НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Java Server Faces

Eugeny Berkunsky, NUoS
eugeny.berkunsky@gmail.com
http://berkut.mk.ua

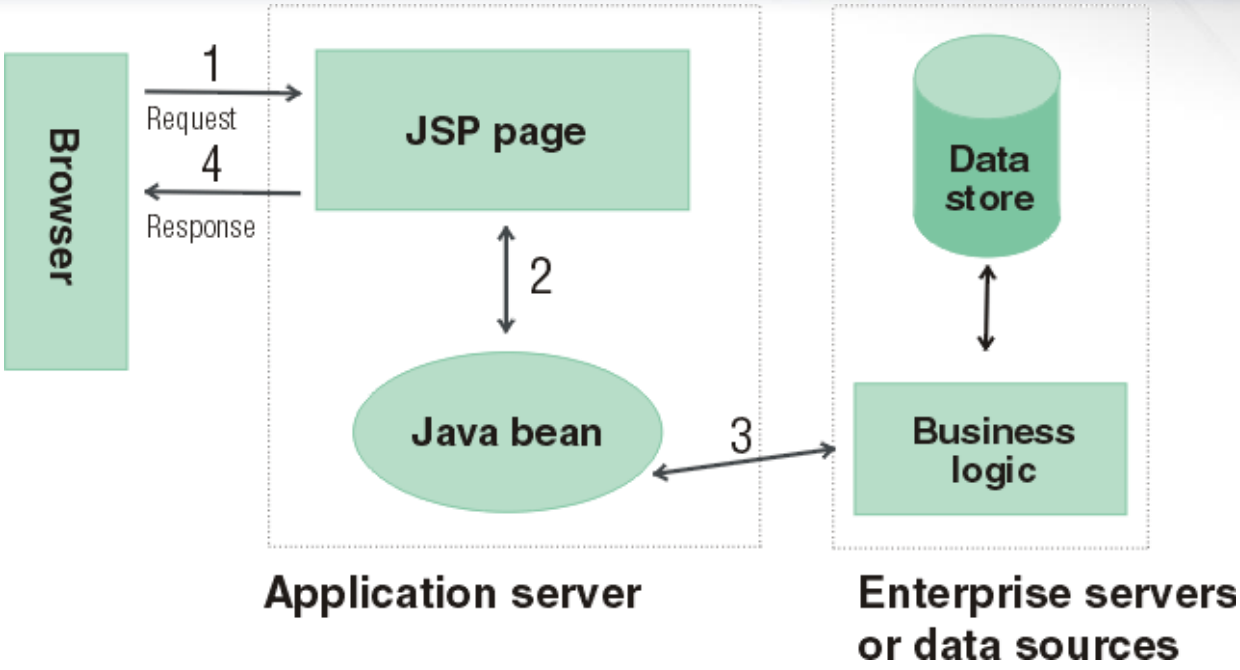
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Contents

JSP Architectures

Model 1

Model 1 (1/2)



- **Web browser directly accessing Web-tier JSP pages**
- **The JSP pages access Web-tier JavaBeans that represent the application model**

Model 1 (2/2)

- The next view to display is determined by
  - Hyperlinks selected in the source document
  - Request parameters
- Application control is decentralized
  - Current page being displayed determines the next page to display
- Each JSP page or servlet processes its own inputs
  - Parameters from GET or POST

