

In this presentation, we are going to take step by step approach for developing a simple yet completely functioning JSF application called guessNumber. This application comes with JSF reference implementation.



**Sang Shin**

[sang.shin@sun.com](mailto:sang.shin@sun.com)  
[www.javapassion.com](http://www.javapassion.com)  
Java™ Technology Evangelist  
Sun Microsystems, Inc.

2

## Disclaimer & Acknowledgments

- ⌚ Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here is created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- ⌚ Sun Microsystems is not responsible for any inaccuracies in the contents.
- ⌚ Acknowledgments:
  - Many slides and speaker notes are created from [JSF tutorial](#)
  - Source code examples are from sample codes that are shipped with JSF beta

## Revision History


- ? 12/15/2003: version 1: created (Sang Shin)
- ? 12/21/2003: version 2: changed contents based on Beta version released on 12/20/2003 (Sang Shin)
- ? 01/03/2003: speaker notes are added (Sang Shin)
- ? 04/03/2004: updated to reflect V1.0 as part of J2EE 1.4 SDK
- ? Things to do
  - slides need polish



# Guess a Number Sample Application



5

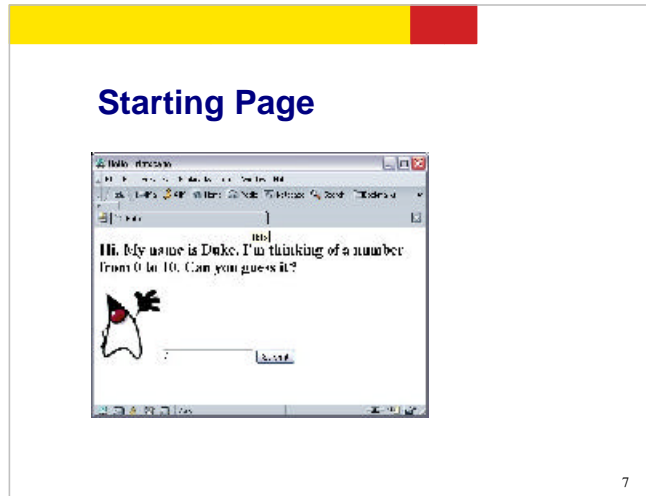


### **Sample JSP Application we are going to build**

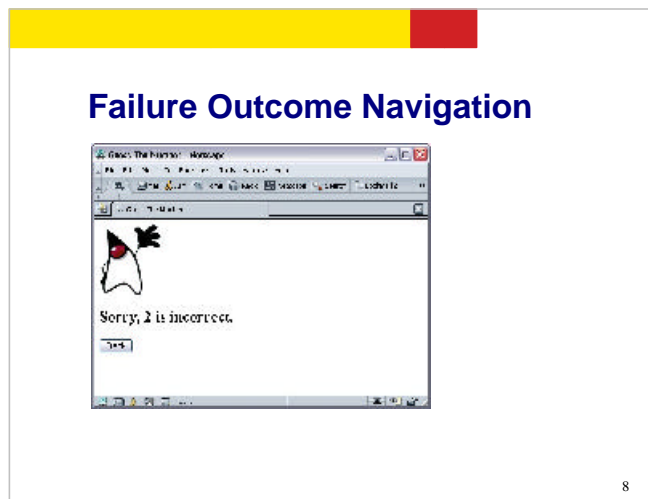
- ? `guessNumber` application that comes with JSF 1.0 as part of J2EE 1.4 SDK
- ? Guess a number between 0 and 10, inclusive
  - The response page tells you if you guessed correctly
- ? Input validation

6

The guessnumber application asks you to guess a number between 0 and 10, inclusive. The second page tells you if you guessed correctly. The example also checks the validity of your input. In other words, if the number entered is not between 1 to 10 inclusive, an error message will be displayed prompting a user to enter a valid number.



