11

Developing Web Interfaces Using JSF

Objectives

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

- Describe the purpose of JavaServer Faces
- Use JSF components
- Explain the use of managed beans
- Describe the life cycle of a JSF application
- Explain the role of the JSF tag libraries
- Describe how JDeveloper supports JSF
- Create a JSF-based JSP in JDeveloper

JSF: Overview

- JavaServer Faces (JSF) is a server-side component framework for Web applications.
- JSF:
  - Is an implementation of the MVC design pattern
  - Enables separation of business and presentation logic
  - Enables separation of navigational and data flow

JavaServer Faces 2.0 New Features

JSF 2.0 provides the following benefits over JSF 1.2:

- AJAX tags and framework
- An annotation-based configuration model that can be used in place of XML descriptors (`faces-config.xml`)
- Ability to bookmark JSF pages
- Improved error handling
- New default page templating framework (Facelets)
- Intelligent defaults for page navigation rules

JSF: Benefits

- Simplifies Java EE development
- Is intended for RAD style development tools
- Is a component-based architecture
- Supports Java EE framework:
  - State management
  - Error handling
  - Input validation
  - Type conversion
  - Event handling
  - Page navigation
- Is portable across JSF implementations

JSF Hello World

Create an `.xhtml` file and place it in a Java EE web application.

```
1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml"
5        xmlns:h="http://java.sun.com/jsf/html">
6     <h:head>
7        <title>Hello World</title>
8    </h:head>
9     <h:body>
10       <h:outputText value="A simple JSF Facelet Example"/>
11    </h:body>
12 </html>
```

Key Terms

- *UI component*: The components that manage data to be displayed on a page
- *Managed bean*: Objects that maintain data and methods that are used across multiple pages
- *Expression Language*: Shortcut to access properties and methods from a JSF page
- *Navigation model*: The rules that govern page flow
- *Life cycle*: The processes that are executed from the time a page is loaded until it is complete

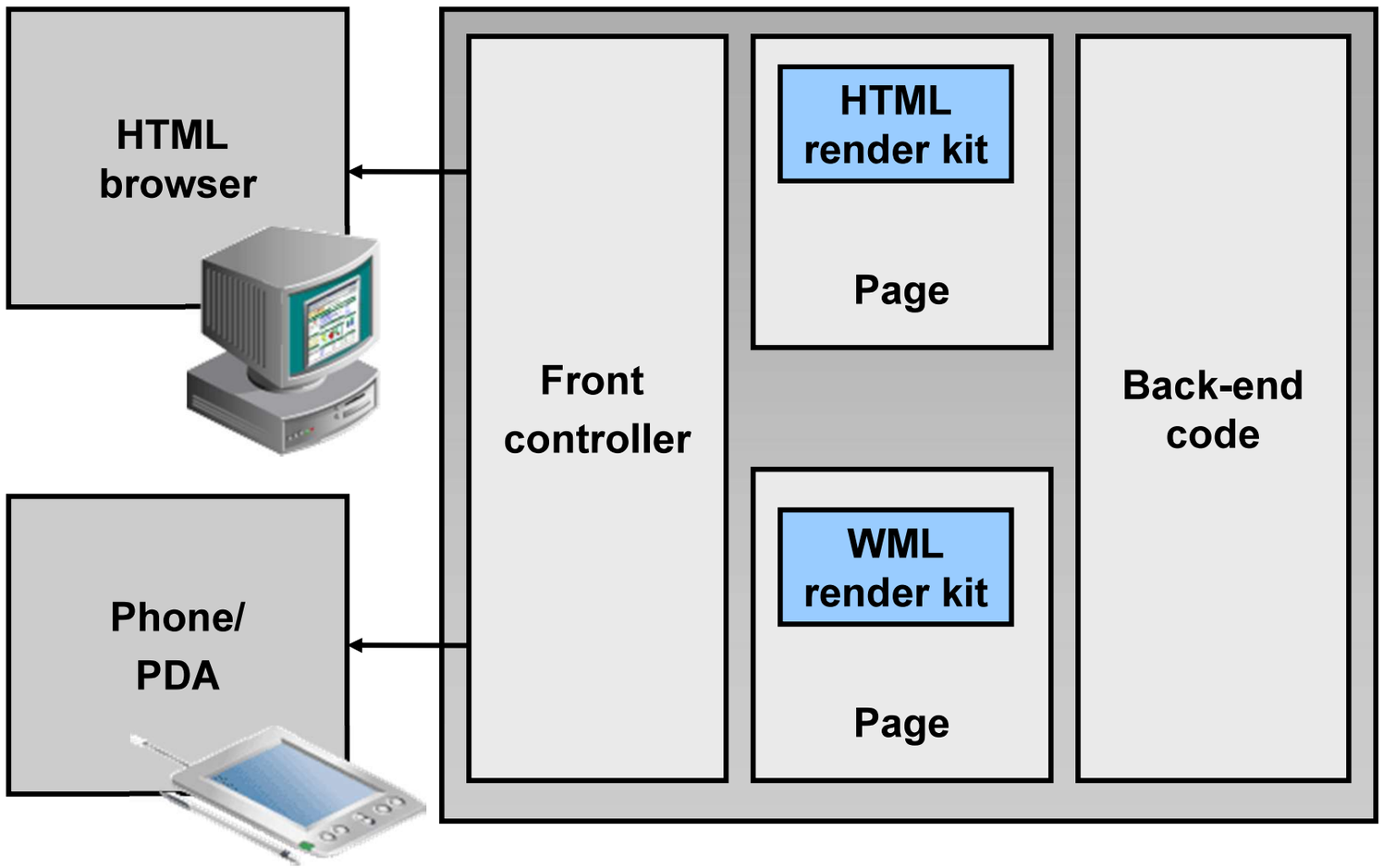
JSF Architecture

JavaServer Faces has:

- A set of prefabricated UI components
- An event-driven programming model
- A component model that enables third-party developers to build additional components

It can be thought of as “Swing for server-side applications.”

