Fundamentals of Java Enterprise Components

Session: 7

Introduction to JavaServer Faces

Objectives

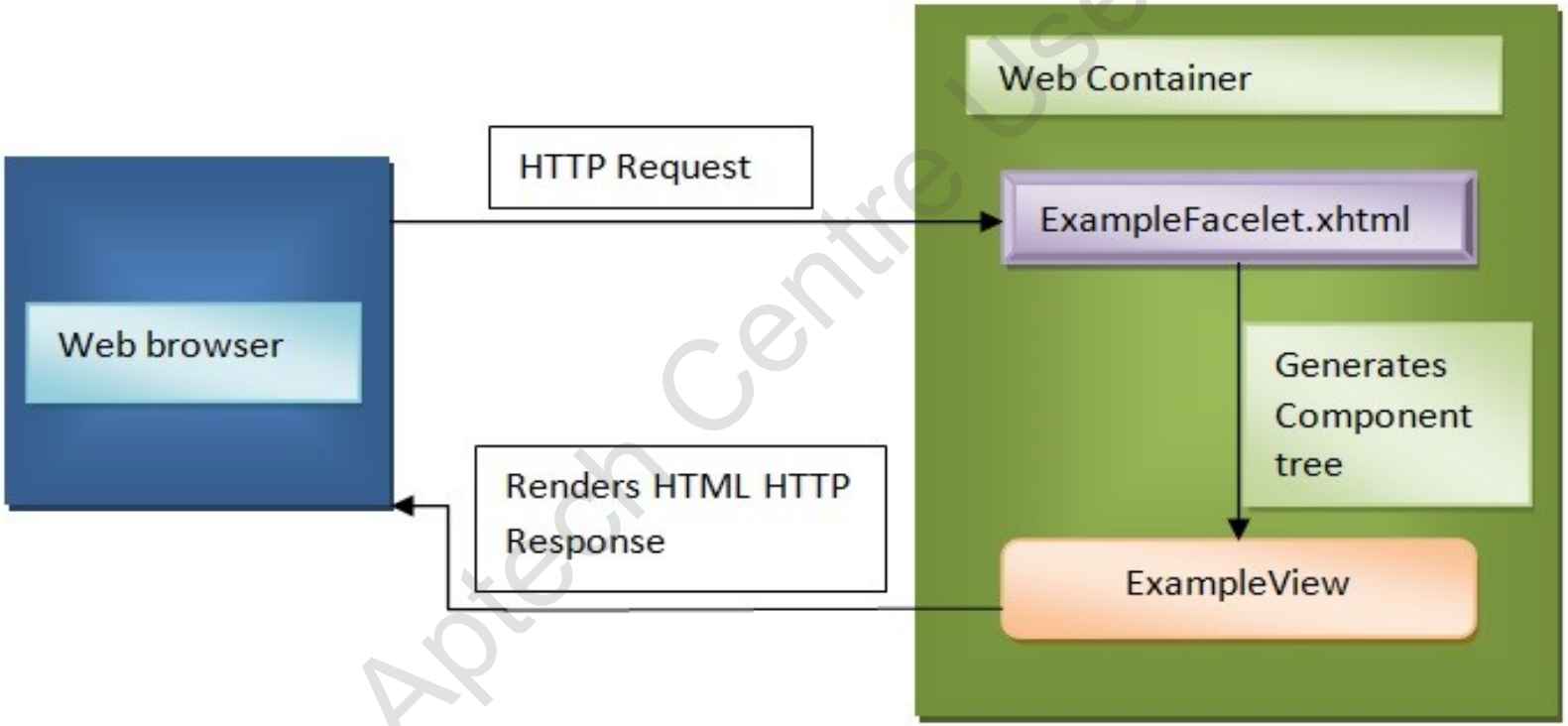
- Describe the basic structure and usage of JSF
- Compare it with other technologies used to develop user interfaces
- Explain the tag library of JSF
- Develop a sample application using JSF
- Explain the component model of UI
- Explain the navigation model
- Describe the lifecycle of JSF applications

Introduction to JSF Technology 1-2

- JSF is a Java based framework used to build and maintain server-side user interface components.
- User interface development is done through an API and supporting tag library.
- The API has implementations for page navigation, event handling, server-side validation, and so on.
- The tag library defines various tags used to connect the user interface components to Web pages and server-side objects.
- Provides a framework to develop custom components and custom tag library.
- Constructs UI from reusable components.
- The UI components are capable of saving and restoring UI state beyond server requests.