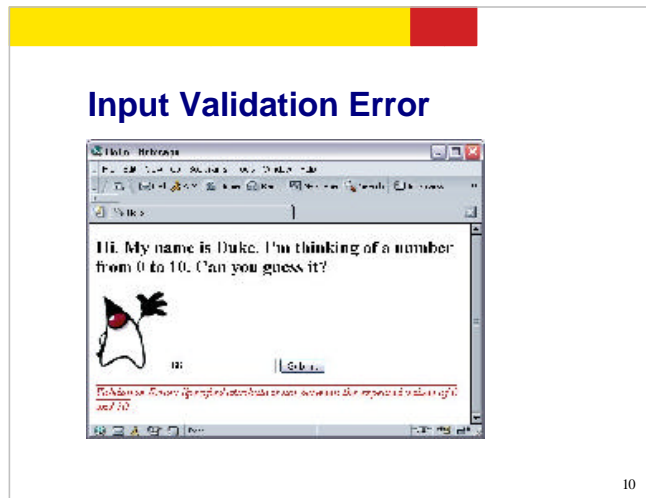
Now let's quickly run the application. I captured the screens from the application. This is the starting page.



When you guess a wrong number, the application displays a message accordingly and then prompts a user to make an another guess.

Please note that this is not an input validation error. This is in a sense a business logic error as opposed to input validation error.





When a user entered a number that is not between 1 to 10 inclusive, that is an input validation error. The input validation occurs before any business logic processing is performed.



**Steps to follow**



11

The slide features a header with a yellow and red bar above a sunset over pyramids. The Sun Microsystems logo is in the top right. The main content area has the title 'Steps to follow' in bold brown text, a small cartoon character below it, and the number '11' in the bottom right corner.



**Steps**

1. Create development directory structure
2. Write web.xml
3. Write ant build script
4. **JSF-specific steps**
5. Build, deploy, and test the application

12

So this is the list of steps. Of course, you don't exactly follow these steps in sequence. In fact, in practice, it is expected that some of these steps will be performed in different order and also in iteration.

Please note that all the steps you will take in developing JSF application is similar to the steps you will take in developing any Web application. So in this presentation, we will spend our time mostly in JS-specific steps.



## Step 1: Create Development Directory Structure



13

The first step is to create development directory structure.

## Development Directory Structure

- ? Same development directory structure for any typical Web application
  - We will use the source/build directory structure of sample Web applications that come with J2EE 1.4 SDK
- ? Ant build script should be written accordingly
  - Just use the build.xml script that comes with J2EE 1.4 SDK

14

The source directory structure is the same directory structure we used for other Web application development under Java WSDP. Of course, the build.xml script should be written accordingly.

**\*.jar files (1) - based in JSF  
(<J2EE1.4\_HOME>/lib)**

- ? jsf-api.jar
  - contains the javax.faces.\* API classes
- ? jsf-impl.jar
  - contains the implementation classes of the JavaServer Faces standard implementation
- ? jstl.jar
  - required to use JSTL tags and referenced by JavaServer Faces standard implementation classes

15

JavaServer Faces applications require several JAR files to run properly. If you are not running the application on the Java WSDP, which already has these JAR files, the WAR file for your JavaServer Faces application must include these jar files.




**\*.jar files (2)**

- ? **standard.jar**
  - required to use JSTL tags and referenced by JavaServer Faces reference implementation classes
- ? **commons-beanutils.jar**
  - utilities for defining and accessing JavaBeans component properties
- ? **commons-digester.jar**
  - for processing XML documents

16

(read the slide)



**\*.jar files (3)**

- ? commons-collections.jar
  - extensions of the Java 2 SDK Collections Framework
- ? commons-logging.jar
  - a general purpose, flexible logging facility to allow developers to instrument their code with logging statements

17

(read the slide)



**Step 2: Write web.xml  
Deployment Descriptor**



18

Step 2 is to write web.xml file.

## web.xml

- ? Same structure as any other Web application
  - `javax.faces.webapp.FacesServlet` is like any other servlet
  - Servlet definition and mapping of FacesServlet
- ? There are several JSF specific `<context-param>` elements

19

Because Struts application is a genuine Web application, it has to follow the same rules that any Web application has to follow. And one of them is the presence of web.xml deployment descriptor file.

The web.xml file should define ActionServlet, which is the controller piece that is provided by the Struts framework, and its mapping with URI.

As you will see in the example web.xml file in the following slide, there are several Struts specific initialization parameters. Also the Struts tag libraries also need to be declared.

