Fundamentals of Java Enterprise Components

Session: 9

Facelets

Objectives

- Define Facelets
- Describe the lifecycle of Facelets
- Describe various components and resources used in Facelets
- Describe various resource library components of Facelets
- Describe how Facelets is compatible with HTML markup
- Explain how to use expression language with Facelets

Facelets

A Web template system used to create Web pages through view declaration language.

Open source technology compatible with HTML.

The components in Facelets are arranged in a tree structure known as component tree.

Component tree determines the navigation through the components of the user interface.

Features of Facelets

- Allows specifying the UIComponent trees in separate files
- Provides templating and decorators for the pages
- Supports expression language in the tags and allows build time validation of EL expressions
- Has a Facelet tag library which supports view definition along with the JSF tags
- Is compatible with XHTML and HTML
- Facelet APIs are not dependent on the container
- XML configuration files are not essential

Advantages of Facelets

- Provides for reuse of code through templates and composite components.
- Allows for content reuse through referencing a file or through defining custom tags.
- Allows compile time validation of EL expression.
- Facelets are rendered independently through a render kit and are compatible with any render kit.

Tag Libraries in Facelets

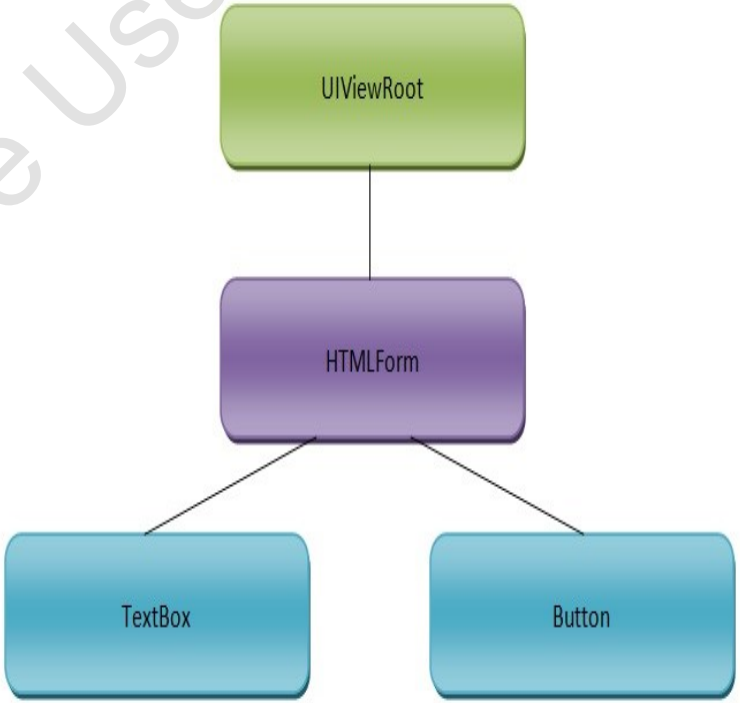
Facelets use XML namespace declarations to support JavaServer Faces tag library mechanism. Following table provides the tag libraries that can be used to add components to a Web page:

Tag Library	Prefix	Utility of the Tags
JavaServer Faces Facelet tag library	ui:	Tags are used for templating the Web pages
JavaServer Faces HTML tag library	h:	JavaServerFaces component tags for all UIComponent objects
JavaServer Faces Core tag library	f:	This library comprises tags for JSF custom actions. These tags are independent of any render kit
PassThrough Elements tag library	p:	Tags to support markup which is HTML friendly
JSTL Core tag library	c:	These are JSTL 1.2 core tags
JSTL Functions tag library	fn:	These are tags used by JSTL functions for version 1.2

Lifecycle of a Facelets Application 1-2

Facelets execute as part of a JSF application, which executes in two phases render and execute. Following are the steps in the life cycle of the Facelets application lifecycle:

- When a page is created through Facelets template is requested, a new component tree is created whose root is javax.faces.component.UIViewRoot object.
- Based on the component tree defined by the UIViewRoot, the view is populated with the components for rendering. Given figure shows a sample component:



Lifecycle of a Facelets Application 2-2

- Based on the client request, certain response is generated by the JSF application. This response is rendered back to the client.
- The state of the response view is saved for the next request. This state includes the state of input components and the form data.
- Based on the response received by the client if the client makes another request, then the saved state of the Web page is restored.
- The process is repeated again for a new request, with intermediate validations as applicable to the request.