JSF Architecture



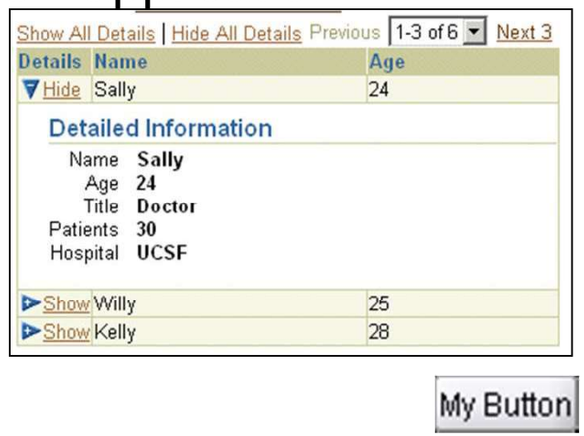
JSF Components

JSF Components consists of three parts:

- *UI components*: Functionality, attributes, or behavior
- *Renderers*: Converts components to and from a specific markup language
- *Render kits*: Library of renderers (The Basic HTML RenderKit is part of the specification.)

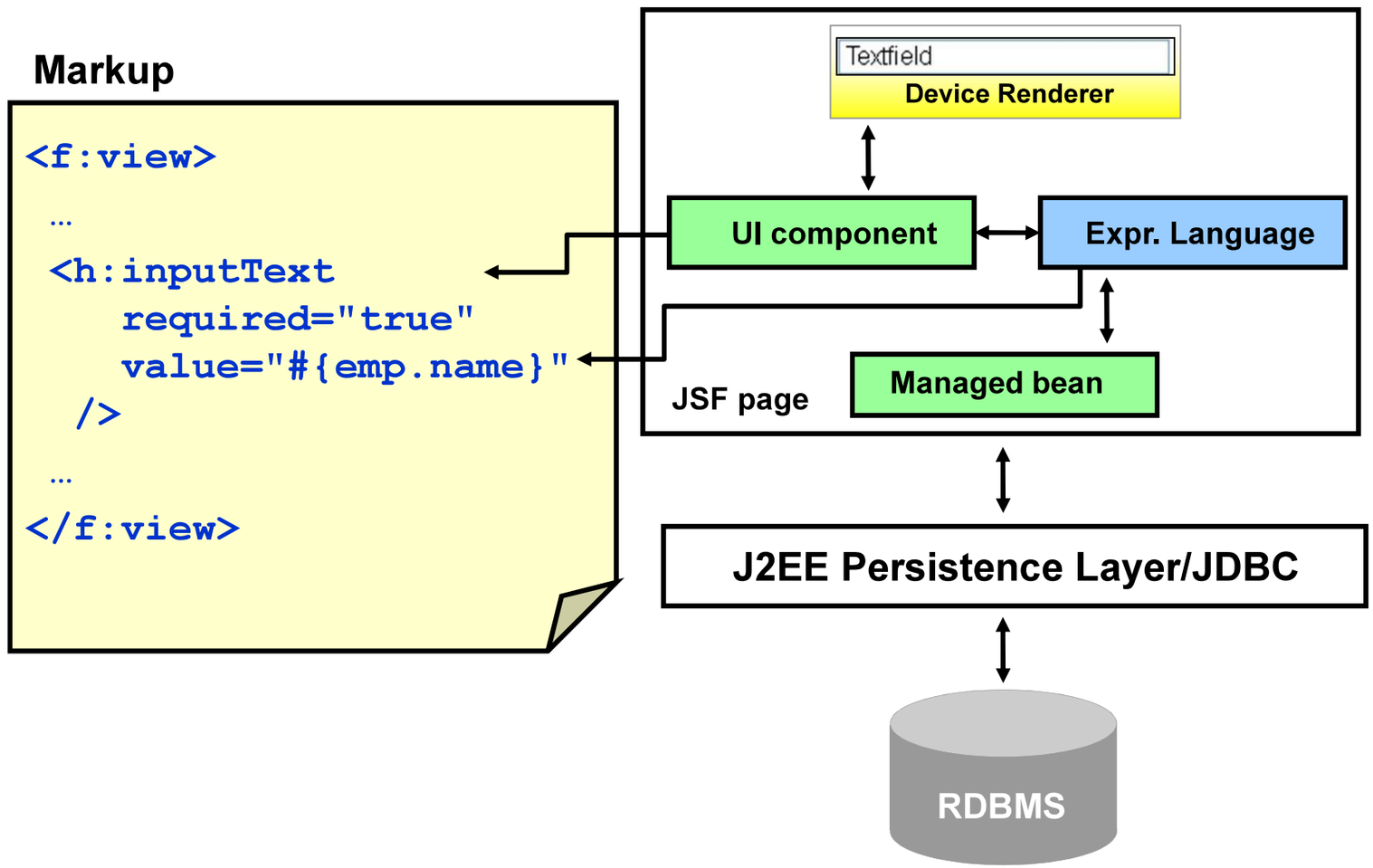
JSF UI Components

- Are the basic building blocks of a JSF application
- Are stateful server objects
- Do not define rendering
- Can represent simple to complex user interface components ranging from a button or input field to a complete page



- Can be associated to model data objects through value binding
- Can use helper objects, such as validators, converters, listeners, and events

JSF Component Architecture

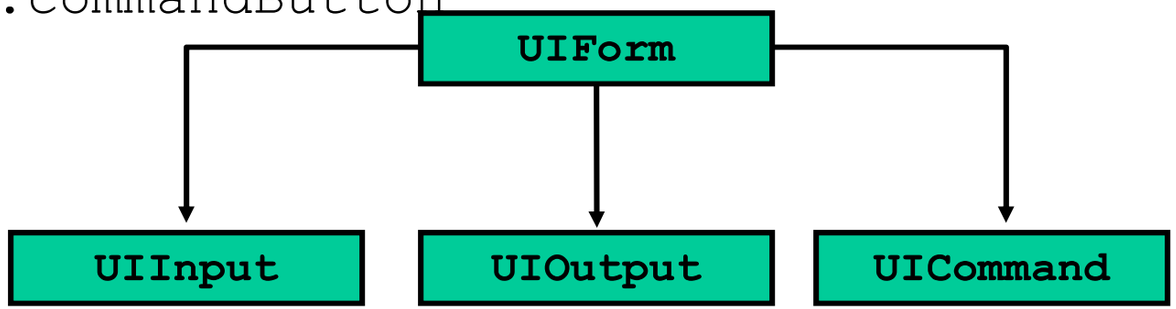


Identify the correct statements about JavaServer Faces.

1. It includes a set of APIs for representing the user interface (UI) components and managing their state, handling events and input validation, converting values, defining page navigation, and supporting internationalization and accessibility.
2. It includes a default set of UI components.
3. It includes a server-side event model.
4. It includes state management.
5. It includes managed beans (JavaBeans created with dependency injection).