Introduction to JSF Technology 2-2

Following figure shows the functioning of a JSF application:



Benefits of JSF Technology

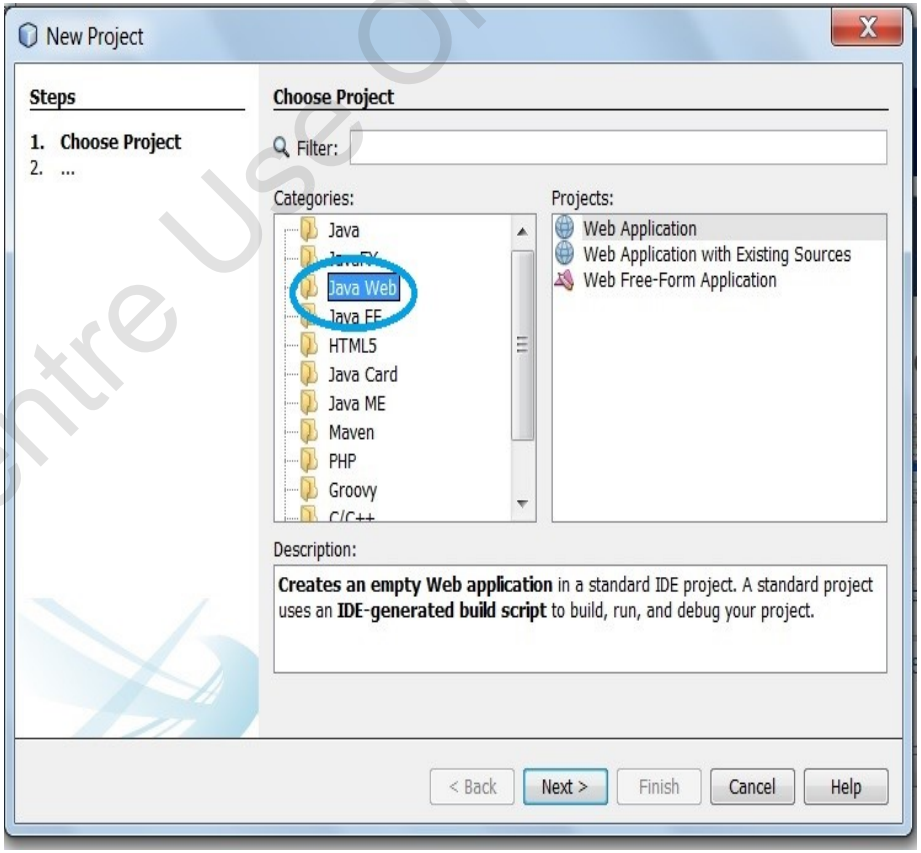
- JSF separates the presentation and application logic on the server-side
- Supports loosely coupled architecture
- Does not limit the developers in terms of scripting or markup language
- The Unified Expression Language enables the JSFs to collaborate with JSPs and servlets
- Facelets of JSF enable reuse of code and also enable extending the functionality over several components of the application through templates and composite component features
- Implicit navigation rules of JSF allow quick configuration of page navigation

JSF Application Demonstration 1-7

A JSF application comprises a set of Web pages written in XHTML.

