Java Server Faces

Subject: Advanced Java Programming

Topic: JSF

INTRODUCTION TO JSF

- Java server pages is a web application framework used to develop the **web application with rich user interface**.
- The framework is based on **MVC** – Model View Controller architecture.
- It **simplifies the construction of user interface (UI) component**. These UI can interact with DB to store data.
- In JSF, the UI component can be created by the use of Application programming Interface(API).

Benefits of JSF

- JSF reduces the effort in creating and maintaining applications, which will run on a
- Java application server and will render application UI on to a target client.
- JSF facilitates Web application development by −
  - Providing reusable UI components
  - Making easy data transfer between UI components
  - Managing UI state across multiple server requests
  - Enabling implementation of custom components
  - Wiring client-side event to server-side application code

Benefits of JSF

- Binding of UI components with some model data
- Providing reusable UI components
- Making easy data transfer between UI components
- Managing UI state across multiple server requests
- Enabling implementation of custom components
- Handling different events on UI component on server- side
- Validating user-inputs
- Defining navigation rule

JSF ARCHITECTURE

- A JSF application is similar to any other Java technology- based web application; it runs in a Java servlet container, and contains

✓ JavaBeans components as models containing application-specific functionality and data

✓ A custom tag library for representing event handlers and validators

✓ A custom tag library for rendering UI components

✓ UI components represented as state-ful objects on the server

✓ Server-side helper classes

✓ Validators, event handlers, and navigation handlers

✓ Application configuration resource file for configuring application resources.

JSF ARCHITECTURE

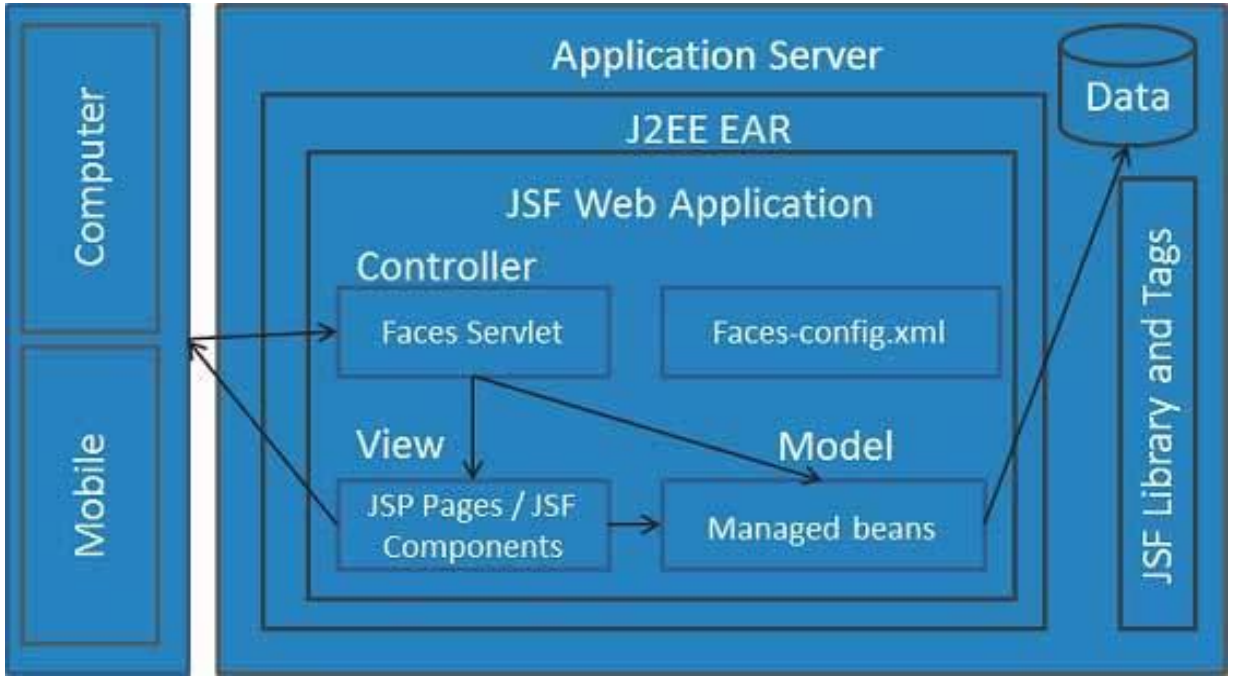
- **JSF technology** is a framework for **developing, building server-side User Interface Components** and using them in a web application.
- JSF technology is based on the **Model View Controller (MVC)** architecture for separating logic from presentation

What is MVC Design Pattern?

