8.1 Using JavaServer Faces Technology in Web Pages

JSF technology provides component tags which can be used for creating the Web pages of an application. Functionality to these component tags is added by registering converters, validators, and listeners with them. Managed beans are also associated with the components in the Web page. The managed beans provide business logic to Web pages.

8.1.1 Web Page Setup Using JSF

A typical JSF Web page comprises the following components:

- Namespace declarations which are used to declare JSF tag libraries.
- HTML head and body tags. However, these are optional.
- Form tag which represents the user input components.

There are various tag libraries defined to create JSF pages. The developer has to declare the tag library which will be used in the definition of the current Web page. Every JSF page requires access to two tag libraries – JavaServer Faces HTML render kit and JavaServer Faces Core tag library. JSF also has a library which defines the tags for HTML user interface; this tag library is known as HTML standard tag library. The HTML standard tag library is associated with an HTML render kit. In order to use certain tag library in the JSF Web page, it has to be declared in the namespace section.

When multiple tag libraries are included for certain Web page, then a tag library specific prefix has to be added to the tag. For instance, for all the HTML tags a letter 'h' is prefixed to the tag.

8.1.2 HTML Tag Libraries in JSF Pages

The HTML tag library in JSF is used to add HTML components to the Web page. Similar to an HTML page these components are used to display data on the Web page, accept user data as input, and so on. Based on the client who is invoking the Web page, each component of the JSF can be rendered in a different way.

Table 8.1 shows some of the important HTML tags used in JSF pages.

Tag	Description
h:form	Represents an input form to which other components can be added
h:datatable	Creates an HTML table which can be modified dynamically
h:graphicImage	Displays an image
h:column	Represents a column of data in a data table of the database
h:commandLink	Used to create a hyperlink
h:inputText	Used as text area which can accept user input
h:message	Used to display a localized message

Tag	Description
h.outputText	Allows display of plain text
h:selectOneListBox	Allows selecting one element from a list of options
h:selectOneMenu	Allows selecting one item from a menu
h.selectRadioOne	Allows choosing one option from a list of options

To add an HTML page structure to the JSF Web pages, <h:head > and <h:body> tags need to be added to the JSF page.

8.1.3 Component Tag Attributes

Each component tag has certain attributes associated with it. Following are the common attributes associated with each tag:

- **binding** – This property binds the component with a bean, which will be invoked for any action with respect to the component.

- **id** – Uniquely identifies the component. This attribute is not usually required for the component but is used when the component has to be referred by another component. The id need not be explicitly defined by the developer as the JSF implementation automatically generates an id for the components.

- **immediate** – This attribute can only assume a value of true, when the value is set to true. It indicates that any validations and conversion with respect to the component should happen immediately.

  Consider an instance where the Web page has two components – a text area and a button.

  If the immediate attribute for both the components is set to true, then the value entered in the text area is used immediately when the button is clicked.

  If the immediate attribute of the text area is not marked true and the immediate attribute of button is marked true, then the value is not considered as a parameter when the button is clicked. The value in the text area is considered for interpretation later.

- **rendered** - This attribute specifies the condition when the component should be rendered. If this condition occurs or holds true, then the component is rendered otherwise the component is not rendered.

  A boolean EL expression is used along with the rendered attribute to determine whether the component should be rendered or not. For instance, in a bank portal every account holder has a login. When the user logs in, the details of the accounts held by the user are displayed. The Web page displaying the account details also has an area where the fixed deposits held by a user need to be displayed. The elements displaying the fixed deposits are rendered only if the account holder actually holds a fixed deposit otherwise, these elements are not rendered.

- **style** – specifies a CSS style for the component.
- **styleClass** – this attribute specifies a CSS class according to which the component has to be rendered.
- **value** – This specifies the value component in the form of a value expression.

Consider the following example:

```
<h:commandLink id="check"
...
rendered="#{cart.numberOfItems > 0}">
<h:outputText
value="#{cart.numberOfItems}"/>
</h:commandLink>
```

The value field in the code is the value to be displayed with the outputText tag which implies the number of items placed in the shopping cart in an online shopping portal. The rendered attribute ensures that the component is rendered only if the value of numberOfItems is greater than zero.

