous example

```
public void changeColorMode(ValueChangeEvent event) {
  boolean flag =
    ((Boolean)event.getNewValue()).booleanValue();
  setUsingColorNames(!flag);
}
```


Step 2: Create Input Form (and Specify Event Listener)

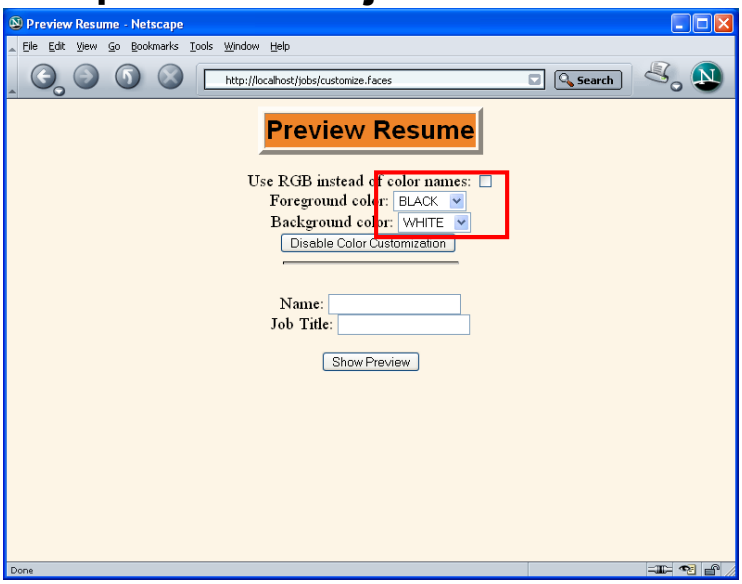
- **New features**
  - Use h:selectBooleanCheckbox to create checkbox
  - Use valueChangeListener to designate event listener code
  - Use onclick to automatically submit form when checkbox value changes
    - You could use onchange on Firefox, but with onchange in Internet Explorer, input focus also has to move
- **Example code**

```
<h:selectBooleanCheckbox
    valueChangeListener="#{resumeBean.changeColorMode}"
    onclick="submit()"
    immediate="true"/>
```

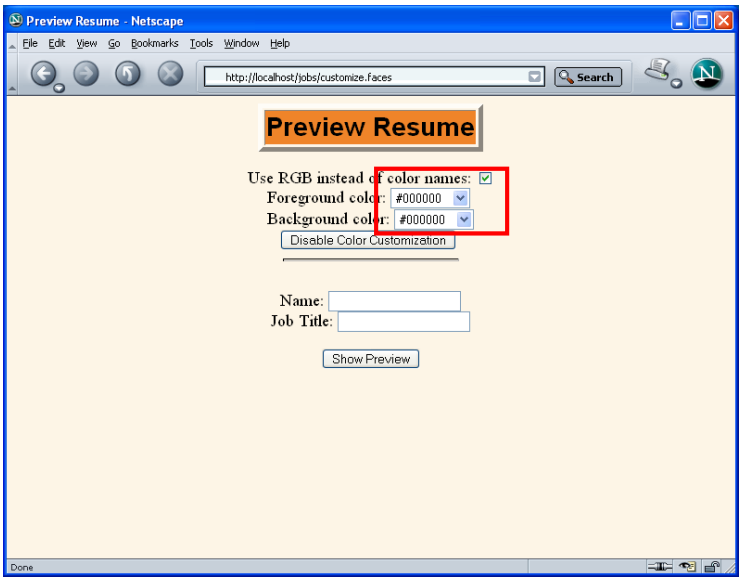
Client-side JavaScript code

Step 2: Initial Result

- File is *tomcat_dir*/webapps/jobs/customize.jsp
- URL is http://localhost/jobs/customize.faces



Steps 2: Result of Changing Checkbox (Independent of Action Controller)