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

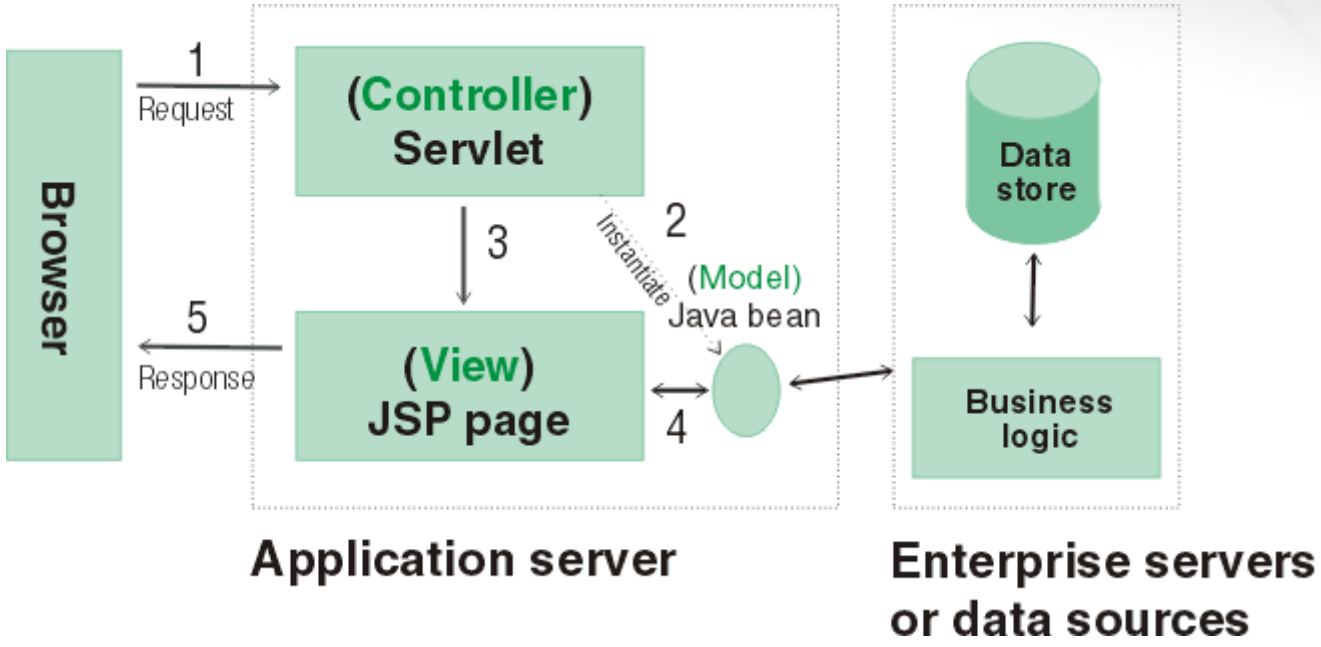
Model 1 (When to Use?)

- Provide a more lightweight design for small, static applications
- Applications which
  - Have very simple page flow
  - Have little need for centralized security control or logging
  - Changes little over time

JSP Architectures

Model 2

Model 2 (MVC)



- **Introduces a controller servlet between the browser and the JSP pages or servlet content being delivered**
- **Views do not refer to each other directly**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Controller Servlet

- Centralizes logic for
  - Dispatching of requests to the next view based on the request URL
  - Input parameters
  - Application state
- Handles view selection
  - Decouples JSP pages and servlets from one another
- Provides
  - Single point of control for security and logging
  - Encapsulation of incoming data into a form usable by the back-end MVC model

Model 2 - Advantages

- Easier to maintain and extend
- There are many ready frameworks so we do not have to write our own
  - Struts
  - Tapestry
  - Spring Web Flow
  - WebWork
  - JavaServer Faces
  - … and many others

Introduction to JavaServer Faces

What is JSF?

What is JavaServer Faces?

- **JavaServer Faces is:**
  - **Web Application Framework**
  - **Request-driven MVC**
  - **Uses component-based approach**
  - **Uses JSP for its display technology, but is not limited to it**

What is JavaServer Faces?

- JSF Includes
  - Set of APIs
  - Two JSP custom tag libraries for expressing UI within JSP
  - Server-side event model
  - State management
  - Managed beans
  - Unified Expression Language

Faces Servlet

- The `FacesServlet`
  - Accepts all incoming JSF requests
  - Initializes resources
- Passes requests to the request lifecycle for processing
- Faces servlet plays the role of the controller servlet in MVC architecture

Faces Servlet - Mapping

- **Mapping for the `FacesServlet` in the `web.xml`**

```
<servlet>
  <display-name>FacesServlet</display-name>
  <servlet-name>FacesServlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>FacesServlet</servlet-name>
  <url-pattern>/some-url-pattern</url-pattern>
</servlet-mapping>
```

- **There are two standard ways to map the faces servlet**
  - **`/faces/*` - prefix mapping**
  - **`*.xhtml` – suffix mapping**