1. Model : Carries Data and login
2. View : Shows User Interface
3. Controller : Handles processing of an application.

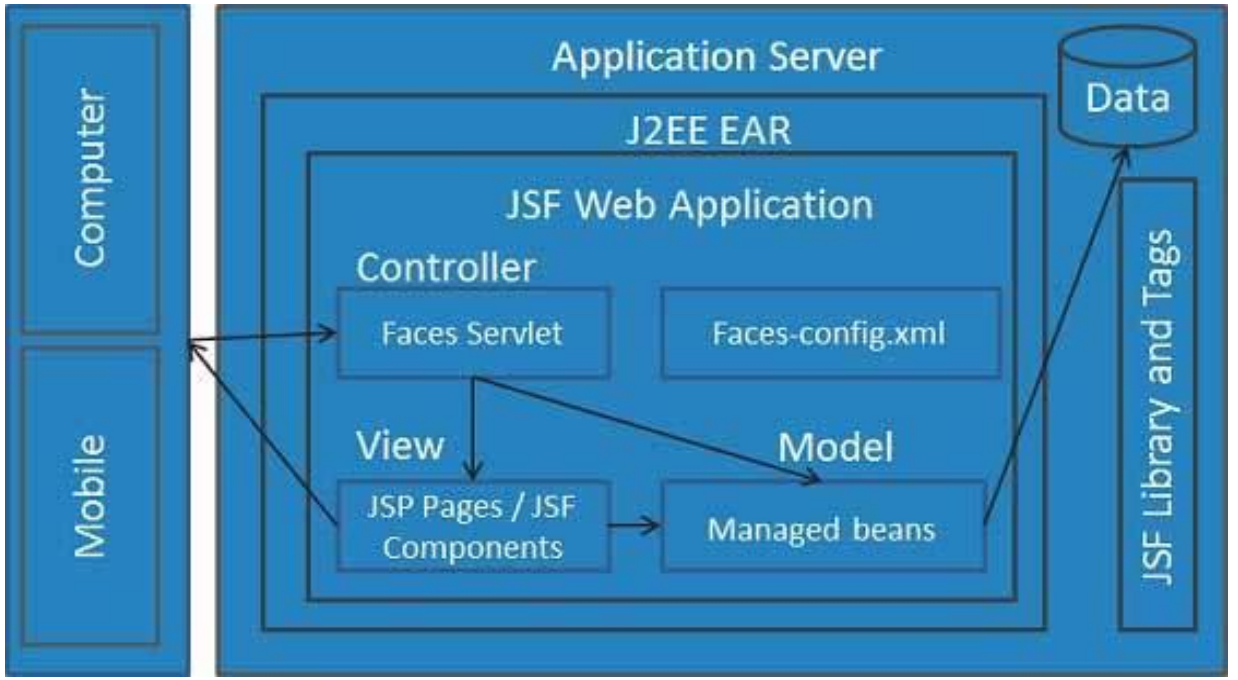
JSF ARCHITECTURE CONTINUED…

- User submit the request to the application server using web browser. This request is received by JSF Faces Servlet. This servlet is a part of JSF Web Application and it need not be written. It basically acts as controller. It routes the request to appropriate page. It can read config.xml file.
- The managed beans contains the form data which can access the backend or Db to retrieve desired data for processing the request. The managed bean acts as a Model.