8.1.4 Using Text Component Tags

Text components of a JSF page are essential to view and display text on the Web page. There are three types of text components – Label, Text area, and Password.

- Label is used to display read only text on the Web page.
- Field allows the user to enter text data, which can be submitted along with the user form.
- Password field also allows the user to enter data, but the data entered in the field is concealed through characters such as asterisks.

There are four types of input tags – h:inputhidden, h:inputSecret, h:inputText, and h:inputTextArea.

- h:inputHidden tag allows accepting a hidden variable in the Web page
- h:inputSecret tag is used for text areas which accept password or any confidential values
- h:inputText tag is used to accept text input in a single line
- h:inputTextArea is used to accept input of multiple lines

Output tags are used to include the output components in the JSF Web page. There are four types of output components – h:outputFormat, h:outputLabel, h:outputLink, and h:outputText.

- h:outputFormat is used to display formatted output
- h:outputLabel is used to display a read-only label
- h.outputLink is used to display a hyperlink
- h.outputText is used to display a one-line text string

8.1.5 Using Command Component Tags

The command component tags are used to perform actions and navigation. h:commandButton tag is used to add a button component to the JSF page and h:commandLink tag adds a hyperlink to the JSF page.

Apart from the attributes such as binding, id, value, and so on which are added as attributes to the command component tags, an action and actionListener can also be added to the command component tags.

The action attribute can present a logical outcome or a reference to a bean which will result in a logical outcome. This outcome will determine the page to be accessed when the command component is activated.

The actionListener attribute refers to the bean method or listener which will process the component action.

8.1.6 Using Data Bound Table Components

The h:datatable component is used to add relational data to the Web page. This component enables binding to a collection of data objects.

The value attribute of the h:datatable tag is used to bind relational data to the component. The data object which can be added to the data table component can be a list of beans, array of beans, a single bean, javax.faces.model.DataModel object, java.sql.Resultset object, javax.servlet.jsp.jstl.sql.Result object, or javax.sql.Rowset object.

8.1.7 Using Core Tags

Apart from the tags implementing the HTML functionality, JSF has a set of core tags used to perform core actions. Following are the categories of core tags supported by JSF:

- **Event handling core tags** – actionListener, phaseListener, setPropertyActionListener, valueChangeListener are the event listening core tags
- **Data Conversion core tags** – converter, converterDateTime, convertNumber are data conversion core tags

- **Facet Core tags** - Facet represents a named section within a container, facet is created through facet tag and the data pertaining to the facet is represented through the metadata tag
- **Core tags used to represent items in a List** – f:selectItem and f:selectItems tags are used to represent items in a list
- **Validator core tags** – These tags are used to validate input values or other values in JSF pages
- Miscellaneous core tags

  All these core tags are inserted in the page definition with a prefix 'f'.

8.2 Using Converters, Listeners, and Validators

Converters, Listeners, and Validators are also components of JSF pages which process the input data of the form. Converters are used to convert the input data of the components. Listeners capture the events occurring in the page and perform actions according to the events. Validators are used to validate the data received from the input components.

8.2.1 Using Standard Converters

The JSF standard implementation provides a set of Converter implementations that can be used to convert component data. As discussed in earlier sections, the application data is represented in two views – a presentation view and a model view.

The presentation view uses the data format which can be used in the Web pages and the model view uses data format which can be operated by the application beans. The data has to be converted between these two formats while executing the application. The standard converter implementations present in javax.faces.Convert can be used to perform required conversions in the application. Each converter is associated with an error message. If a converter is unable to convert the input value of the component, then it returns the error message. Following are the converter implementations present in javax.faces.Convert package.

- BigDecimalConverter
- BigIntegerConverter
- BooleanConverter
- ByteConverter
- CharacterConverter
- DateTimeConverter
- DoubleConverter
- EnumConverter

- FloatConverter
- IntegerConverter
- LongConverter
- NumberConverter
- ShortConverter

In order to use a converter with a component, the converter has to be registered with the component first. Following are the ways in which the converter can be registered with a component:

- The converter tag is nested within the component tag. This is generally used for date and time converters.
- Bind the value attribute of the component with the managed bean property which in turn has the converter.
- Add a converter attribute to the component tag and refer to the converter from the converter attribute.
- A converter tag can be nested in a component tag. Then, one can use either a converter tag's converterId attribute or its binding attribute to reference the converter.