Creating a Simple Facelets Application

An user registration module is implemented through Facelets. Following is a typical flow of data for an user registration module:

- The user sends details required for the registration as a request to the server.
- Server invokes servlets or managed beans to handle the request.
- The managed bean checks with the database whether the requested user name is available for allocation to the current user or not.
- If the user name can be allotted to the current user, then the registration is confirmed otherwise the application prompts for a new user name and password. This process continues until a valid user name is allotted to the user.
- The final page is a confirmation page with the details entered in the login page.

Facelets generates the view of the application through component tree and also defines the page navigation of the application.

Using Facelets Templates 1-6

Template defines the format of the Web page. Following code depicts the template for the header content of the Website of an 'XYZ' bank, which is created in a file named headertemplate.xhtml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>TODO supply a title</title>
        <meta name="viewport" content="width=device-width"/>
     </head>
    <body>
        <div>
           <ui:composition>
                 <header> XYZ Bank</header>
             </ui:composition>
        </div>
    </body>
</html>
```

Using Facelets Templates 2-6

Following code depicts the template for the footer content of the Website of an 'XYZ' bank, which is created in a file named footertemplate.xhtml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>TODO supply a title</title>
        <meta name="viewport" content="width=device-width"/>
    </head>
    <body>
        <div><ui:composition>
        <footer>XYZ Bank…Best in class</footer>
        </ui:composition></div>
    </body>
</html>
```

Using Facelets Templates 3-6

Following code depicts the template created for representing the default content of the Web page, which is stored in a file named contenttemplate.xhtml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    <head>
        <title>TODO supply a title</title>
        <meta name="viewport" content="width=device-width"/>
    </head>
    <body>
        <div>
 <ui:composition>
            <p>Welcome to XYZ Bank</p>
                <p> . </p>
                <p> . </p>
                <p> . </p>
                <p> . </p>
                <p> . </p>
            </ui:composition>
</div>
    </body></html>
```

Using Facelets Templates 4-6

Following code demonstrates the usage of header, footer, and default content templates defined earlier:

```
<?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:ui="http://java.sun.com/jsf/facelets">
    <h:head>
        <title>XYZ Bank</title>
    </h:head>
    <h:body>
       <ui:insert name="header" >
      <ui:include src="header.xhtml" />
   </ui:insert>
   <ui:insert name="content" >
      <ui:include src="contenttemplate.xhtml" />
   </ui:insert>
   <ui:insert name="footer" >
      <ui:include src="footertemplate.xhtml" />
   </ui:insert>
    </h:body> </html>
```

Using Facelets Templates 5-6

Following figure displays the resultant Web page:

Using Facelets Templates 6-6

- Templating helps in maintaining a standard look across multiple Web pages in an application.
- Facelets define a set of templating tags which are used to define templates.
- Following table shows the tags used for creating the templates:

Tag	Description
ui:component	An user interface component is created and added to the component tree. A table in a Web page can be defined through the component tag.
ui:composition	This tag uses a template optionally to define the set of components present in a Web page. All the content outside the composition tag is trimmed while rendering.
ui:debug	This tag is used to define and add a debug component to the Web page.
ui:decorate	The utility of this tag is similar to that of ui:composition tag. However, the content outside this tag is not trimmed.
ui:define	This tag defines the content which is expected to be inserted into the page through a template.
ui:include	This tag is used to reuse code in a Web page from other existing Web pages.
ui:insert	This tag is used to insert content into the Web page.
ui:param	This tag is used to add parameters to an included file.
ui:repeat	This tag is equivalent to loop constructs in Java. As in loop constructs the repeat tag also repeats a block of statements.
ui:remove	This tag is used to remove content from the Web page.

Composite Components 1-8

Composite component is an aggregation of facelet components such as validators, converters, tags, and so on defined as a single component.

Following are the steps for creating a composite component:

- Composite components are defined through the components in the namespace as follows:
  - xmlns:composite=”http://java.sun.com/jsf/composite”
  - Include the namespace declaration in the xhtml file in which the composite components are being defined for the application.
- Every composite component has an interface and implementation associated with it. The definition of the interface and implementation is done through the tags composite:interface, composite:attribute, and composite:implementation.
- The .xhtml file in which the composite components are defined is to be placed in the resources folder of the project.

Composite Components 2-8

Following table shows the tags that can be used to define composite components:

Tag	Function
composite:interface	This tag is used to determine the variables which are exposed to the user to enable usage of the composite component.
composite:implementation	If there is an interface element defined in the composite:interface, then there should be a corresponding implementation element to be defined.
composite:attribute	This tag defines the attributes of the composite components. These attributes may assume a value when the composite component is created.
composite:insertChildren	Composite attributes allow for the definition of component trees within the composite component. This tag indicates the location where the component tree has to be added.
composite:ValueHolder	This tag allows accommodating a ValueHolder object within the component.
composite:EditableValueHolder	This tag allows for holding an EditableValueHolder object.
composite:ActionSource	This tag accommodates an ActionSource object in the component.

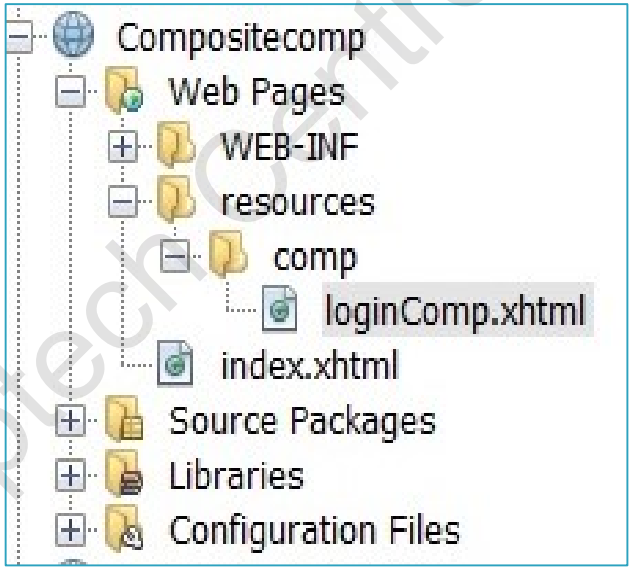
Composite Components 3-8

Following code demonstrates how to create composite components:

```
<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:h="http://java.sun.com/jsf/html"

      xmlns:composite="http://java.sun.com/jsf/composite" >

<composite:interface>

   <composite:attribute name="usernameLabel" />

   <composite:attribute name="usernameValue" />

</composite:interface>

<composite:implementation>

<h:form>

   #{cc.attrs.usernameLabel} : <h:inputText id="username"
value="#{cc.attrs.usernameValue}" />

<h:commandButton value="Submit" action="response"></h:commandButton>

</h:form>

</composite:implementation>

</html>
```

Composite Components 4-8

The code for composite component is created in a file named loginComp.xhtml and stored in a resources directory in the Web Pages directory of the project. Following is the hierarchy of directories:



Composite Components 5-8

Following code demonstrates the usage of the composite component:

```
<?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://java.sun.com/jsf/html"
   xmlns:j="http://java.sun.com/jsf/composite/comp">
   <h:form>
        <j:loginComp usernameLabel= "Enter User Name: "
        usernameValue= "#{userData.name}" />
   </h:form>
</html>
```

Composite Components 6-8

The data accepted in the composite component has to be bound with the bean properties. Following code depicts the bean class UserData:

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
@ManagedBean
@SessionScoped
public class UserData {
    private String name;
    public UserData() {
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

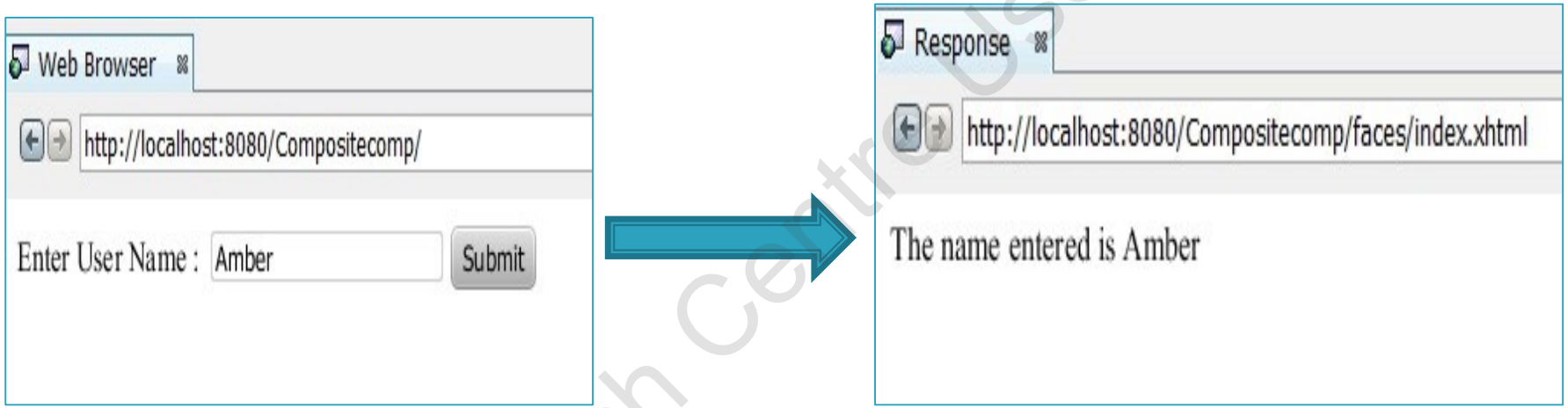
Composite Components 7-8

Following code depicts response page:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <head>
        <title>Response</title>
        <meta name="viewport" content="width=device-width"/>
    </head>
    <body>
            <h:outputText id ="output" value="The name entered is "/>
            <h:outputText id ="output" value="#{userData.name}"/>
    </body>
</html>
```