Tag Handlers and Component Trees

- Each tag has a corresponding tag handler class.
- JSF tag handlers collaborate with each other to build a component tree.
- Examples of JSF tags are: `f:form`, `h:inputText`, `h:inputSecret`, `h:commandButton`



Tag Libraries

- Core tag library
  - `f:` tags for faces
  - Custom actions independent of any rendering kit
  - Example: `f:validator, f:converter, f:selectItem`
- HTML tag library
  - `h:` tag for HTML
  - Component tags for all UI components and HTML rendering kit
  - Example: `h:form, h:inputText, h:messages`

Configuration Files

- `faces-config.xml`
  - Defines managed beans
  - Specifices navigation rules
  - Must exist with `WEB-INF`
  - Similar to `struts-config.xml`
- Servlet mapping in `web.xml` uses a faces extension.
- URL pattern `*.faces` maps to `java.faces.webapp.FacesServlet`.

JSF Renderers

- Two rendering models:
  - Direct implementation
  - Delegated implementation
- Delegated implementation enables separation of UI from how they appear to the user.
- Each JSF implementation must provide a default RenderKit instance.
- JSF specification includes the Standard HTML RenderKit Specification.

Managed Beans

- Java objects (empty constructor), maps, lists
- Defined in `faces-config.xml`
- Defined with various scopes:
  - application
  - session
  - request
  - none
- Lazy initialization by JSF as needed

Managed Bean Configuration

After creating a JavaBean it must be configured to act as a managed bean. Configuration can be done in two ways:

Annotation-based configuration (JSF 2.0)

```
@ManagedBean
public class NumberGameBean {...}
```

XML-based configuration (`faces-config.xml`)

```
<managed-bean>
  <managed-bean-name>...</managed-bean-name>
  <managed-bean-class>...</managed-bean-class>
</managed-bean>
```

Managed Bean Life Cycle and Scope

Managed beans are instantiated when they are first used in an EL expression on a page. By default, managed beans are discarded after each request. The lifetime of a bean can be extended by configuration, for example:

```
@ManagedBean(name="gameBean")
@SessionScoped
public class NumberGameBean {...}
```

Commonly used scopes are:

`@SessionScoped`

`@RequestScoped`

`@ApplicationScoped`

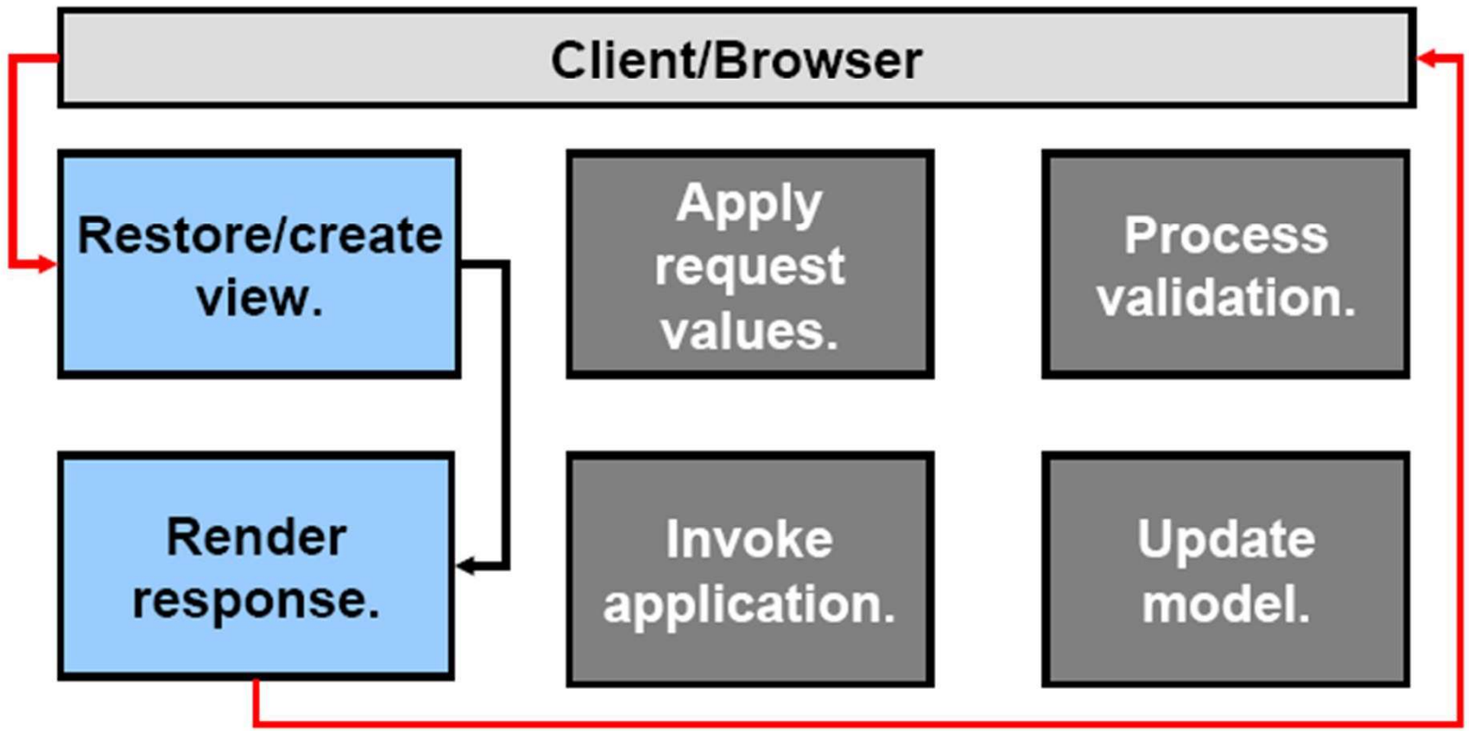
Expression Language

- Dot notation for attributes (JavaBean model):
  `#{userbean.name}` same as `instance.getName();`
- Map notation:
  `#{foo["baa"]}` same as `instance.get("baa");`
- Expressions can be of any depth:
  `#{foo["baa"].person.name}`