## Example: web.xml

```
1 <web-app>
2   <display-name>JavaServer Faces Guess Number Sample Application
3 </display-name>
4   <description>
5     JavaServer Faces Guess Number Sample Application
6 </description>
7
8   <context-param>
9     <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
10    <param-value>client</param-value>
11  </context-param>
12
13  <context-param>
14    <param-name>javax.faces.application.CONFIG_FILES</param-name>
15    <param-value>/WEB-INF/faces-config.xml</param-value>
16  </context-param>
17
18  <context-param>
19    <param-name>com.sun.faces.validateXml</param-name>
20    <param-value>true</param-value>
21  </context-param>
```

20

This is the first part of web.xml file. Please note various <context-param> elements. These context parameters are read by JSF implementation at the time JSF application is loaded and started.

The first context parameter, javax.faces.STATE-SAVING\_METHOD, indicates whether state is being maintained in the client or server side. The second context parameter, javax.faces.application.CONFIG\_FILES, specifies the name and location of application configuration file. The third context parameter, com.sun.faces.validateXml, indicates to the JSF implementation whether it should validate application configuration file or not based on the scheme definition file.

## Example: web.xml

```
1
2 <listener>
3   <listener-class>com.sun.faces.config.ConfigListener</listener-class>
4 </listener>
5
6 <!-- Faces Servlet -->
7 <servlet>
8   <servlet-name>Faces Servlet</servlet-name>
9   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
10  <load-on-startup> 1 </load-on-startup>
11 </servlet>
12
13
14 <!-- Faces Servlet Mapping -->
15 <servlet-mapping>
16   <servlet-name>Faces Servlet</servlet-name>
17   <url-pattern>/guess/*</url-pattern>
18 </servlet-mapping>
19
20 </web-app>
```

21

This is the second part of web.xml file. Please note the servlet definition and mapping of javax.faces.webapp.FacesServlet. In this example, any URL that maps to <root context>/guess/\* goes to the FacesServlet class.




**Step 3: Write Ant Build Script**



22

The slide features a header with a sunset over pyramids and the Sun Microsystems logo. The main content area contains the title 'Step 3: Write Ant Build Script' and the Ant logo. A small page number '22' is located in the bottom right corner of the slide frame.



**Ant build.xml**

- ? Leverage the build.xml file of JSF sample apps or other Web application

23

Just like any other Web application development, you might want to create ant build script. Since JSF sample applications come with ant build scripts, you can leverage them or you can use build scripts that come with other Web applications.



**Step 4: JSF-Specific Development Steps**



24

OK, now take a look at the steps that are unique to JSF applications. As I mentioned before, I am going to spend the rest of this presentation to JSF specific steps.



**“JSF-Specific” Steps for Developing JSF Application**

1. Create the Pages
  - Using the UI component and core tags
2. Define Page Navigation
  - In the application configuration file
3. Develop the backing beans (Model objects)
  - Model objects hold the data
  - Validator, convertor, event handler, navigation logic
4. Add managed bean declarations
  - To the application configuration file

25

Developing a simple JavaServer Faces application requires performing these tasks:

- \* Create the pages using the UI component and core tags
- \* Define page navigation in the application configuration file
- \* Develop the backing beans
- \* Add managed bean declarations to the application configuration file

These tasks can be done simultaneously or in any order. However, the people performing the tasks will need to communicate during the development process. For example, the page author needs to know the names of the objects in order to access them from the page.

The example used in this section is the guessNumber application, located in the <JSF\_HOME>/samples directory. It asks you to guess a number between 0 and 10, inclusive. The second page tells you if you guessed correctly. The example also checks the validity of your input.



# JSF Step1: Creating Pages



26

```
greeting.jsp (1)

<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">
```

27

This is the first part of greeting.jsp page. The greeting.jsp page is one of the two pages (the other one is response.jsp) in guessNumber application. This is the first page that gets displayed when a user enters the guessNumber application.

Please note that taglib declaration. JSF comes with its own set of JSP custom tag libraries and the declaration of the libraries and usage of the tags work in the same way as other custom tag libraries and tags.

Here we declare two JSF tag libraries, html and core. The html JSF tag library contains HTML rendering tags while the core JSF tag library contains JSF core tags.

**greeting.jsp (2)**

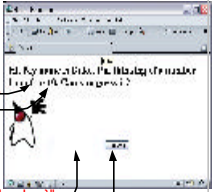
```

<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}"
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"/>

</h:form>
</f:view>
</HTML>

```



28

This is the remaining page of the greeting.jsp. Please note that there are several UI components in this page. And they are `<h:output_text>`, `<h:inputText>`, `<h:commandButton>`.