- **Using DateTimeConverter**

  The DateTimeConverter is used to convert the input text of a component into java.util.Date type. This conversion is possible by including a convertDateTime tag in the component tag as shown in the following code:

```
<h:outputText value="#{ShoppingBean.shipDate}">
<f:convertDateTime type="date" dateStyle="full" />
</h:outputText>
```

  The value attribute is set to java.util.Date type and the date format is converted according to the tag definition.

  The tag will display the date in the following format:

```
Saturday, September 21, 2013
```

  The DateTimeConverter can have the following attributes:

  - **binding** – It is used to bind the converter to a bean.
  - **dateStyle** – This attribute defines the date format according to java.text.DateFormat.
  - **for** – This attribute is used in case of composite components, in this case to bind the dateTimeConverter to certain component in the group.

- **locale** – This attribute is used to represent a date format which is part of predefined set of date styles.
- **pattern** – This attribute determines the formatting pattern of the date and time.
- **timeStyle** – This attribute is also used to define the format of the date and time.
- **timeZone** – This attribute of the DateTimeConverter defines the time zone in which the application is running.
- **type** – This attribute defines whether a string value has date, time, or both date and time.

➔ **Using NumberConverter**

Numberconverter tag is used to transform the data from the input component to java.lang. Number type object. Similar to DateTimeConverter, NumberConverter also has several attributes which define the parameters of the conversion. Few of these attributes are binding, currencyCode, currencySymbol, for, groupingUsed, and so on.

8.2.2 Registering Listeners on Components

Listener objects are used to capture events in the Web page. These listeners can be implemented as classes or as managed bean methods. The listener method is referenced through actionListener or valueChangeListener if it is a managed bean method. If it is implemented as a class, then the listener is referenced through the tag actionListener or through tag valueChangeListener.

➔ **Registering a ValueChangeListener on a Component**

The ValueChangeListener can be registered on a component which implements EditableValueHolder, by nesting a f:valueChangeListener tag within the component's tag on the JSF page.

The ValueChangeListener has two attributes namely, type and binding. The type attribute specifies the implementation of ValueChangeListener along with the tag, whereas the binding attribute binds the component to a managed bean.

In Code Snippet 1, an example is shown on the usage of the type attribute

Code Snippet 1:

```
<h:inputText id="name" size="30" value="#{ShoppingBean.name}"
required="true">
<f:valueChangeListener
type="Dealsstore.listeners.UserNameChanged" />
</h:inputText>
```

In the given code, the listener on the inputText component is a valueChangeListener. The Listener object is a user defined listener which keeps track of the changed name value in the inputText field and UserNameChanged event in the given hierarchy.

➔ **Registering an ActionListener on a Component**

ActionListeners can be registered on command components such as buttons. The ActionListener can be registered by including a tag in the command component tag as in case of ValueChangeListener. The actionListener also has type and binding attributes which are used to reference the user defined listener implementations.

The core tag library also has an setActionListener tag which can be used to register an actionListener with an ActionSource instance.

8.2.3 Using Standard Validators

JSF provides a standard set of validators that can be used by the developers to validate component's data.

Following is the list of the standard validators present in JSF:

➔ **BeanValidator class** – It is a class used to register a bean validator for the component; the corresponding tag is validateBean.

➔ **DoubleRangeValidator class** – It is a class used to validate whether a certain float value of a component is within the specified range or not. The tag used for this validator is validateDoubleRange.

➔ **LengthValidator class** – This class is used to validate string inputs from the components. The validation is based on whether the string length is within the expected limit or not. The tag for the validator is LengthValidator.

➔ **LongRangeValidator** – This class is used to define validators which will check the range of floating type value. The tag used for this validator is validateLongRange.

➔ **RegexValidator** – This class is used to define a validator of an input component against a regular expression from the java.util.regex package. The tag used for this validator is validateRegEx.