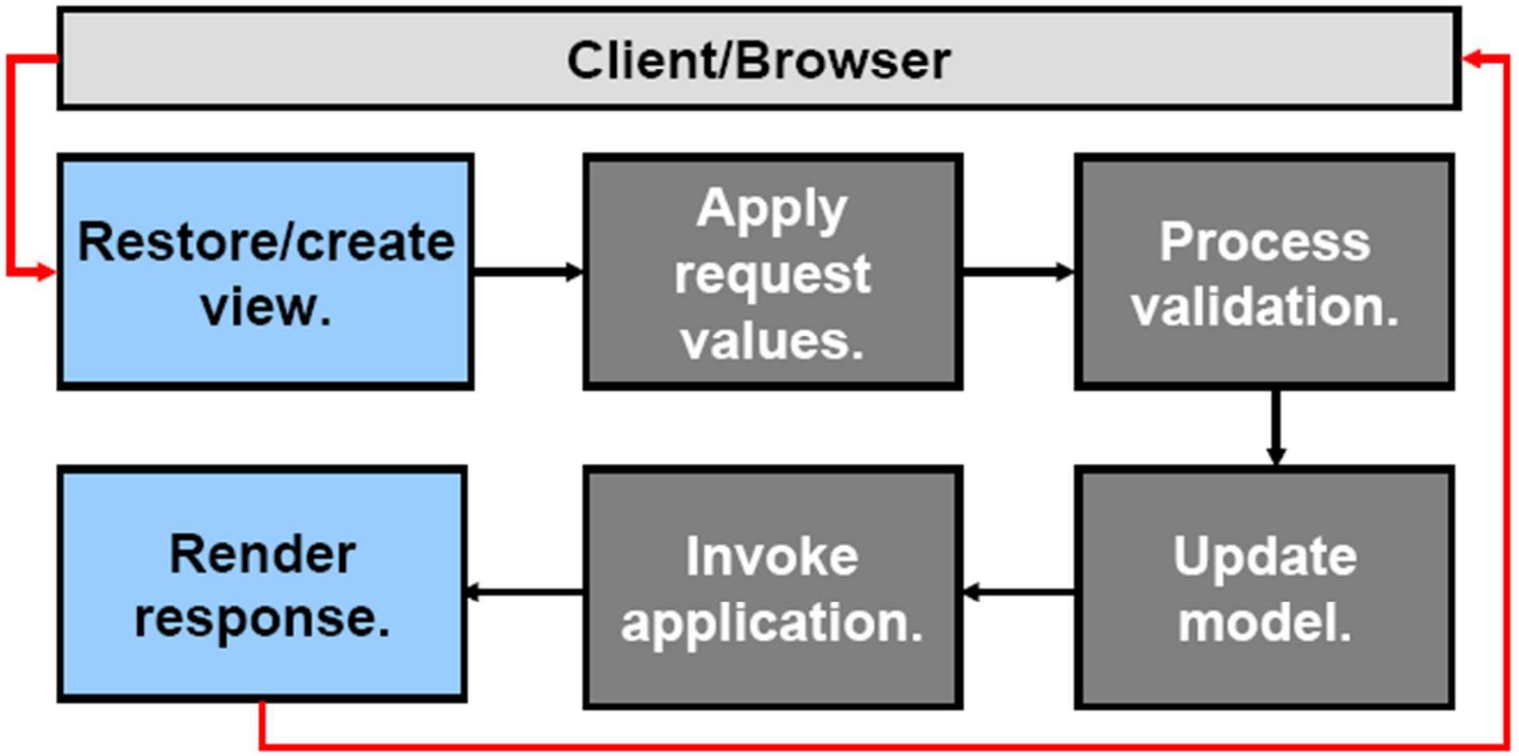
Life Cycle of a JSF Page

- A JSF page is represented by a tree of UI components, called a view.
- When a client makes a request for the page, the life cycle starts.
- During the life cycle, JSF implementation must build the view while considering the state saved from the previous postback.
- When the client performs a postback of the page, JSF implementation must perform life-cycle steps:
  - Validation
  - Conversion

JSF Life Cycle: Initial Request



JSF Life Cycle: Postback

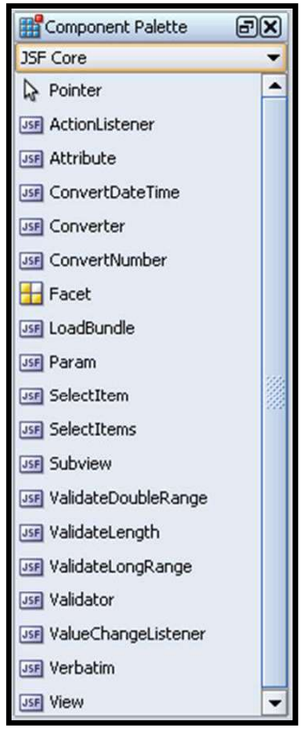


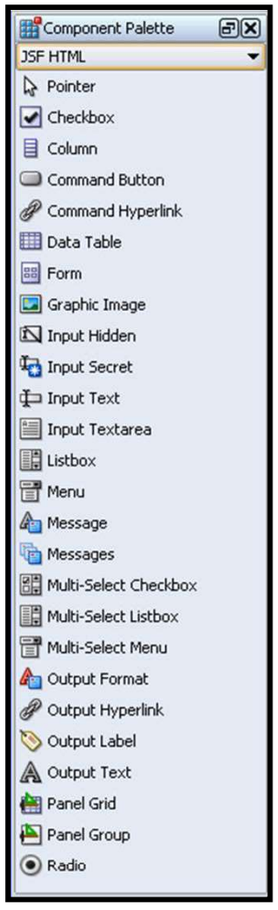
Backing beans are JSF-managed beans that are associated with the UI components on a JSF page.

1. True
2. False

JSF UI components are encapsulated in JSP tag libraries:

- Core: For application tasks, such as validation, and data type conversion
- HTML: For rendering basic HTML, such as input fields, menus, tables, and buttons



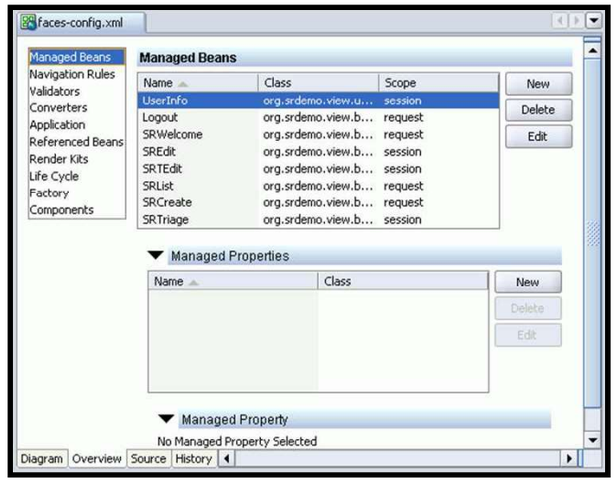


A typical JSF application consists of:

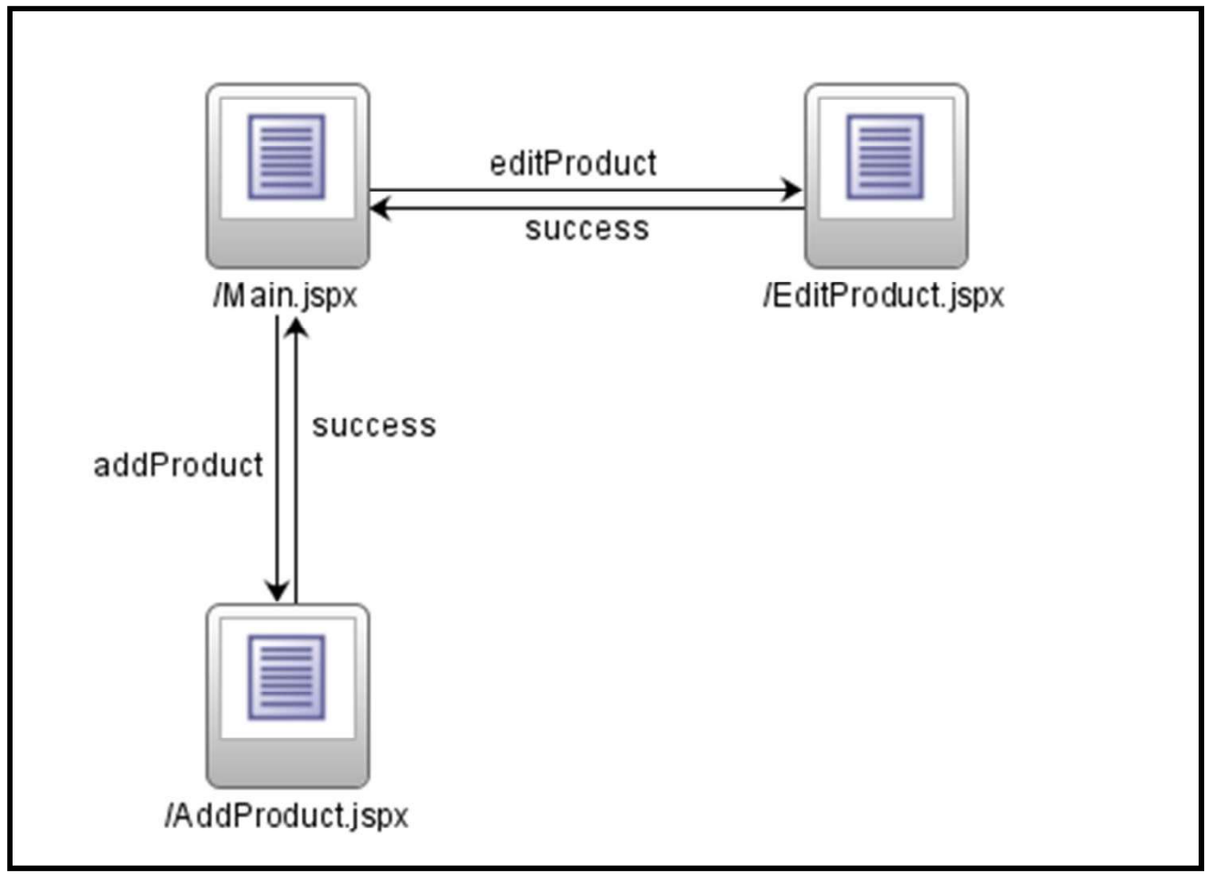
- One or more JSPs containing JSF components
- A navigation model specified in the `faces-config.xml` file
- A set of managed beans that facilitate the UI logic of the application

JDeveloper provides many features for creating JSF components:

- JSF visual editing
  - JSF UI component visual editing
  - Provides back-end code generation (double-click)
- JSF Configuration Editor for productive editing of `faces-config.xml`

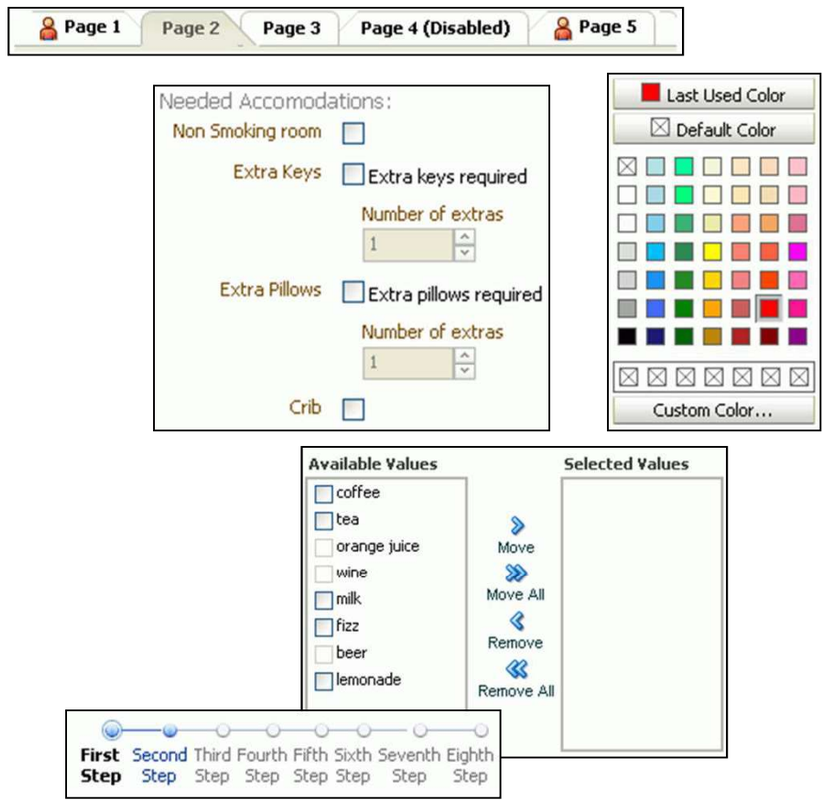


JSF Navigation Diagram



Adding to JSF with ADF Faces

- Built on top of JSF APIs
- Much larger component set: over 100 types
- More advanced and interesting components
  - Partial-page rendering
  - Scrollable, sortable table
- Rich feature set for customizing applications
- Uses Ajax, SVG, and Flash
- ADF model support
- Runs on any JSF-compliant implementation