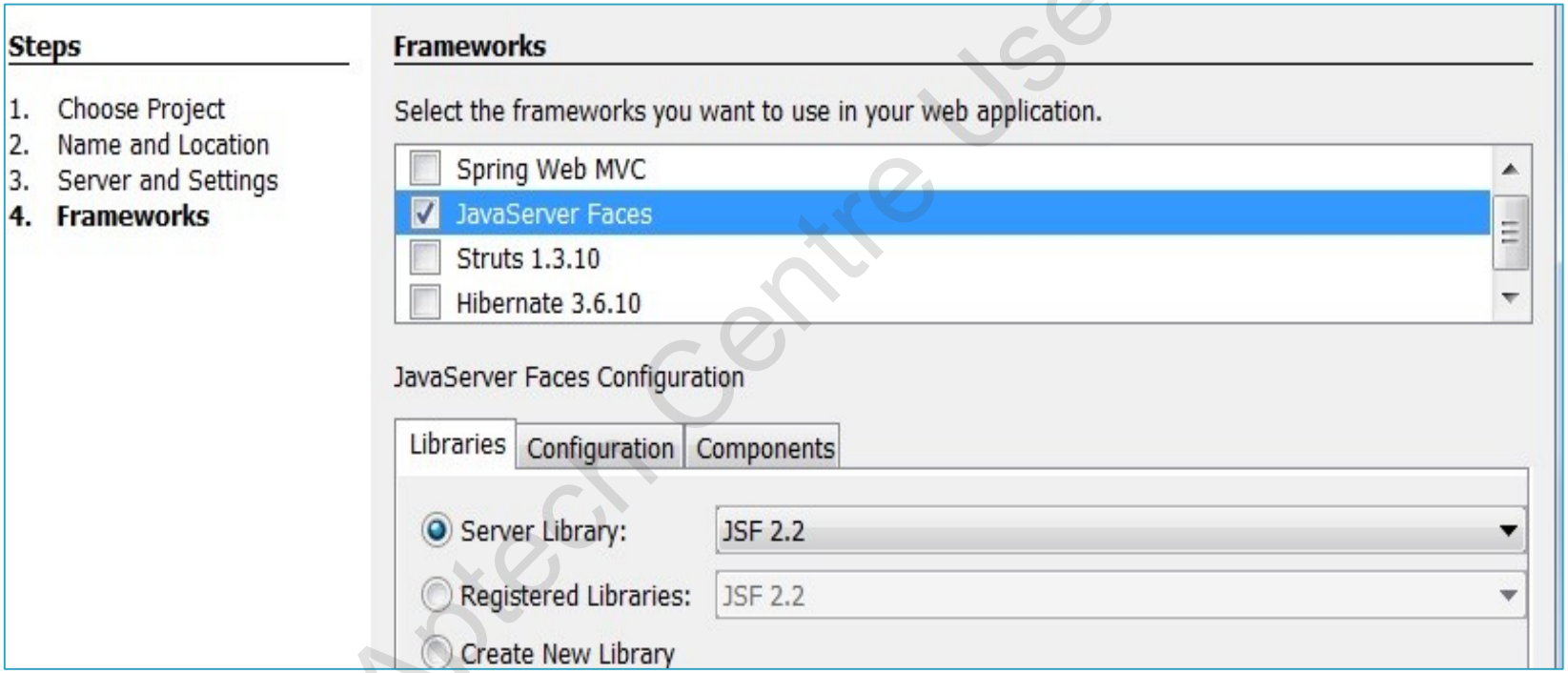
These Web pages in turn have JSF tags and managed beans.

A JSF application is created in the same manner as other Web applications, by choosing a Web application while creating new projects in NetBeans.



JSF Application Demonstration 2-7

- To create a JSF project, select the JavaServer Faces checkbox as shown in the following figure:



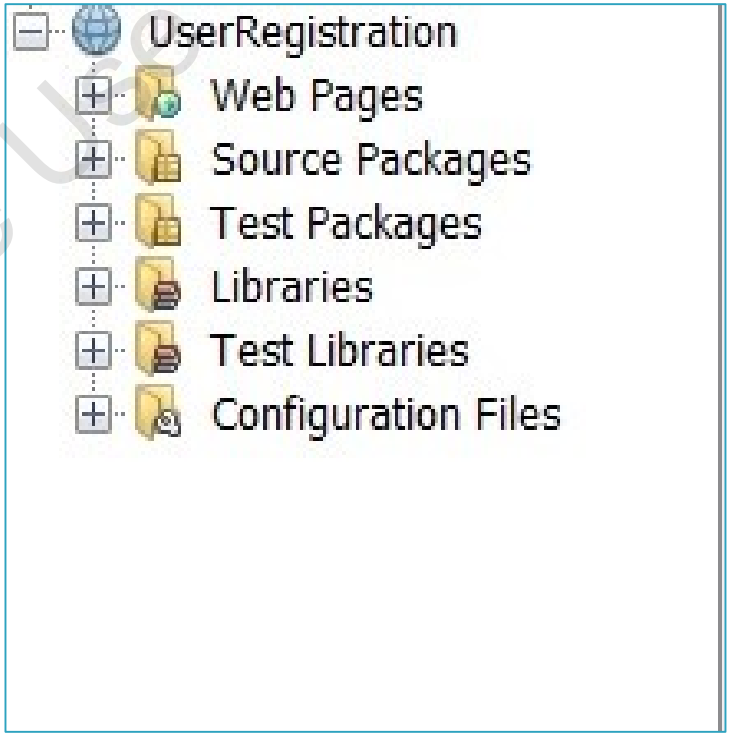
JSF Application Demonstration 3-7

Netbeans creates a project with the directory structure as shown in figure.