➔ **RequiredValidator** – This class is used to define a validator which checks that the current value is not empty. This is applied on a component which accepts input. The tag for this validator is validateRequired.

All the standard validator classes implement the Validator interface. User-defined validators can also be written in an application implementing the Validator interface. These validators can be associated with standard error messages which can be displayed if the validation fails on the component data.

The validations of the input data can also happen through bean validation, where the developer can specify bean validation constraints on the managed bean properties. Upon specifying the constraints on the managed bean properties, the constraints are automatically specified on the corresponding user interface fields.

➔ **Validating a Component's Value**

Similar to converters, the validators must be registered on the components where validation is required. One can register the validators with the components in one of the following ways:

- Nest the validator tag within the corresponding component tag.
- Refer to a method which performs the validation as an attribute to the component tag.
- Nest the validator tag within the component tag and use the validatorId or binding attribute to refer to the validator.

Validation can be performed only on those components which implement EditableValueHolder.

Following is an example of using a validator tag:

```
<h:inputText id="check" size="4" value="#{cartItems.quantity}">
<f:validateLongRange minimum="1"/>
</h:inputText>
```

In the example, the validator tag is nested within the inputText tag on which the validation is to be performed. This tag is responsible for validating the value entered in the textbox and imposes a condition that the minimum value should be 1. The attributes of all the validator tags accept Expression Language (EL) expressions.

8.2.4 Referencing a Managed Bean Method

Components can be associated with managed beans through attributes. Following are attributes which enable referencing managed bean methods that can perform certain functions on the component:

➔ **action** – The action attribute refers to a managed bean. This attribute is used on command components or on components which have certain event listeners registered.

➔ **actionListener** – This attribute refers to a managed bean which can respond to event actions.

➔ **validator** – This attribute refers to a managed bean method which can validate the component value.

➔ **valueChangeListener** – This attribute refers to a managed bean method, which can handle the change of values in various components.

Components implementing ActionSource interface can use the action and actionListener attributes and components implementing EditableValueHolder interface can use the validator and valueChangeListener attributes.

The managed bean has to be defined to handle all the required validations, which can later be used in attributes of the components.

8.3 Developing Applications with JSF Technology

Managed beans are bean classes which add functionality to the JSF page. They comprise getter and setter methods for the properties defined in the bean and also business logic relevant to the application.

8.3.1 Creating a Managed Bean

Managed bean in Netbeans can be created by right-clicking the JSF project and adding the managed bean element to the project. If the developer is defining the class manually, then it should be a class defined with default constructor, a set of properties associated with the editable text area, and a set of accessor and mutator methods for each component. The managed bean properties can be bound to any one of the following:

- Component value
- Component instance
- Converted value
- Listener instance
- Validator instance

A managed bean component can validate component's data, handle an event fired by a component, and define page navigation.

- **Using Expression Language to Reference Managed Beans**

  Expression language can be used with the component tags to reference managed beans. Expression language offers features such as deferred evaluation of expressions, use value expression to both read and write data, and method expressions.

  There are various phases in the application lifecycle of a JSF application. It will be optimal to defer the evaluation of an expression until the validation and conversion phases are complete. Code Snippet 2 demonstrates how the values entered in the Web page are associated with the beans.

  Code Snippet 2:

```
<h:inputText id="payamount" size="30" value="#{ShoppingBean.price}"
required="true">
converter = "#shoppingBean.IntegerConverter"
validator="#{shoppingBean.validateNumberRange}"
</h:inputText>
```

  In Code Snippet 2, the input text area is an editable place holder which expects numerical data. First, it converts the input text to integer and then, checks the value of the integer. According to the application domain, since the entered value is a currency, it should always be positive. This can be ensured by defining the minimum attribute of the validator.

All JavaServer Pages attributes accept EL expressions. Apart from referencing bean properties, these attribute expressions can also refer to implicit objects, lists, arrays, and maps.

8.3.2 Writing Bean Properties

A component tag value can be bound with the managed bean property through the value attribute and the component instance can be bound to the managed bean through binding attribute.