The `<h:output_text>` elements contains value attributes whose values come from the property values of a JavaBean called `UserNumberBean`. The names of these properties of this bean are called `minimum` and `maximum`. By the way, we will go over these in detail later on in this presentation so don't worry if you don't understand what I am talking about right now.

`<h:graphic_image>` element displays waving duke image. The `<h:inputText>` element displays the `userNumber` property of the `UserNumberBean` and also prompts the user to enter a new number in the textfield. The `<h:commandButton>` displays a button that can be pressed for submitting the form data.

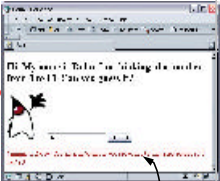
### greeting.jsp (2) in (Input Validation Error)

```

</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}"
  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:view>
</HTML>

```

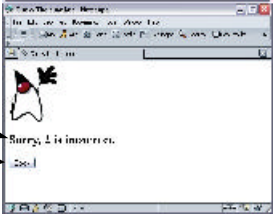


29

This is the same greeting.jsp page. Here the `<h:message>` element is used for displaying any error messages that are generated from input validation errors.

**response.jsp with wrong guess**

```
<HTML>
<HEAD> <title>Guess The Number</title> </HEAD>
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
<body bgcolor="white">
<f:view>
<h:form id="responseForm" >
  <h:graphicImage id="waveImg" url="/wave.med.gif" />
  <h2><f:outputText id="result"
    value="#{UserNumberBean.response}"/></h2>
  <f:commandButton id="back" value="Back" action="success"/><p>
</h:form>
</f:view>
</HTML>
```

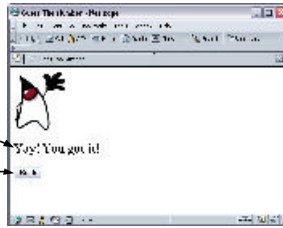


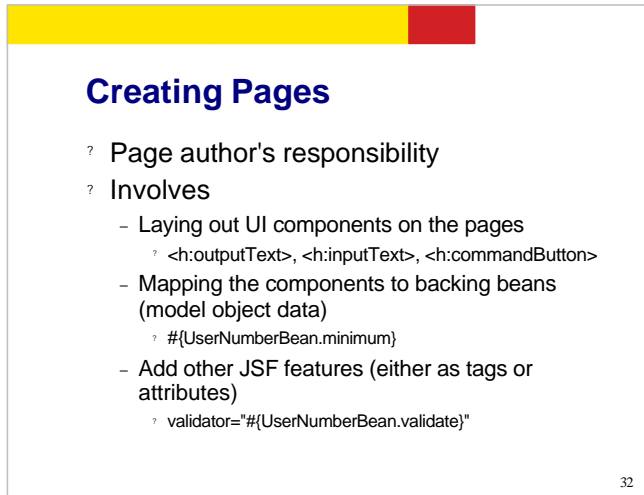
30

This is the 2<sup>nd</sup> and the remaining JSP page called response.jsp. This JSP page is used for displaying a message for both incorrectly guessed number and correctly guessed number. This slide shows a display when a user made a wrong guess while the next slide shows a display when a user made a correct guess.

**response.jsp with correct guess**

```
<HTML>
<HEAD> <title>Guess The Number</title> </HEAD>
<%@ taglib uri="http://java.sun.com/jsp/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
<body bgcolor="white">
<f:view>
<h:form id="responseForm" >
  <h:graphicImage id="waveImg" uri="/wave.med.gif" />
  <h2><h:outputText id="result"
    value="#{UserNumberBean.response}"/></h2>
  <h:commandButton id="back" value="Back" action="success"/><p>
</h:form>
</f:view>
</HTML>
```





## Creating Pages

- ? Page author's responsibility
- ? Involves
  - Laying out UI components on the pages
    - ? <h:outputText>, <h:inputText>, <h:commandButton>
  - Mapping the components to backing beans (model object data)
    - ? #{UserNumberBean.minimum}
  - Add other JSF features (either as tags or attributes)
    - ? validator=#{UserNumberBean.validate}"

32

Creating the pages is the page author's responsibility. This task involves laying out UI components on the pages, mapping the components to model object data, and adding other core tags (such as validator tags) to the component tags.