A JSF project named 'UserRegistration' has been created here.

The files web.xml and index.xhtml are created by default.

The project stage is set to 'Development' by default. The developer can check the stage of the project in web.xml file.



JSF Application Demonstration 4-7

index.xhtml is the Web page created by default in the JSF application. Web application execution can begin from an .html, .xhtml, or .jsp page. Following code is generated for index.xhtml file created for the project 'UserRegistration':

```
!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>
        <title> Facelet title</title>
</h:head>
<h:body>
        Hello from Facelets
</h:body> </html>
```

JSF Application Demonstration 5-7

Following code demonstrates the modified index.xhtml file:

```
!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>
         <title> Facelet title</title>
</h:head>
<h:body>
<f:view>
    <h:form>
    <h:outputLabel value="Name:" for="name"/>
    <h:selectOneMenu id="prefix"
value="#{Members.reg.RegistrationManagedBean.prefix}" >
            <f:selectItem itemLabel="Mr." itemValue="Mr"/>
            <f:selectItem itemLabel="Mrs." itemValue="Mrs"/>
            <f:selectItem itemLabel="Ms" itemValue="Ms"/>
               </h:selectOneMenu>
<h:inputText id="Name" label="Name" required="true"
```

JSF Application Demonstration 6-7

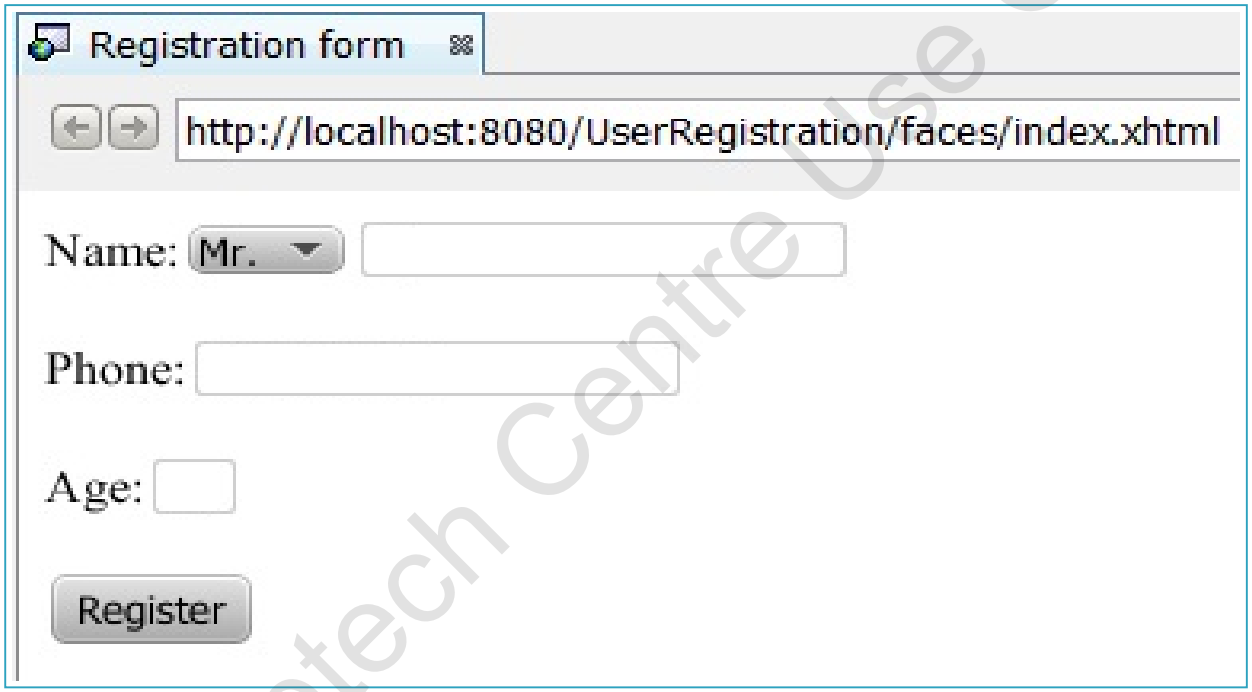
```
value="#{Members.reg.RegistrationManagedBean.Name}"/>
        <h:message for="firstName" />
        <p></p>
        <h:outputLabel value="Phone:" for="phone"/>
         <h:inputText id="contactno" label="Phone"
                  required="true"
                  value="#{Members.reg.RegistrationManagedBean.contactno}">
         </h:inputText>
        <p></p>

         <h:outputLabel for="age" value="Age:"/>
         <h:inputText id="age" label="Age" size="2"
value="#{Members.reg.RegistrationManagedBean.age}"/>

         <p></p>
          <h:commandButton id="register" value="Register" action="confirm" />
        </h:form>
    </f:view>
</h:body> </html>
```