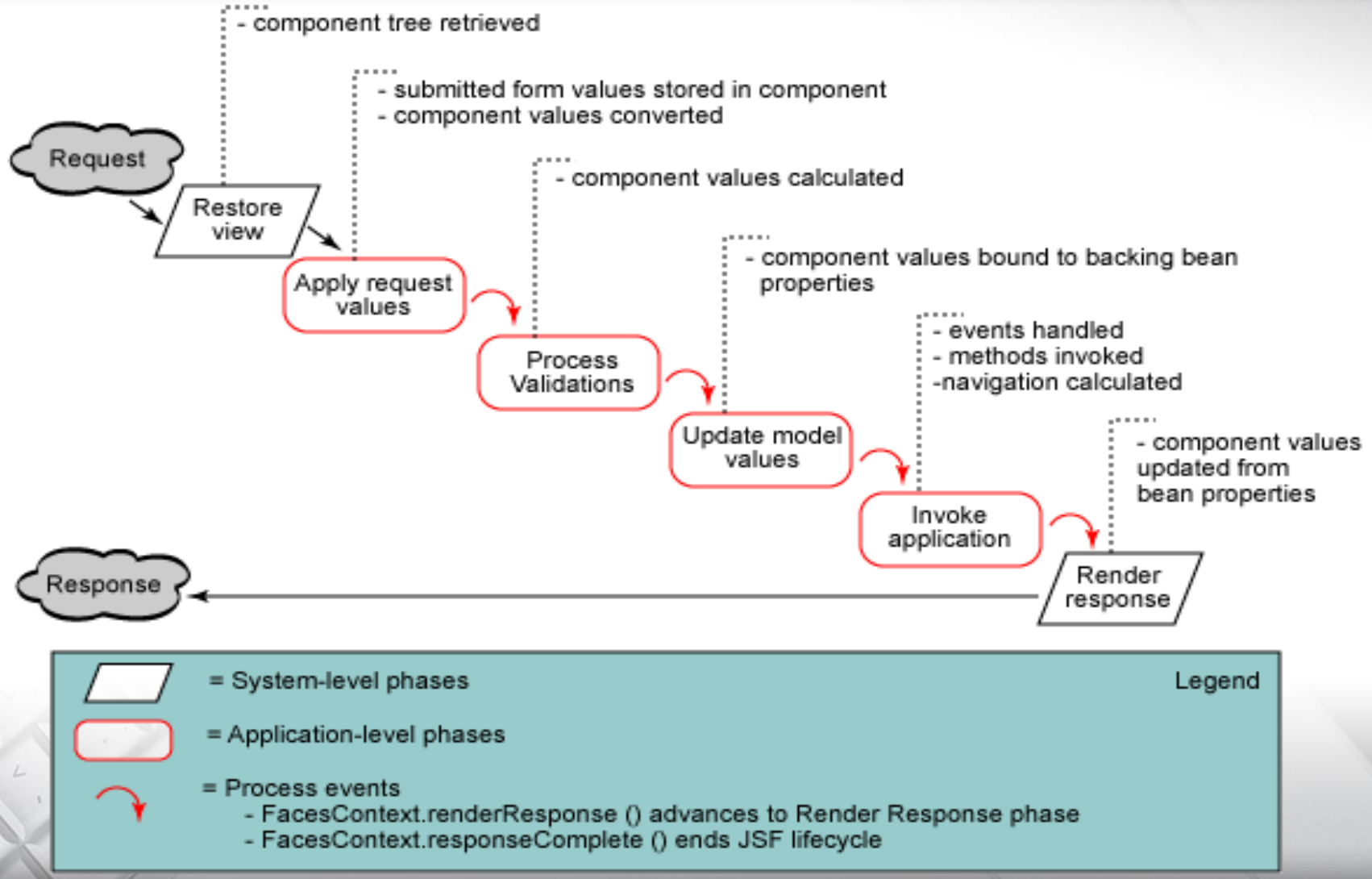
JSF View

- A JSF page is represented by a tree of UI components, called a **view**
  - Almost always the view is described with JSP pages (but not mandatory)
- When a client makes a request for the page, if it is caught by the Faces Servlet and the JSF request lifecycle starts
- During the lifecycle, JSF implementation must build the view considering the state saved from the previous postback

Lifecycle Phases

- Request lifecycle consists of 6 phases:
  - Restore View Phase
  - Apply Request Values Phase
  - Process Validations Phase
  - Update Model Values Phase
  - Invoke Application Phase
  - Render Response Phase
- These phases are described in details at the end of the lecture

Request Lifecycle Diagram



JSF Hello World Application

JSF Hello Application

- To make the simplest JSF application you should perform the following steps:
  - Create new Web project
  - If your application container does not support JSF (for example Tomcat) you should add the JSF API and Implementation in your `lib` folder
  - JSF 1.2 also requires JSTL 1.2 in your libraries
  - For my-faces following jars are also required:
    - `commons-beanutils-1.7.0.jar`, `commons-codec-1.3.jar`, `commons-collections-3.2.jar`, `commons-digester-1.8.jar`, `commons-discovery-0.4.jar`, `commons-logging-1.1.1.jar`