The 'binding' attribute of all the converters, listeners, and validators can be used to bind them to the managed bean. To successfully bind the converters, listeners, and validators to a component, the managed bean property must accept and return same type of converter, listener, and validator object. Similarly, to bind the component values with the managed bean properties the corresponding types of the data must match.

➔ **Writing Properties Bound to Component Values**

When the developer intends to write the component value as a bean property, then the data type of the bean property and the component value must be same.

Following are the component classes supported by javax.faces.components and acceptable types of their values.

- The components of class UIInput, UIOutput, UISelectItem, and UISelectOne can have data of any primitive Java data type for which a converter is defined in the javax.faces.convert. Converter class.

  In a JSF page they are represented through the tags <h:input> and <h:output>, and the values are bound through the value attribute.

- The component class UIData reads a group of data items from the component, it can have data of types, array of beans, List of beans, single bean, java.sql.ResultSet, javax.servlet.jsp.jstl.sql.Result, and javax.sql.RowSet.

  In JSF the UIData component is represented through the tag <h:dataTable>.

- The components of class UISelectBoolean can have variables of type class Boolean or the primitive type boolean.

  In a JSF page, the UISelectBoolean component is represented through the tag <h:selectBooleanCheckbox>.

- The components of class UISelectItem can be bound to properties of type java.lang.String, Collection, Array, and Map.

- The components of class UISelectMany can be bound to properties of type array or List.

  The corresponding JSF tag for the UISelectMany component is <h:selectMany>.

➔ **Writing Properties Bound to Component Instances**

When a property is bound to component instance, it will return a component instance instead of values. For instance, in a travel portal, the payment options on the Web page may not be rendered initially when the page is loaded. When a button is clicked, a component event listener is associated with the managed bean. The event listener component invokes the managed bean, which may in turn render the payment options component, which is an aggregation of other components. Code Snippet 3 shows the example of binding with a component instance.

Code Snippet 3:

```
<h:selectBooleanCheckbox id="payment"
binding="#{payBean.paymentGateway}" />
<h:outputLabel for="payment" rendered="false"
binding="#{payBean.cardDetails}" />
</h:outputLabel>
```

In Code Snippet 3, a checkbox is rendered which is connected to the paymentGateway object in the managed bean. The card details are initially not rendered but they are rendered after the checkbox instance representing the paymentGateway object is selected. Once the component is rendered after checking the checkbox, the card details label is rendered. The for attribute of the outputLabel tag is bound with the checkbox control using the id attribute of the checkbox.

➔ **Binding Properties to Converters, Listeners, and Validators**

The converter, listener, and validator implementations are bound to managed bean attributes. This binding enables the managed bean to manipulate their implementations and add additional functionality to them.

8.3.3 Writing Managed Bean Methods

Similar to any other class, methods can be defined in managed beans also. These methods perform several application specific functions such as processes associated with the page navigation, handling action events, performing validation on the component values, and handling value change events.

It is a good programming practice to implement all the validator, listener, and converter methods in the same managed bean to which the attribute values of the JSF page are bound. This enables safe access to the attributes of the managed bean by the methods of the bean.

➔ **Writing a Method to Handle Navigation**

An action method is bound to the action attribute of the component tag. This method should be a public method which accepts no parameters and returns an object which is a logical outcome of the action attribute determining the page that is to be displayed by the navigation system.

The action method typically returns a string outcome. An Enum class can be defined to encapsulate all the possible outcome strings.

➔ **Writing a Method to Handle an Action Event**

A method that handles an action event should be public in the managed bean. This method will accept the action event as a parameter and return void. The action method handling an action event is referenced by the component actionListener attribute. The action method handling an action event can be implemented only by components that implement javax.faces.component.ActionSource.

➔ **Writing a Method to Perform Validation**

A method can be included in the managed bean to validate input instead of implementing the javax.faces.validator.Validator. A managed bean method which performs the validation must accept the javax.faces.context.FacesContext, this object represents the component whose data must be validated.

The validator method of the managed bean is referred through the validate attribute of the component. Only the components of the class UIInput can be validated.

➔ **Writing a Method to Handle a Value Change Event**

Managed beans can handle the value change event in the components through a method which accepts a value change event and returns void. The method handling a value change should be public. The method is referenced from a valueChangeListener attribute of the component.