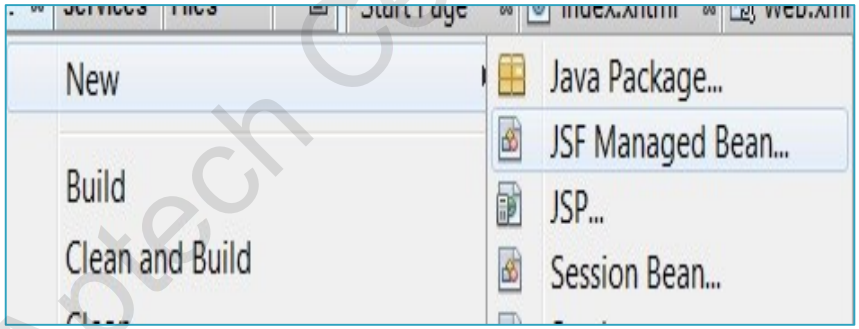
JSF Application Demonstration 7-7

Following is the output generated after executing the application:



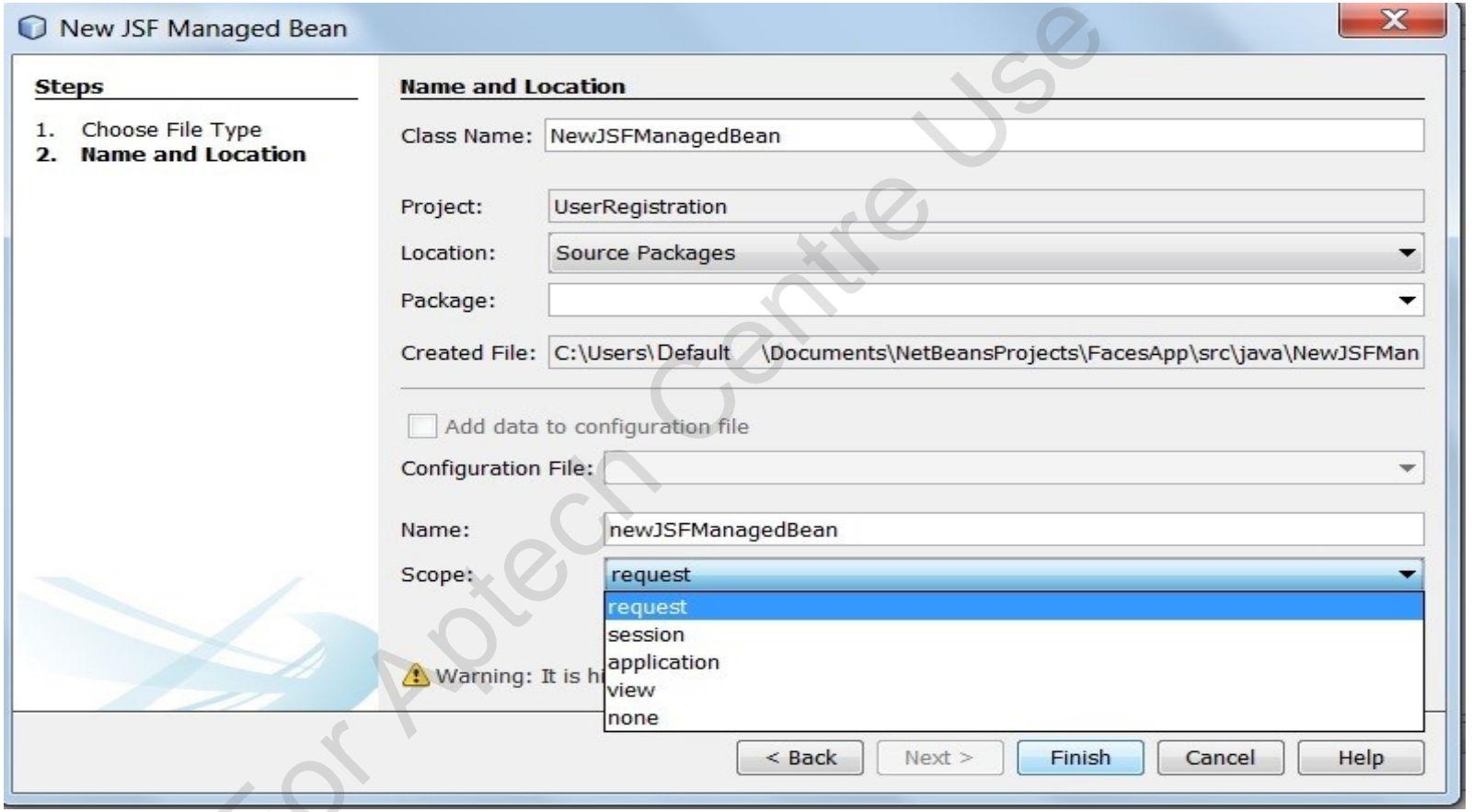
Creating a Managed Bean 1-3

The business logic of the application is implemented through the JSF managed bean. A JSF managed bean can be created by choosing the option from the New submenu as shown in the following figure:



Creating a Managed Bean 2-3

Following is the dialog box for creating the JSF Managed Bean. Various options pertaining to the JSF managed bean are selected here.



Creating a Managed Bean 3-3

There are five options for the scope of the managed bean and they are as follows:

- **request** – bean available for a single HTTP request
- **session** – bean available for the entire session
- **application** – bean available as long as application is active
- **view** – bean is available for Ajax requests
- **none** – implies that bean is not created for any particular scope but it is invoked on demand

Once the JSF managed bean is created, the variables are added to the bean which are referenced in the JSF page.

There should be one variable for every data field read from the form. The getter and setter methods are invoked for each of the variables.

Creating the Response Page 1-3

- A response xhtml page is created for the index page and named as specified in the action field of the index.xhtml file.
- Following code demonstrates the response page 'confirm' as defined:

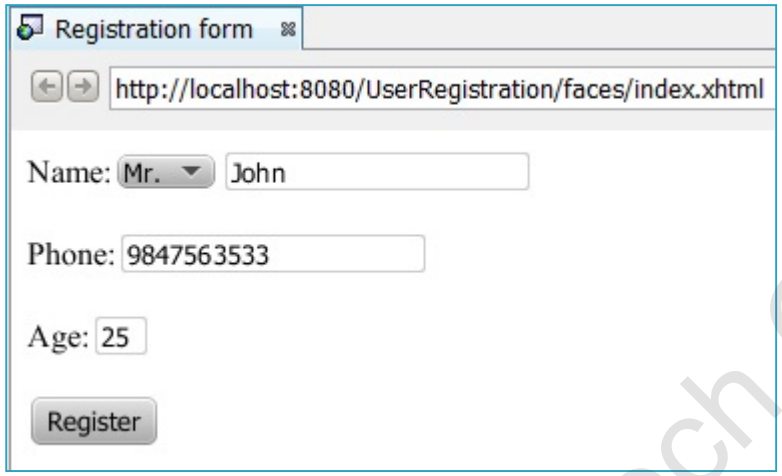
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title> Confirmation</title>
    </h:head>
    <h:body>
        <h:outputText value="Registration confirmed"/>
        <p></p>
          <h:outputText value="Name:"/>
          <h:outputText
```

Creating the Response Page 2-3

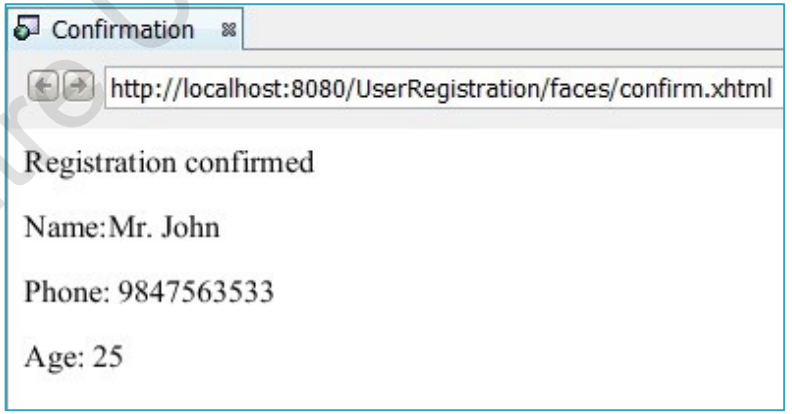
```
value="#{Members.reg.RegistrationManagedBean.prefix}" />
          <h:outputText value="
#{Members.reg.RegistrationManagedBean.name}" />
          <p></p>
          <h:outputText value="Phone:"/>
          <h:outputText value="
#{Members.reg.RegistrationManagedBean.contactno}"/>
          <p></p>
          <h:outputText value="Age:"/>
          <h:outputText value="
#{Members.reg.RegistrationManagedBean.age}" />
          <p></p>
     </h:body>
</html>
```

Creating the Response Page 3-3

On running the project and submitting the values on the index page, the following response page is generated:



Index Page



Response Page

UI Component Model

JavaServer Faces provides a repository of user interface components which can be used to create JavaServer Faces view.

The JSF component architecture has the following elements:

- javax.faces.component.UIComponent classes are used to define the components, their state, and behavior.
- A rendering model defines the usage of the components.
- A conversion model defines the conversion of data format between components and registers data converters on the component.
- An event and listener model handles the events occurring with respect to the components.
- A validation model enables validators to be registered on a component to validate the data handled by the component.

Component Classes of the UI 1-3

A predefined set of UI component classes and associated behavioral interfaces are provided by JSF technology which can be used to create user interface components.

This basic set of classes enables the developers to create customized components.

The base class for all the classes is javax.faces.component. UIComponent is an abstract class.

The base class of all the components is defined in the class UIComponentBase.

Component Classes of the UI 2-3

Following table shows some component classes that are available as a part of JavaServer Faces technology:

Component	Description
UIData	An instance of this class is bound to the collection of data which is an instance of javax.faces.model.DataModel. The collection can be a database.
UIColumn	An object which represents a column of data from the UIData object.
UICommand	An instance of this class fires an action in the user interface.
UIForm	An instance of UI form has several child UI components which can accept user input.
UIGraphic	An instance of UIGraphic is used to display an image.
UIInput	UIInput instance can be used to accept input from the user.
UIMessage	An instance of UIMessage is used to display an error message.
UIMessages	Used to hold set of error messages.
UIOutcomeTarget	Used to display a hyperlink.
UIOutput	Used to display output.

Component Classes of the UI 3-3

The component classes have certain behavior which is implemented through behavioral interfaces. Following table shows various behavioral interfaces provided by JSF:

Interface	Description
ActionSource	This interface implies that the current component can fire an action.
ActionSource2	Extends ActionSource functionality and allows the components to use expression language while referencing to the event handlers.
EditableValueHolder	Allows providing additional features to input data holders.
NamingContainer	This container holds all the objects which require an unique identifier.
StateHolder	Defines that the component has a state that need to be saved between requests.
ValueHolder	It implies that there is a local value associated with the current component.
javax.faces.event.SystemEventListenerHolder	Maintains a set of event listener instances for various events.