Steps 3-5

- Unchanged from previous slides

Combining Action Listeners and Action Controllers for Same Button

Motivation

- **Usually, action listeners and action controllers are attached to different buttons**
- **Sometimes, you are forced to use both**
  - Listeners have access to low-level details of GUI object
    - Renderer, client ID, etc.
    - Especially relevant when we cover custom components
- **Most common example: server-side image maps**
  - h:commandButton with image instead of value results in image map
  - Actual incoming request parameter names are *clientID*.x and *clientID*.y
    - Only listener can discover the client ID

Example: Choosing Background Color from Image Map

- **Display grayscale color bar to user**
- **Let them click on desired color for background**
- **Build corresponding RGB value**
  - Read the *clientID*.x value
  - Normalize to 0-256 based on image width
  - Output as hex digits into #RRGGBB string
  - Done in action listener
- **Forward to JSP page**
  - Done in action controller

Steps in Using JSF

1) Create a bean
- A) Properties for form data
- B) Action controller and event listener methods
- C) Placeholders for results data

2) Create an input form
- A) Input fields refer to bean properties
- B) Button specifies action controller method that will return condition
- C) Button or other GUI control specifies event listener method

3) Edit faces-config.xml
- A) Declare the bean
- B) Specify navigation rules

4) Create results pages
- Output form data and results data with h:outputText

5) Prevent direct access to JSP pages
- Use a filter that redirects blah.jsp to blah.faces

Step 1: Create Bean

- **(A) Form data**

```
package coreservlets;

...

public class ColorBean {
  private String bgColor = "WHITE";

  public String getBgColor() { return(bgColor); }

  public void setBgColor(String bgColor) {
    this.bgColor = bgColor;
  }
```

Step 1: Create Bean

- **Listener looks up x value and creates color**

```
public void selectGrayLevel(ActionEvent event) {
  FacesContext context = FacesContext.getCurrentInstance();
  String clientId = event.getComponent().getClientId(context);
  HttpServletRequest request =
    (HttpServletRequest)context.getExternalContext().getRequest();
  String grayLevelString = request.getParameter(clientId + ".x");
  int grayLevel = 220;
  try {
    grayLevel = Integer.parseInt(grayLevelString);
  } catch(NumberFormatException nfe) {}
  // Image is 440 pixels, so scale.
  grayLevel = (256 * grayLevel) / 440;
  String rVal = Integer.toString(grayLevel, 16);
  setBgColor("#" + rVal + rVal + rVal);
}
```

- **Controller designates output page**

```
public String showPreview() {
  return("success");
}
```

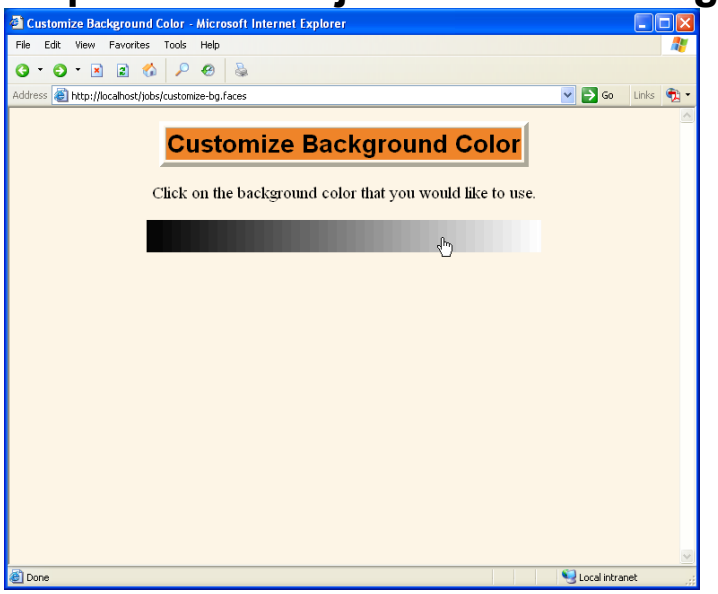

Step 2: Create Input Form

- **Specify both action and actionListener**
  - Listener runs first and has access to low-level internals
  - Also use h:commandButton with image instead of value
    - Results in server-side image map
- **Code**

```
<h:form>
  ...
  <h:commandButton
      image="images/GrayBar.gif"
      actionListener="#{colorBean.selectGrayLevel}"
      action="#{colorBean.showPreview}"/>
</h:form>
```