8.4 Configuring JSF Applications

For large applications, configuration of the application is essential to ensure appropriate functioning of the application. Following are the tasks required to configure the applications:

➔ Registering the managed beans with the application. This ensures that all the classes and components of the application can access the managed beans.

➔ Configuration of the managed beans so that the beans are instantiated with appropriate values whenever referenced.

➔ Defining the navigation rules among the pages of the application.

➔ Packaging the application to include all the resource files in the application.

JSF provides a portable configuration document which is an XML file to configure the resources of the application. Each application may have more than one application resource configuration files. The faces-config.xml file is one such configuration file used in JSF applications.

Each configuration file must include the following information in the given order:

➔ XML version number with an encoding attribute.

➔ A faces-config tag with all the declarations of the tag libraries and name spaces used in the application.

➔ The application can have more than one configuration file which can be located in any one of the following ways:

- Locating a resource /META-INF/faces-config.xml in the application's /WEB-INF/lib/ directory.
- A context initialization parameter javax.faces.application.CONFIG_FILES in the Web deployment descriptor specifies the path to the multiple configuration files of the Web application.
- A resource named faces-config.xml is the configuration file for simple Web applications.

An instance of javax.faces.application.Application class is created for all the Web applications in JSF. This object helps in accessing all the resources pertaining to the application.

As the configuration information of the application is stored in multiple configuration files, it is important to define an order in which these files are accessed. This order of access is defined by an ordering XML element. The ordering of the configuration files can be absolute ordering or relative ordering.

If the ordering is an absolute ordering then the files listed are processed in the order specified. A relative ordering element has sub elements before and after. The configuration files are specified with these sub elements to determine the order accessing the configuration files.

8.5 Using Validators and Converters in a JSF Application

To use validators and converters in a JSF page, you can apply the code given in Code Snippet 4. In Code Snippet 4, two validators have been defined each on the Product textbox and Cost textbox. The product name cannot be less than 4 characters and the cost of the product cannot be less than 200.

A standard converter is also used to convert the cost of the product into a number with precision of two digits after the decimal point.

Code Snippet 4:

```
<?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">
    <h:head>
        <title>Purchase form</title>
    </h:head>
    <h:body>
        <h:form>
            <f:view>
        <h:outputLabel for="Name" value="ProductName:"/>
        <h:inputText id="Name" label="ProductName" required="true" >
            <f:validateLength minimum="4"></f:validateLength>
        </h:inputText>
```

Concepts

```
        <h:message for="ProductName" />
        <p></p>
         <h:outputLabel for="cost" value="Cost:"/>
         <h:inputText id="cost" label="Cost" size="2" >
             <f:validateLongRange minimum="200" />
              <f:convertNumber maxFractionDigits ="2" />
          </h:inputText>
         <p></p>
         <h:commandButton id="register" value="Register"
                action="confirm" />
        </f:view>
        </h:form>
  </h:body>
</html>
```

Figure 8.1 shows the output page when the validators on the page come into action and find the data in the page elements is not according to what is prescribed by the validator.

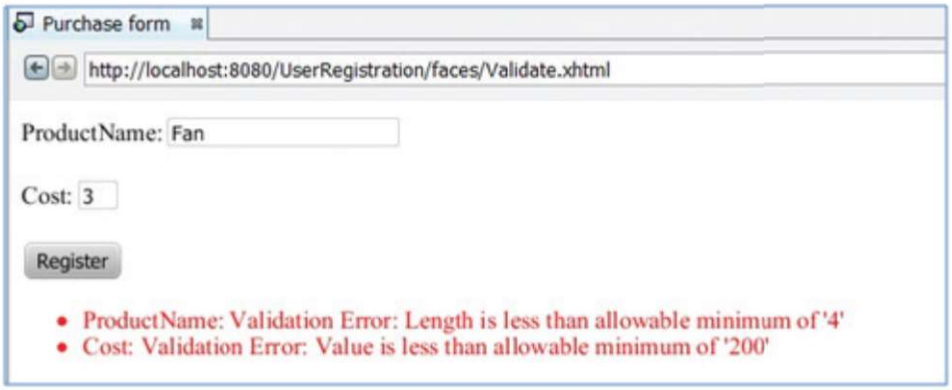

Figure 8.1: Validators in JSF Page

➔ **Converter Implementation through Managed Beans**

Here the application accepts the value in rupees and converts the currency to dollars.