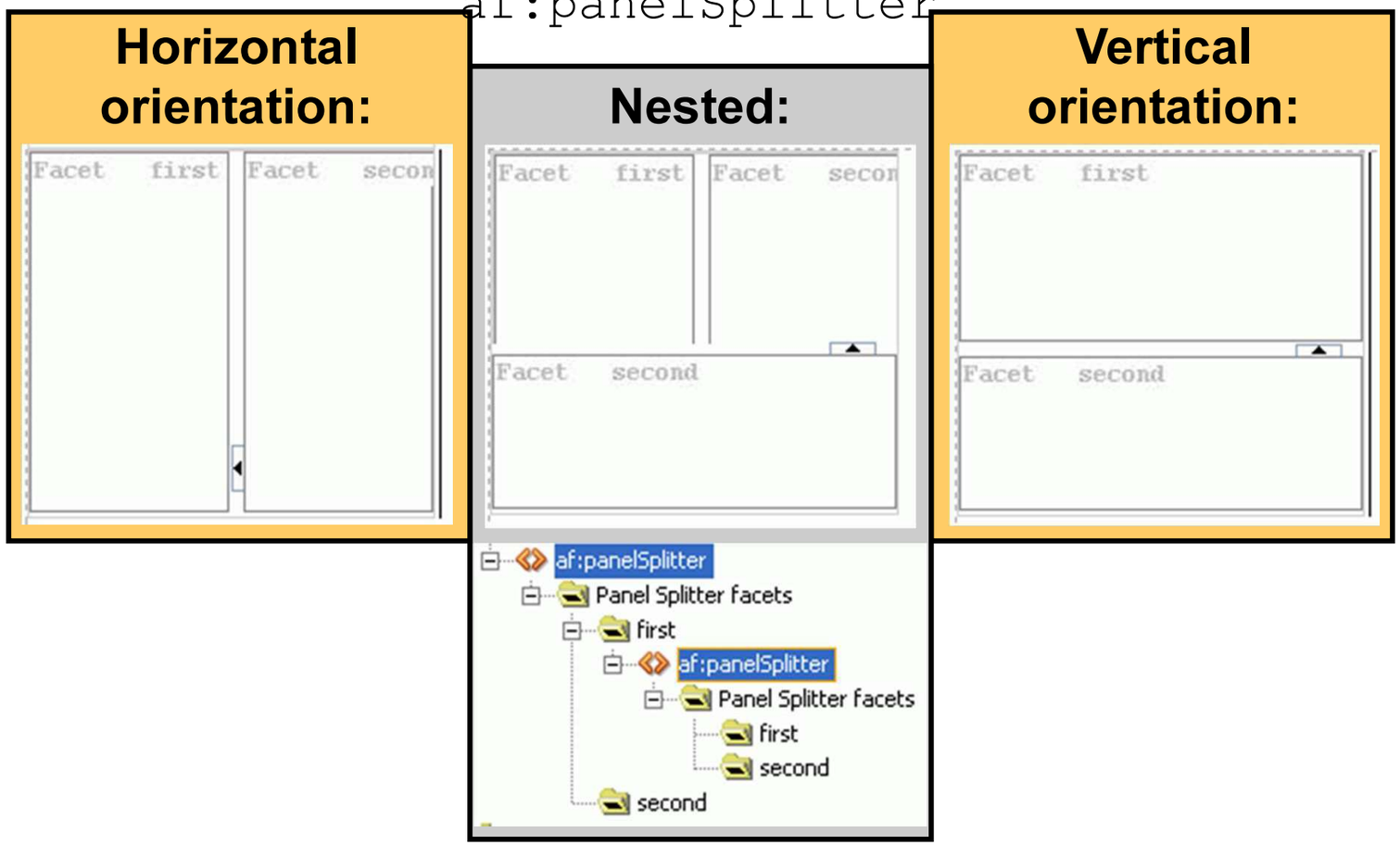


You can use the following components to achieve the desired layout:

- af:spacer
- af:separator
- af:panelSplitter
- af:panelStretchLayout
- af:panelAccordion
- af:panelFormLayout
- af:panelTabbed
- af:showDetail
- af:panelGroupLayout
- af:panelCollection
- af:panelHeader
- af:showDetailHeader
- af:group
- af:panelList
- af:panelBox
- af:panelBorderLayout

Panel Splitters

`af:panelSplitter`



Creating Collapsible Panes

Attributes:

collapsed	positionedFromEnd	Behavior
`true`	`false`	First pane hidden; second pane stretches
`true`	`true`	Second pane hidden; first pane stretches
`false`	`true`	Both panes displayed, with `splitterPosition` governing the size of the second pane and splitter arrow pointed toward the second pane
`false`	`false`	Both panes displayed, with `splitterPosition` governing the size of the first pane and splitter arrow pointed toward the first pane