### greeting.jsp: <h:form ...> tag

- ? Represents an input form, which allows the user to input some data and submit it to the server, usually by clicking a button
- ? **UIInput** and **UIOutput** components are nested inside
  - <h:inputText>
  - <h:commandButton>
  - <h:outputText>

33

The form tag represents an input form, which allows the user to input some data and submit it to the server, usually by clicking a button. The tags representing the components that comprise the form are nested in the form tag. These tags are h:input\_number and h:commandButton.

### greeting.jsp: <h:outputText> tag

```
<h:outputText value="#{UserNumberBean.minimum}"/>  
<h:outputText value="#{UserNumberBean.maximum}"/>
```

? value="#{X.Y}" attribute

- X matches the name defined by the [managed-bean-name](#) element corresponding to the proper managed-bean declaration from the application configuration file
- Y matches name defined by the [property-name](#) element corresponding to the proper [managed-property](#) element

34

The optional id attribute corresponds to the ID of the component object represented by this tag. The id attribute is optional. If you don't include one, the JavaServer Faces implementation will generate one for you.

## Relationship between JSP page and Backing Bean Declaration in App. Conf. File

```

? greeting.jsp
<h:outputText value="#{UserNumberBean.minimum}"/>
<h:outputText value="#{UserNumberBean.maximum}"/>

? Application configuration file (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>java.lang.Long</property-class>
    <value>0</value>
  </managed-property>
  <managed-property>
    <property-name>maximum</property-name>
    <property-class>java.lang.Long</property-class>
    <value>10</value>
  </managed-property>
</managed-bean>

```

35

This slide shows the relationship between JSP page and backing bean declaration in the application configuration file. Every backing bean needs to be declared in the application configuration file. Here in this guessNumber example, there is only one bean called UserNumberBean, which has two properties called minimum and maximum whose types and initial values are also defined.

In the JSP page, you refer to these properties using JSF EL expression notation like `#{UserNumberBean.minimum}` or `#{UserNumberBean.maximum}`.

## Relationship between JSP page and Backing Bean Class

? greeting.jsp

```
<h:outputText value="#{UserNumberBean.minimum}"/>
<h:outputText value="#{UserNumberBean.maximum}"/>
```

? UserNumberBean.java

```
public class UserNumberBean {
    private int maximum = 0;
    public int getMaximum() {
        return (this.maximum);
    }
    public void setMaximum(int maximum) {
        this.maximum = maximum; ...
    }

    private int minimum = 0;
    public int getMinimum() {
        return (this.minimum);
    }
    public void setMinimum(int minimum) {
        this.minimum = minimum; ...
    }
}
```

36

This slide shows the relation between the JSP page and the Bean code. Just like any other Java Bean, the UserNumberBean contains getter and setter methods of its properties as shown in this slide.

## greeting.jsp: <h:inputText> tag

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

- ? value="#{UserNumberBean.userNumber}"
  - The **value** attribute refers to the `userNumber` property on the `UserNumberBean` bean
  - After the user submits the form, the value of the `userNumber` property in `UserNumberBean` will be set to the text entered in the field corresponding to this tag

37

The optional `id` attribute corresponds to the ID of the component object represented by this tag. The `id` attribute is optional. If you don't include one, the JavaServer Faces implementation will generate one for you.

## Relationship between JSP page and Backing Bean Class

? greeting.jsp

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

? UserNumberBean.java

```
public class UserNumberBean {
    Integer userNumber = null;
    public void setUserNumber(Integer user_number) {
        userNumber = user_number;
        System.out.println("Set userNumber " + userNumber);
    }

    public Integer getUserNumber() {
        System.out.println("get userNumber " + userNumber);
        return userNumber;
    }
    ...
}
```

38

This slide shows the relationship between JSP page and backing bean. As in the case of `<h:output_text>`, the value attribute of `<h:inputText>` can also refer to a property of a Bean, in this case, `userNumber` property. The bean code has getter and setter method of this property.

## greeting.jsp: <h:inputText> tag

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

- ? validator="#{UserNumberBean.validate}"
  - Is a JSF EL expression pointing to a backing-bean method that performs validation on the component's data.