javax.faces.component.behavior.ClientBehaviorHolder	Instances of client behavior can be attached through this interface.

Rendering Components on Web Page

- JavaServer Faces architecture separates the definition of components and rendering the components.
- This separation of code enables reuse of code.
- The user interface component is rendered through a renderer class.
- A render kit maps the UI component classes onto component tags based on the client.
- A render kit defines a set of javax.faces.render.Renderer class for all the supported components.
- When a custom tag is defined, its functionality and rendering attributes should also be defined.

Conversion Model

The interpretation of data by the managed beans and user interface components is different.

The application therefore has two views of data as follows:

- Model view
- Presentation view

JSF technology makes the conversion automatic among these two views.

JSF allows registering a javax.faces.convert.Converter implementation on UIOutputComponent. When a converter is registered it is responsible for the conversion of data between the model view and presentation view.

Event and Listener Model 1-2

There are three types of events according to JSF specification as follows:

- Application events
- System events
- Data model events

Events in an application are generated by objects of type UIComponent.

The application is notified regarding the events through listener objects.

Components are registered with the listener objects whose events have to be captured.

Event and Listener Model 2-2

UI components may generate two types of events as follows:

- **Action events** -An action event generates an object of class javax.faces.event.ActionEvent, when the user activates a component that implements the behavioral interface ActionSource.
- **Value change events** - a value change event generates an object of class javax.faces.event.ValueChangeEvent. This event occurs when certain value of a component which accepts user input, get change. These are instances of UIInput.

When an action event or a value change event occurs in the application, the system can react through an event listener or by invoking a managed bean to handle the event.

System events occur due to objects in the application but not due to UIComponents. These events have to be handled in the application definition.

A data-model event occurs when there is a change to the underlying database of the application.

Validation Model

JSF provides validation of data in the user input components through a set of standard classes.

Standard tags for validation are implemented through javax.faces.validator.Validator.

In addition to the validators on individual components, the developer can also register default validators on the application.

Custom validators can also be created on the application. There are two ways to implement custom validation as follows:

- By implementing a Validator interface. While implementing the validator interface, the validator must be registered with the application and corresponding custom tag must be created.
- By implementing a managed bean.

Navigation Model 1-3

Navigational model defines the order in which the pages in the application should be traversed according to the control flow of the application.

In JSF, navigation is the set of rules for selecting the subsequent page to be loaded by the application.

Navigation can be implicit or user defined.

The action attribute of the event defines the Web page to be loaded. This is defined by the developer while implementing the logic of the application. Following is an example of implicit navigation:

- `<h:commandButton value="submit" action="response">`

Navigation Model 2-3

- User defined navigation implies navigating through the Web pages according to the explicit configuration in the config files of the application.
- Navigation rules are defined in the configuration files such as faces-config.xhtml.
- Following is the default structure of a navigation rule:

```
<navigation-rule>
      <description></description>
      <from-view-id></from-view-id>
      <navigation-case>
            <from-action></from-action>
            <from-outcome></from-outcome>
            <if></if>
            <to-view-id></to-view-id>
      </navigation-case>
</navigation-rule>
```

Navigation Model 3-3

Following is an example of navigation rule definition:

```
<navigation-rule>
<from-view-id>/Registration.xhtml</from-view-id>
      <navigation-case>
            <from-outcome>success</from-outcome>
      <to-view-id>/Confirmation.xhtml</to-view-id>
      </navigation-case>
</navigation-rule>
```

Lifecycle of JSF Application 1-5

- The lifecycle of a JSF application begins when a client makes an HTTP request and is completed when the response is rendered to the client.
- There are mainly two phases in the lifecycle:
  - Execute
  - Render
- The execute phase is organized according to a component tree, including various tasks such as validation of the input data, handling of component events, and so on.
- This component tree is also known as view.

Lifecycle of JSF Application 2-5

Following is the order of events in JSF application lifecycle:

- Restore view of the Web page.
- Apply requests to the Web page.
- Process the events that trigger in the Web page.
- Process the validations of the data in the Web page.
- Connect to the managed bean to update the model values of the application.
- If updating the model values triggers any events, handle them.
- Render the response.

There are two types of requests handled during the lifecycle of the application as follows:

- Initial requests
- Postback requests

Lifecycle of JSF Application 3-5

The events that occur during the lifecycle of an application can be categorized into the following phases:

- Restore view phase
- Apply request values phase
- Process validations phase
- Update Model values phase
- Invoke application phase
- Render response phase

Lifecycle of JSF Application 4-5

Restore View phase

- If it is a postback request such as submission of data through a form, then a view corresponding to this page present in the FacesContext of the application is restored otherwise a new view is created.

Apply Request Values phase

- The request parameters retrieved in the restore view phase are restored.

Process Validations phase

- Once all the values are set, then the validation of the values is done in this phase.

Update Model Values phase

- After completion of data validation, the values of the components are set to the local values.

Lifecycle of JSF Application 5-5

Application Invoke phase

- All the application level tasks are carried out such as submitting the form.

Render Response phase

- The components of the Web page are rendered on the client-side according to the JSF specification. If it is an initial request, then the page is added to the component tree.

Summary

- JavaServer Faces is an application framework used to develop user interfaces based on templates.
- JSF follows component-based development of the UI and uses a standard tag library. It also allows developers to define custom tags for application development.
- The response of the request can be generated by communicating with a managed bean or without using the bean.
- The page navigation in JSF can be implicit or user-defined. If it is user-defined navigation, the navigation rules are defined in the configuration files.
- There are two important phases in the lifecycle of the JSF application, execute phase and render phase. The execute phase has other phases based on the inputs received and events triggered during the application execution.
- The lifecycle of JSF application can start with restore view phase if the request is a postback request. The lifecycle ends with a Render response phase.