Code Snippet 5 shows the code for the index page where the application execution begins.

Code Snippet 5:

```
<?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"

      <h:head>
       </h:head>
```

Concepts

```
    <h:body>
        <h:form>
            <h3> Currency Converter </h3>
            <p></p>
   <h:outputLabel for="rupees" value="Amount in Rupees:"/>
   <h:inputText id="rupees" value="#{currencyBean.rupees}"/>
            <h:commandButton  id="submit" value="Submit" action =
"response"/>

        </h:form>
    </h:body>
</html>
```

Code Snippet 6 shows the managed bean associated with the JSF page.

Code Snippet 6:

```
package converter;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean (name = "currencyBean")
@SessionScoped
public class CurrencyBean {

int rupees;
int dollars;
   public CurrencyBean() {
    }

    public int getRupees() {
        return rupees;
    }
    public void setRupees(int rupees) {
        this.rupees = rupees;
    }
    public int getDollars() {
        this.dollars = ((this.rupees)*60);
        return dollars;
    }
    public void setDollars(int dollars) {
        this.dollars = dollars;
    }
}
```

In Code Snippet 6, two instance variables rupees and dollars are given whose values are encapsulated through getter and setter methods. These values are used in the Web pages.

Code Snippet 7 shows the code for the response page.

Code Snippet 7:

```
<?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">
    <h:head>
    </h:head>
    <h:body>
         <h3> Currency Converter </h3>
         <h:outputText id="output"  value="The converted value is:
$"/>
         <h:outputText id="out"  value="#{currencyBean.dollars}"/>
      </h:body>
</html>
```

In Code Snippet 7, the response page displays the output of the conversion by accessing the dollars attribute of the currencyBean, which is a managed bean. Figure 8.2 shows the Web page when the code given in Code Snippet 5 runs successfully.

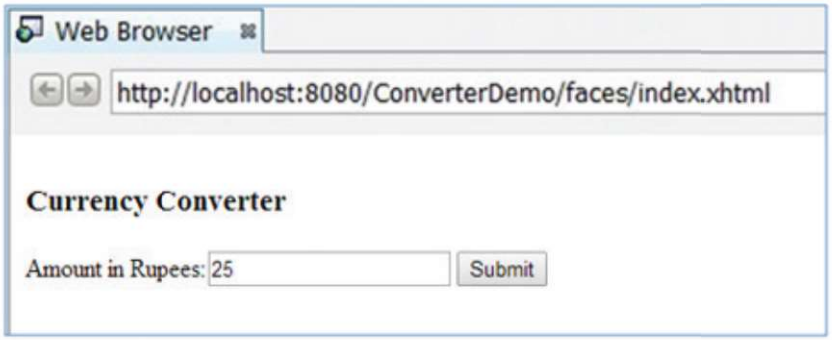

Figure 8.2: Index Page for the Converter

Figure 8.3 shows the converted value on the response.xhtml page after clicking Submit.

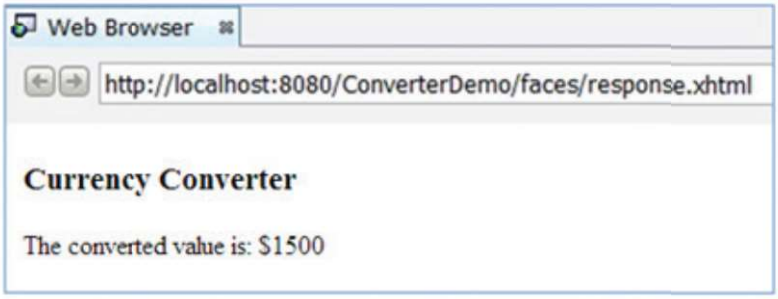

Figure 8.3: Output of Currency Converter

Custom Converters for complicated conversions can be created by implementing the Converter interface.

Concepts