Step 2: Result

- File is *tomcat_dir*/webapps/jobs/customize-bg.jsp
- URL is http://localhost/jobs/customize-bg.faces



Step 3: Edit faces-config.xml

- Declaring bean

```
...
<faces-config>
  ...
  <managed-bean>
    <managed-bean-name>colorBean</managed-bean-name>
    <managed-bean-class>
      coreservlets.ColorBean
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
  ...
</faces-config>
```

Step 3: Edit faces-config.xml

- **Specifying navigation rules**

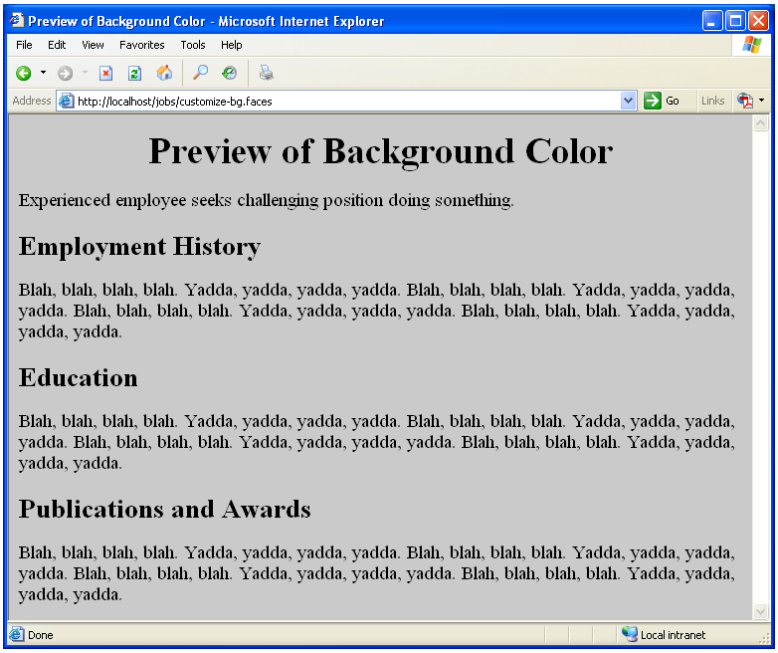
```
…
<faces-config>
  …
  <navigation-rule>
    <from-view-id>/customize-bg.jsp</from-view-id>
    <navigation-case>
      <from-outcome>success</from-outcome>
      <to-view-id>
        /WEB-INF/results/show-preview2.jsp
      </to-view-id>
    </navigation-case>
  </navigation-rule>
</faces-config>
```


Step 4: Create Results Pages

- **WEB-INF/results/show-preview2.jsp**

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<f:view>
<!DOCTYPE ...>
<HTML>
<HEAD><TITLE>Preview of Background Color</TITLE></HEAD>
<BODY BGCOLOR="<h:outputText value="#{colorBean.bgColor}"/>">
<H1 ALIGN="CENTER">Preview of Background Color</H1>
Experienced employee
seeks challenging position doing something.
<H2>Employment History</H2>
Blah, blah, blah, blah. Yadda, yadda, yadda, yadda.
…
</HTML>
</f:view>
```

Step 4: Example Result



Step 5: Prevent Direct Access to JSP Pages

- **Use filter that captures url-pattern *.jsp**
  - No changes from previous examples

Summary

- **Event listeners are used to handle events that affect only the user interface**
  - Typically fire before beans are populated and validation is performed
    - Use immediate="true" to designate this behavior
  - Form is redisplayed after listeners fire
    - No navigation rules apply
- **Action listeners**
  - Attached to buttons, hypertext links, or image maps
  - Automatically submit the form
- **Value change listeners**
  - Attached to radio buttons, comboboxes, list boxes, checkboxes, textfields, etc.
  - Require onclick="submit()" or onchange="submit()" to submit form
    - Test carefully on all expected browsers

Questions?