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

web.xml

- In the `web.xml` file we define
  - Welcome file – `index.html`
  - Define Faces Servlet and its mapping

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.xhtml</url-
  pattern>
</servlet-mapping>
```

If non-negative, the Web container will load this servlet on deployment

Faces Servlet is mapped to all requests that ends with `.jsf`

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

index.html and hello.xhtml

- **index.xhtml**

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Welcome to JSF Hello World Demo</title></head>
<body>
  <a href="hello.jsf">Go to Hello JSF Page</a>
</body>
</html>
```

- **When Faces Servlet sees `*.xhtml` it will look for template to create the view**
- **In `hello.xhtml` all content that is JSF specific should be in enclosed in `<f:view>` tag**
- **We also need to specify the needed JSF tag libraries**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

hello.xhtml

- **Here we do 3 things:**
  - **Define "JSF core" and "JSF html" tag libraries**
  - **JSF view (`f:view` tag)**
  - **Use `outputText/outputLabel` component to show message**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
  Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
  transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
...
  <f:view>
    <h:outputLabel value="Hello World" />
  </f:view>
...
</html>
```

JSF Hello World Application

Live Demo

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Application Configuration File (`faces-config.xml`)

- **`faces-config.xml` defines**
  - **Page navigation rules**
  - **Configures managed beans**
  - **Other custom objects, such as custom components**
- **It is located in the `WEB-INF` folder of the project**
- **Basic structure:**

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="4.0" xmlns="https://jakarta.ee/xml/ns/jakartaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
              https://jakarta.ee/xml/ns/jakartaee/web-facesconfig_4_0.xsd">

</faces-config>
```

JSF Component Model

Component Model (1/2)

- A set of **UIComponent classes** for specifying the state and behaviour of UI components
- A **rendering model** that defines how to render the components in different ways
- One UI component may have several presentations
  - Example: `UICommand` – button and hyperlink

Component Model (2/2)

- A server-side **event-listener model** that defines how to handle UI 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

JSF UI Components

What is an UI Component?

- Configurable, reusable elements that compose the user interfaces of JSF applications
- UI components can be:
  - Simple, like a button or text box
  - Compound, like a table, which can be composed of multiple other components
- Accessible via JSF tags in xhtml page

UI Component Classes

- The JSF implementation provides a set of UI component classes
  - Developers can extend the UI component classes to create custom UI components
  - Third party component libraries are available: PrimeFaces, RichFaces, ICEfaces, etc.
- All JSF UI component classes extend `UIComponentBase`
  - `UIComponentBase` defines the default state and behaviour of an UIComponent

UI Component Classes (1/2)

- **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 (2/2)

- A JSF UI component is typically a collection of classes and resources:
  - UIComponent Class – the core logic of the component, e.g. `HtmlCommandLink`
  - Renderer Class, e.g. `HtmlLinkRenderer`
  - UI Component Tag Class, e. g. `HtmlCommandLinkTag`
  - Tag Library Descriptor File (`*.tld`), e.g. `html-basic.tld`
  - Other classes: converters, validators, action listeners, etc.

Component Rendering Model

- Usually the components do not specify how they are rendered
- Each component can have multiple renderers which render it to one or more clients
- Render Kit – defines how components are rendered for a particular client
- The JSF implementation includes a standard HTML Render Kit for rendering its components to an HTML client

JSF Commonly Used Tags

Commonly Used Tags – JSF Core Library (1/2)

- `<f:view>` – root element for all JSF pages
- `<f:converter>` – creates an instance of the specified converter and binds it to its parent component
- `<f:validator>` – creates an instance of the specified validator and binds it to its parent component (`UIInput` component)
- `<f:actionListener>` – creates an instance of the action listener and binds it to its parent component

Commonly Used Tags – JSF Core Library (2/2)

- **`<f:param>` – creates name-value pair for its parent component**
- **`<f:loadBundle>` – loads a specified bundle and stores it in the request scope**
- **`<f:subview>` – acts as a naming container so that the components inside it can be made unique**
  - **Use it to import another JSF page**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Commonly Used Tags – JSF HTML Library (1/2)

- `<h:form>` – used as a container for elements which send data (HTML form)
- `<h:inputText>`, `<h:inputSecret>`, `<h:inputHidden>` – used to input text, passwords and hidden values
- `<h:outputText>`, `<h:outputLabel>` – used to output text
- `<h:commandButton>`, `<h:commandLink>` – renders a button or hyperlink
- `<h:graphicImage>` – renders an image

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Commonly Used Tags – JSF HTML Library (2/2)