39

Now let's talk about how input form validation can be performed. For <h:inputText>, you can also specify a validator method using validator attribute. Here the UserNumberBean, in addition to getter and setter methods of its properties, also contains a method that perform the input validation. And the name of the method happened to be called "validate" in this example code.

### Relationship between JSP page and Backing Bean Class (page 1)

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

? UserNumberBean.java

public class UserNumberBean {
  ...
  public void validate(FacesContext context, UIInput component) {
    if ((context == null) || (component == null)) {
      throw new NullPointerException();
    }
    Object value = component.getValue();
    if (value != null) {
      try {
        int converted = intValue(value);
        if (maximumSet &&
            (converted > maximum)) {
          // Queue error message to FacesContext object
        }
      }
    }
  }
  ...
}
```

40

So this slide shows the validate() method of the UserNumberBean bean. The input validation logic is basically making sure the entered number is in between 1 to 10 inclusive.

If the number that is entered is not between 1 to 10, this method creates an error message and then queues it to the FacesContext object.

### greeting.jsp: <h:commandButton ...> tag

```
<h:commandButton id="submit" action="success"  
value="Submit" />
```

- ? Represents the button used to submit the data entered in the text field
- ? **action** attribute specifies an outcome that helps the navigation mechanism to decide which page to display next

41

The commandButton tag represents the button used to submit the data entered in the text field. The action attribute specifies an output that helps the navigation mechanism to decide which page to open next.

### greeting.jsp: <h:message ...> tag

```
<h:messages style="color: red; font-family: 'New Century  
Schoolbook', serif; font-style: oblique; text-decoration:  
overline" id="errors1" for="userNo"/>
```

- ? **for** attribute refers to the ID of the component that generated the error messages
- ? **<h:messages>** tag will display the error messages wherever the messages tag appears in the page.
- ? **style** attribute allows you to specify the style of the text of the message. In the example, the text will be red, New Century Schoolbook, serif font family, oblique style, and a line will appear over the text

42

The `output_errors` tag will display an error message if the data entered in the field does not comply with the rules specified by the validator. The error message displays wherever you place the `output_errors` tag on the page. The `for` attribute refers to the component whose value failed validation.

## greeting.jsp (2)



```
<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}"
    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"/>


  </h:form>
</f:view>
```

43

This is the same greeting.jsp page we've already seen. Here I am showing it again to show the relationship between the “for” attribute of <h:message> element and “id” attribute of the <h:inputText>.



## JSF Step 2: Define Page Navigation



44

## 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

45

Another responsibility that the application developer has is to define page navigation for the application, which involves determining which page to go to after the user clicks a button or a hyperlink. The JavaServer Faces navigation model, new for this release, is explained in Navigation Model. Navigating Between Pages (page 890) explains how to define the navigation rules for an entire application.

The application developer defines the navigation for the application in the application configuration file, the same file in which managed beans are declared.

## Navigation Rule 1 for guessNumber Example

```
<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>
```

46

Each navigation-rule defines how to get from one page (specified in the from-tree-id element) to the other pages of the application. The navigation-rule elements can contain any number of navigation-case elements, each of which defines the page to open next (defined by to-tree-id) based on a logical outcome (defined by from-outcome).

## Navigation Rule 2 for guessNumber Example

```
<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>
```

47


Each navigation-rule defines how to get from one page (specified in the from-tree-id element) to the other pages of the application. The navigation-rule elements can contain any number of navigation-case elements, each of which defines the page to open next (defined by to-tree-id) based on a logical outcome (defined by from-outcome).



**JSF Step 3:  
Developing Backing  
Beans (Model objects)**



48

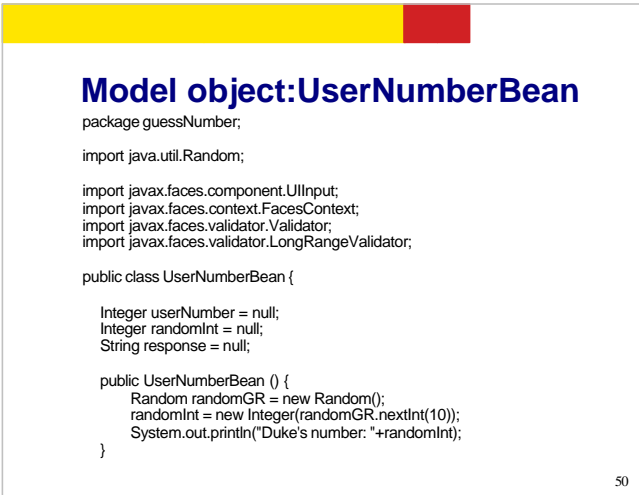


## Developing Model Object

- ? Model object holds data
  - It is a JavaBean
- ? Responsibility of application developer
  - Page author and the application developer need to work in tandem to make sure that the **UI component tags refer to the proper object properties**

49

Developing model objects is the responsibility of the application developer. The page author and the application developer might need to work in tandem to make sure that the component tags refer to the proper object properties, that the object properties have the proper types, and take care of other such details.