Creating Collapsible Panes

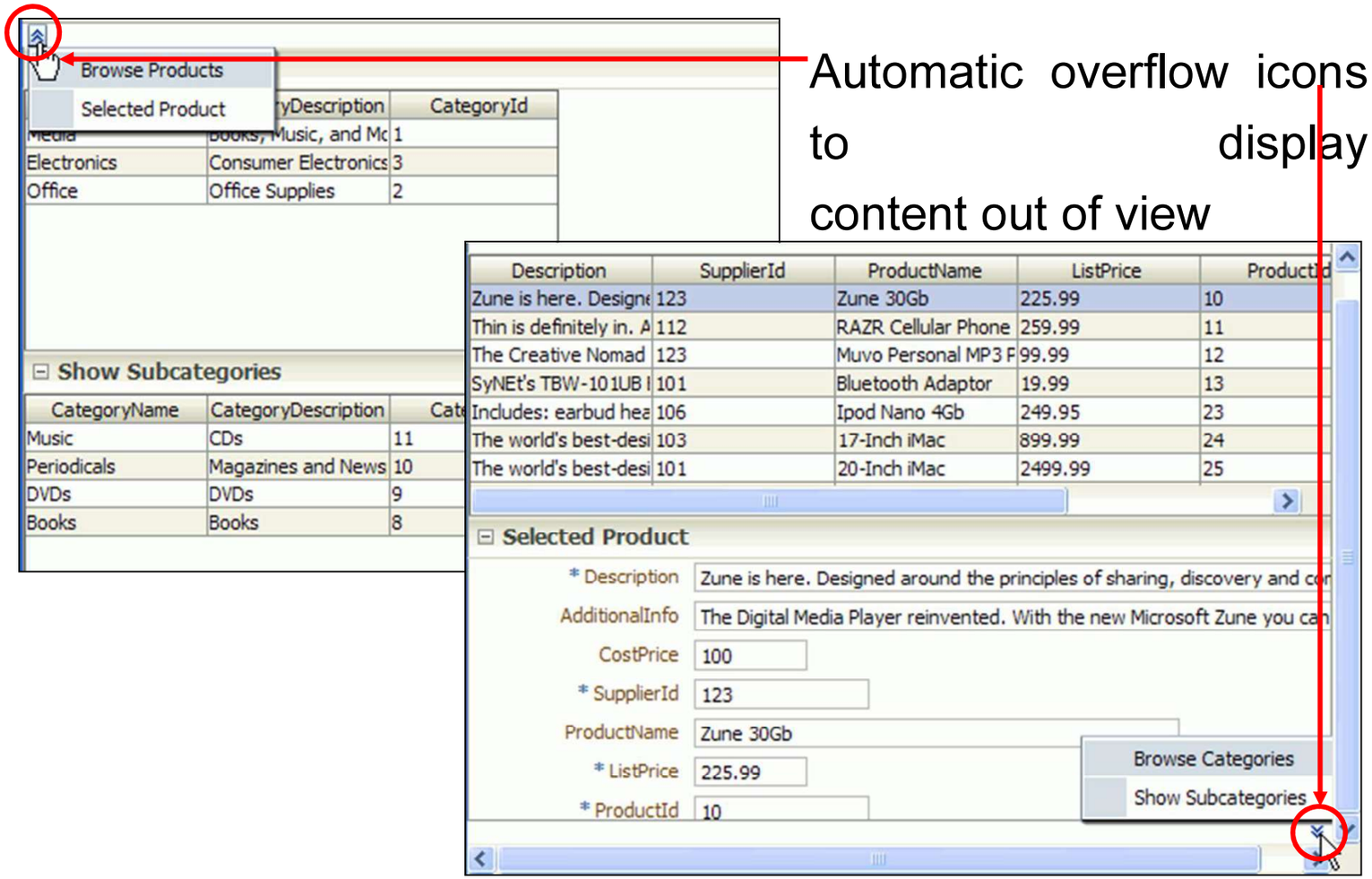
With default settings

Characteristics of Panel Accordion component:

- **Does not automatically stretch it children**
- **Panes cannot be resized at run time—only expand or contract**
- **Panes defined by `showDetailItem`**

```
<af:panelAccordion>
  <af:showDetailItem text="This is a pane"/>
  <af:showDetailItem text="This is another pane"/>
</af:panelAccordion>
```

Panel Accordion Overflow



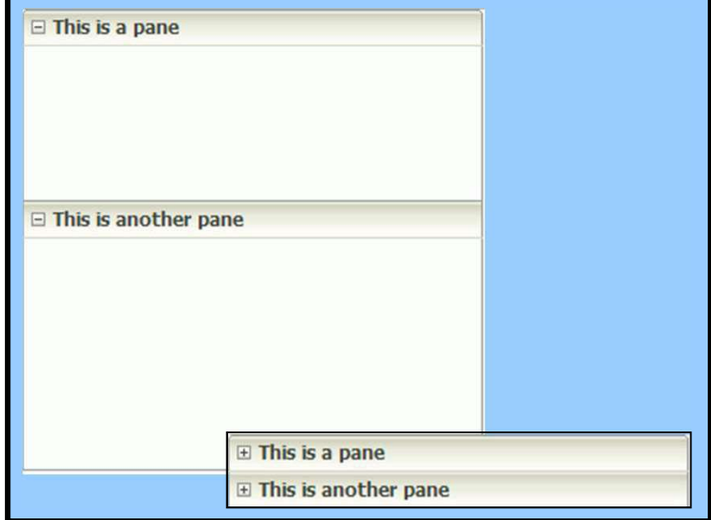
Automatic overflow icons to display content out of view

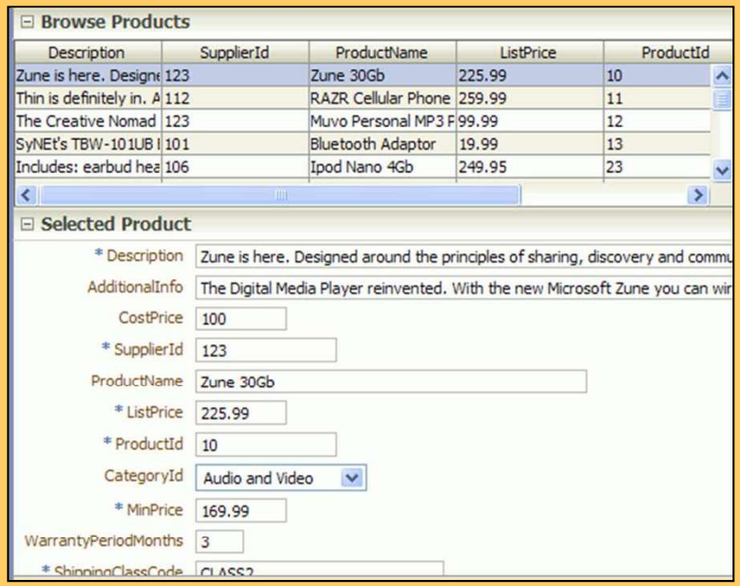
Setting Panel Accordion Properties

```
<af:showDetailItem
 text="Browse Products"
inlineStyle="width:100%;
 height:100%;" flex="1">
```

```
<af:showDetailItem
 text="Selected Product"
inlineStyle="width:100%;
 height:100%;" flex="2">
```

With `discloseMany="true"`
`discloseNone="true"`



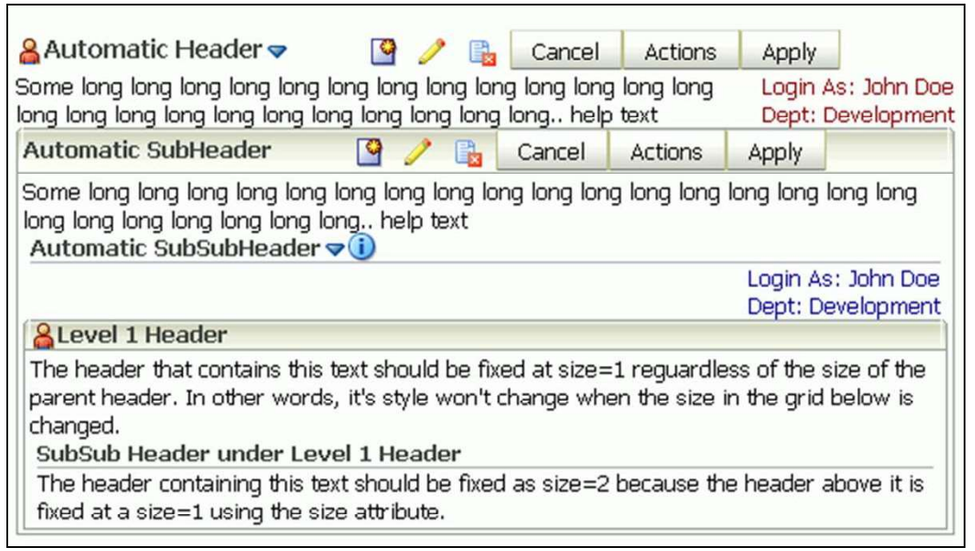


Notes page only

Creating Titled Sections and Subsections

Panel header component with sections and subsections:

`af:panelHeader`

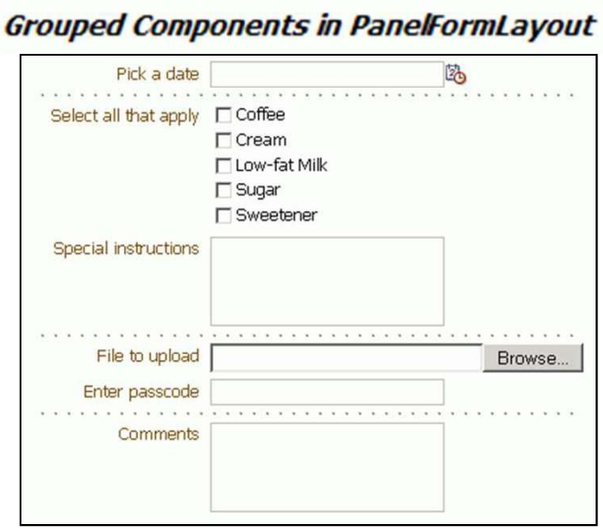


Show detail header component with sections that expand or collapse: `af:showDetailHeader`

Grouping Related Components

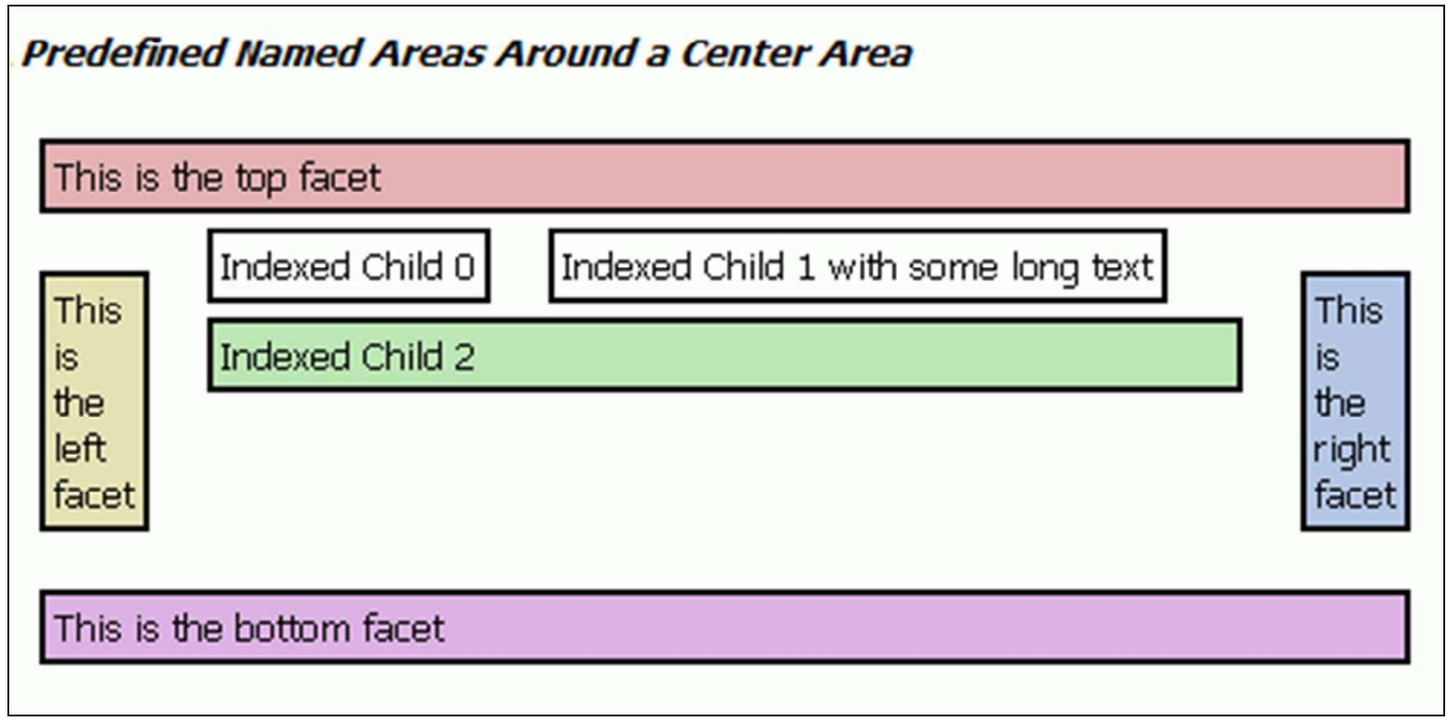
Use `af:group` to:

- Add multiple components to a `facet`
- Group related child components




```
<af:panelFormLayout>
  <af:inputDate label="Pick a date"/>
  <!-- first group -->
  <af:group>
  <af:selectManyCheckbox label=
    "Select all that apply">
    <af:selectItem label="Coffee" value="1"/>
    //other select items
  </af:selectManyCheckbox>
    <af:inputText label="Special instructions"
    rows="3"/>
  </af:group>
  <!-- Second group -->
  <af:group>
    <af:inputFile label="File to upload"/>
      <af:inputText label="Enter passcode"/>
  </af:group>
  <af:inputText label="Comments" rows="3"/>
  <af:spacer width="10" height="15"/>
  <f:facet name="footer"/>
</af:panelFormLayout>
```

Arranging Items Around a Central Area



Notes page only

Summary

In this lesson, you should have learned how to:

- Describe the purpose of JavaServer Faces
- Use JSF components
- Explain the use of managed beans
- Describe the life cycle of a JSF application
- Explain the role of the JSF tag libraries
- Describe how JDeveloper supports JSF
- Create a JSF-based JSP in JDeveloper

Practice 11: Overview

This practice covers the following topics:

- Creating a Managed Bean to support a page
- Creating a basic JSF page
- Adding layout components to the JSF page