- `<h:dataTable>` – creates HTML table
  - `<h:column>` – used as child element of `<h:dataTable>`
- `<h:selectBooleanCheckBox>` – renders as checkbox
- `<h:message>` – renders the first `FacesMessage` assigned to the component defined in the `for` attribute
- `<h:selectManyCheckBox>` - renders as list of checkboxes
  - Elements stored in `<f:selectItem>` / `<f:selectItems>`

HTML UI Components

Live Demo

Managed Beans

Managed Beans

- **Managed beans are JavaBeans which:**
  - **Provide the logic for initializing and controlling JSF components**
    - **Data binding, action listeners, validation, conversion, navigation, etc.**
  - **Manage data across page requests, user sessions, or the application as a whole**
  - **Created by JSF and stored within the `request`, `session` or `application`**
  - **Also called "backing beans"**

Mapping Elements (2)

- **`<managed-property>` – property enclosing element**
- **`<property-name>` – this element's value is the name of a bean property**
- **`<property-class>` – the fully qualified name of the class of the property**
- **`<value>` – value of the property**
  - **If the value is another managed bean then the scope of the other bean should be greater than the scope of the bean**

Binding Values

- **Managed beans and their properties can be used as values for the components**
  - **Example: we have a session scoped managed bean of class `UserBean` with property `userName` we can do**

```
<h:inputText id="userNameInput"
   value="#{userBean.userName}" />
```

  - **JSF will automatically apply component entered value to the `userName` property and vice versa**

Managed Beans – Example

- We have two JavaBeans:

```
@Named @ApplicationScoped
public class ApplicationInfoBean implements
   Serializable {
   private static final long serialVersionUID = 1L;
   private String info;
   // Getters and setters come here
}

@Named @SessionScoped
public class UserBean implements Serializable {
   private String userName;
   private ApplicationInfoBeanapplicationInfoBean;
   // Getters and setters come here
}
```

Managed Beans – Example

- Bind JSF controls with the beans:

```
<h:outputFormat value="Hello, {0}">
   <f:param value="#{userBean.userName}" />
</h:outputFormat>
<h:outputFormat value="Application Info: {0}">
   <f:param value="#{applicationInfoBean.info}" />
</h:outputFormat>
<h:form id="userNameForm">
   <h:outputLabel for="userInput" value="User Name:" />
   <h:inputText id="userInput" value="#{userBean.userName}" />
   <h:commandButton value="Apply" />
</h:form>
<h:form id="appInfoForm">
   <h:outputLabel for="appInfoInput" value="Application Info:" />
   <h:inputText id="appInfoInput"
        value="#{userBean.applicationInfoBean.info}" />
   <h:commandButton value="Apply" />
</h:form>
```

Managed Beans

Live Demo

JSF Navigation Model

What Is Navigation?

- Navigation is a set of rules for choosing the next page to be displayed
  - Applied after a button or hyperlink is clicked
- The selection of the next page is determined by:
  - The page that is currently displayed
  - The action method invoked by the action property of the component that generated the event
  - An outcome string that was returned by the action method or passed from the component

Navigation Elements in `faces-config.xml`

- `<from-view-id>` element defines the source page.
  - May be a pattern. For example `/*`. This will cause all JSF pages to redirect to some view on given outcome.
- `<from-outcome>` element defines the logical outcome as specified in the `action` attribute of the event source
- `<to-view-id>` element defines the page to be displayed when the specified outcome is returned
- `<from-action>` element refers to an action method that returns a `String`, which is the logical outcome

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Navigation Rules – Example

```
<navigation-rule>
  <from-view-id>/*</from-view-id>
  <navigation-case>
    <from-outcome>home</from-outcome>
    <to-view-id>/navigation-demo.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/navigation-demo.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>login_success</from-outcome>
    <to-view-id>/logged.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/navigation-demo.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>login_failed</from-outcome>
    <to-view-id>/login-failed.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Action Attribute in JSF Form

- To specify what to be the form outcome you can
  - Provide a constant string as action attribute of an event source

```
<h:commandButton value="Next Page" action="nextPage" />
```

  - Provide a managed bean method with no parameters which returns `String`
    - Using this approach you can add some logic in this method that returns different result in different situations

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

Navigation Model

Live Demo

Creating JSF Wizard

- Step 1
  - Enter first name, last name
- Step 2
  - Enter phone, email
- Final step
  - Display all entered data
- We will use managed bean `Person` with session scope to store entered data
- We will use navigation rules for the active page

JSF Wizard

Live Demo

JavaServer Faces

Questions?