```
Model object:UserNumberBean
package guessNumber;

import java.util.Random;

import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.LongRangeValidator;

public class UserNumberBean {

    Integer userNumber = null;
    Integer randomInt = null;
    String response = null;

    public UserNumberBean () {
        Random randomGR = new Random();
        randomInt = new Integer(randomGR.nextInt(10));
        System.out.println("Duke's number: "+randomInt);
    }
}
```

50

Here is the UserNumberBean class that holds the data entered in the text field on greeting.jsp.

As you can see, this bean is just like any other JavaBeans component: It has a set of accessor methods and a private data field for each property. This means that you can reference beans you've already written from your JavaServer Faces pages.

## Properties of Model Object

- ? Property can be any of the basic primitive and reference types
  - Number types, String, int, double, and float
- ? JSF implementation automatically convert the data to the type specified by the model object property
- ? You can also apply a **converter** to a component to convert the components value to a type not supported by the component.

51

As you can see, this bean you saw in the previous slide is just like any other JavaBeans component: It has a set of accessor methods and a private data field for each property. This means that you can reference beans you've already written from your JavaServer Faces pages.

A model object property can be any of the basic primitive and reference types, depending on what kind of component it references. This includes any of the Number types, String, int, double, and float. JavaServer Faces technology will automatically convert the data to the type specified by the model object property.

You can also apply a converter to a component to convert the components value to a type not supported by the component.

In the `UserNumberBean`, the `userNumber` property has a type of `Integer`. The JavaServer Faces implementation can convert the `String` request parameters containing this value into an `Integer` before updating the model object property when you use an `input_number` tag. Although this example converts to an `Integer` type, in general, you should use the native types rather than the wrapper classes.



**JSF Step 4:  
Adding Managed Bean  
Declarations**



52

The slide features a header with a sunset background and the Sun Microsystems logo. The main content area contains the title 'JSF Step 4: Adding Managed Bean Declarations' and the JSF logo. The page number '52' is located in the bottom right corner of the slide.

### Adding Managed Bean Declarations to App. Conf. File

- ? JSF implementation processes Application Configuration File on application startup time and initializes the `UserNumberBean` and stores it in session scope if no instance exists
- ? Bean is then available for all pages in the application
- ? No need for `<jsp:useBean>` tag

53

After developing the beans to be used in the application, you need to add declarations for them in the application configuration file. The task of adding managed bean declarations to the application configuration file can be done by any member of the development team.

The JavaServer Faces implementation processes this file on application startup time and initializes the `UserNumberBean` and stores it in session scope if no instance exists. The bean is then available for all pages in the application. For those familiar with previous releases, this managed bean facility replaces usage of the `jsp:useBean` tag.

## 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>java.lang.Long</property-class>
    <value>0</value>
  </managed-property>
  <managed-property>
    <property-name>maximum</property-name>
    <property-class>java.lang.Long</property-class>
    <value>10</value>
  </managed-property>
</managed-bean>
```

54

We have seen UserNumberBean declaration in the application configuration file already. I included it here again.



The top section of the slide features a horizontal banner. On the left, a sunset scene with a sun low on the horizon behind several pyramids. On the right, the Sun Microsystems logo, which consists of a stylized 'S' made of four interlocking squares, followed by the word 'Sun' in a serif font and 'microsystems' in a smaller sans-serif font below it. The banner has a yellow and red gradient at the top.

**Live your life  
with Passion!**



A small, stylized graphic of a person sitting at a desk with a laptop, positioned below the main text.

55