Composite Components 8-8

Following is the execution pattern after the application is deployed and executed:



Web Resources

Web resources refer to software elements such as images, style sheets, and so on, which are essential for proper rendering of the Web pages.

Resources of an application can be packaged in a directory and be located as a sub directory /resources directory of the application.

If a resource is packaged in the class path of the Web application, then it must be located as a sub directory of META-INF/resources of the application.

Every resource of the application is associated with a resource identifier. The format of resource identifier is as shown below:

- `[locale-prefix/][library-name/][library-version/]resource-name[/resource-version]`

Relocatable Resources

Relocatable resources in a Web page are those which are referenced in one section of the Web page but are rendered in another part of the Web page.

In order to achieve this the target attribute of the tag is used.

Relocatable resources can be used for composite components that use stylesheets or the composite components that use JavaScript.

Resource Library Contracts 1-2

- Resource library contracts are used to define the look of the Web pages.
- All the resource library contracts are defined in the contracts folder of the application.
- Each contract folder has a CSS file and a template file which together define the appearance of the Web pages.
- Each template has insertion points defined for the resources.
- The template, the insertion points, and CSS files together form the resource library contract.

Resource Library Contracts 2-2

Resource library contracts can be defined according to the pattern as shown in figure:

Following demonstrates the format of mapping the contracts to the Web pages:

```
<application>
 <resource-library-contracts>
 <contract-mapping>
 <url-pattern> /RegisteredUsers/*
</url-pattern>
 <contracts>colored</contracts>
 </contract-mapping>
<contract-mapping>
 <url-pattern> * </url-pattern>
 <contracts>colorless</contracts>
 </contract-mapping>
</resource-library-contracts>
</application>
```

HTML5 Friendly Markup 1-3

JSF allows the user to use HTML components in the Web pages or create custom components that suit the application.

The HTML5 support to JSF can be categorized into the following:

- Pass-through elements
- Pass-through attributes

Pass-through elements are rendered as Facelet tags.

Pass-through attributes are rendered through the browser directly without any intermediate interpretation.

All the pass-through elements and attributes are supported through the namespace - xmlns:p=http://xmlns.jcp.org/jsf/passthrough

In general, a non-JSF element can be made a pass-through element by using at least one of the attributes from the namespace -http://xmlns.jcp.org/jsf.

HTML5 Friendly Markup 2-3

Following table shows some of the pass-through elements of HTML5 which are represented as a combination of element and jsf attribute:

HTML Element	jsf Attribute	Facelet Tag
A	jsf:action or jsf:actionListener	h:commandLink
A	jsf:value	h:outputLink
A	jsf:outcome	h:link
Body		h:body
Button	jsf:outcome	h:button
button		h:commandButton
Input	type = “button”	h:commandButton
Input	type = “checkbox”	h:selectBooleanCheckBox
Input	type = “*”	h:inputText

HTML5 Friendly Markup 3-3

Pass-through attributes allow passing JavaServer Faces attributes to the browser without any intermediate interpretation.

Pass-through attributes can be specified in any of the following ways:

- By using the given JavaServer Faces namespace for the pass-through attributes.
  - `xmlns:p=http://xmlns.jcp.org/jsf/passthrough`
- Pass-through attributes can also be added through f:passThroughAttribute tag.
- To pass a group of non-JavaServer Faces attributes, f:passThroughAttributes tag can be used within the component tag.
- A set of values can be passed as map declarations where each entry of the Map object is name-value pair.

Summary

- Facelets is a view declaration language used to define views for a Web application.
- It can also define reusable templates which can be used among multiple Web pages.
- Facelets have a set of core tags and can use other tag libraries to define the view.
- Facelets provide for definition of user defined tags, composite components, and templates.
- Web resources are included in the resources directory and the resource library contracts in the contracts directory of the application.
- Through resource library contracts facelets provides a uniform look and feel among the Web pages of the application. It also allows defining the appearance of a subset of pages.
- HTML 5 tags and non JavaServer Faces attributes can be implemented through pass-through elements and pass-through attributes.