JSF ARCHITECTURE CONTINUED…

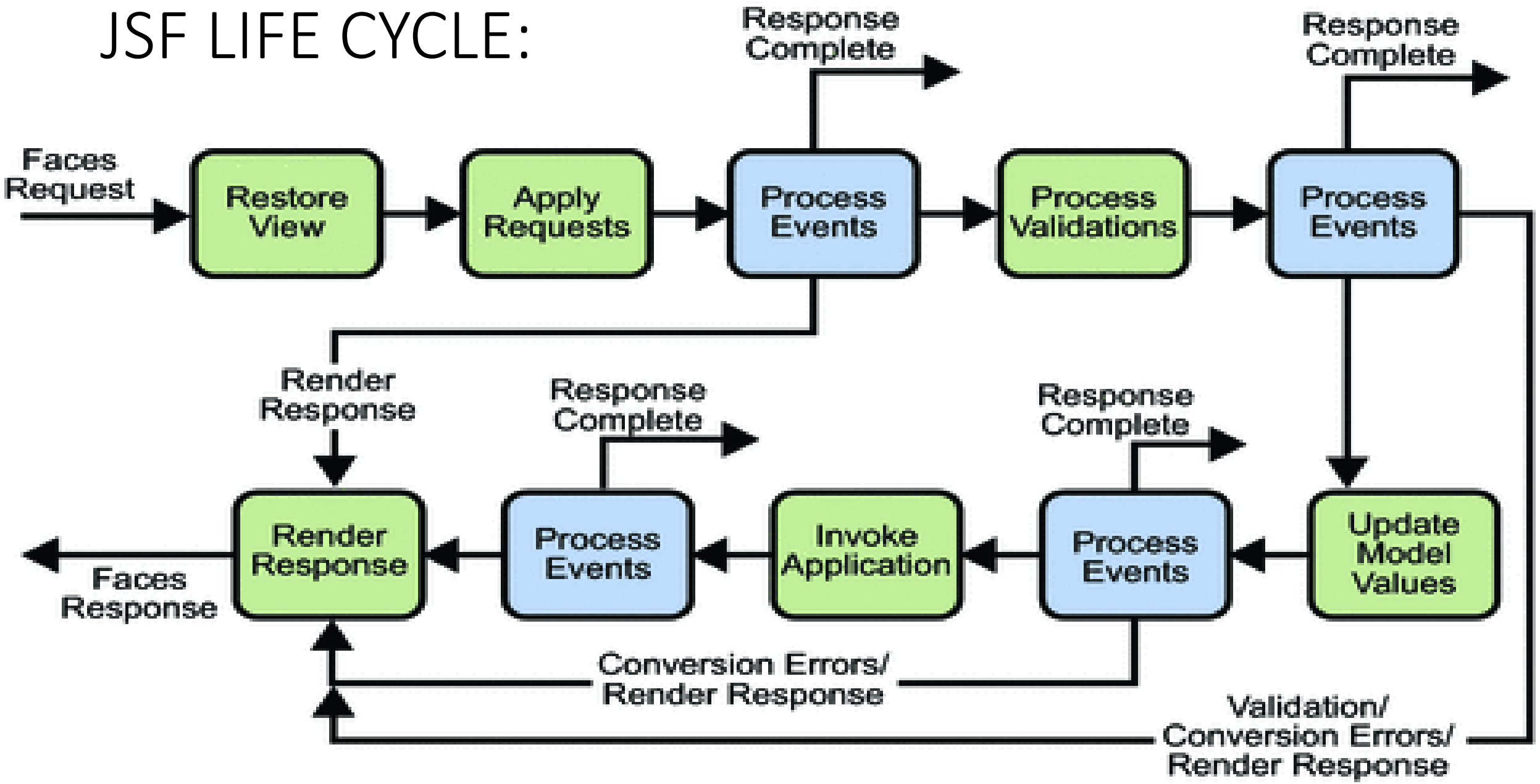
- The Face Servlet will determine and route the request to appropriate page for display information. Typically these web pages are in the xhtml file. This part acts as view.
- Finally the web page is rendered and sent back to the web browser.



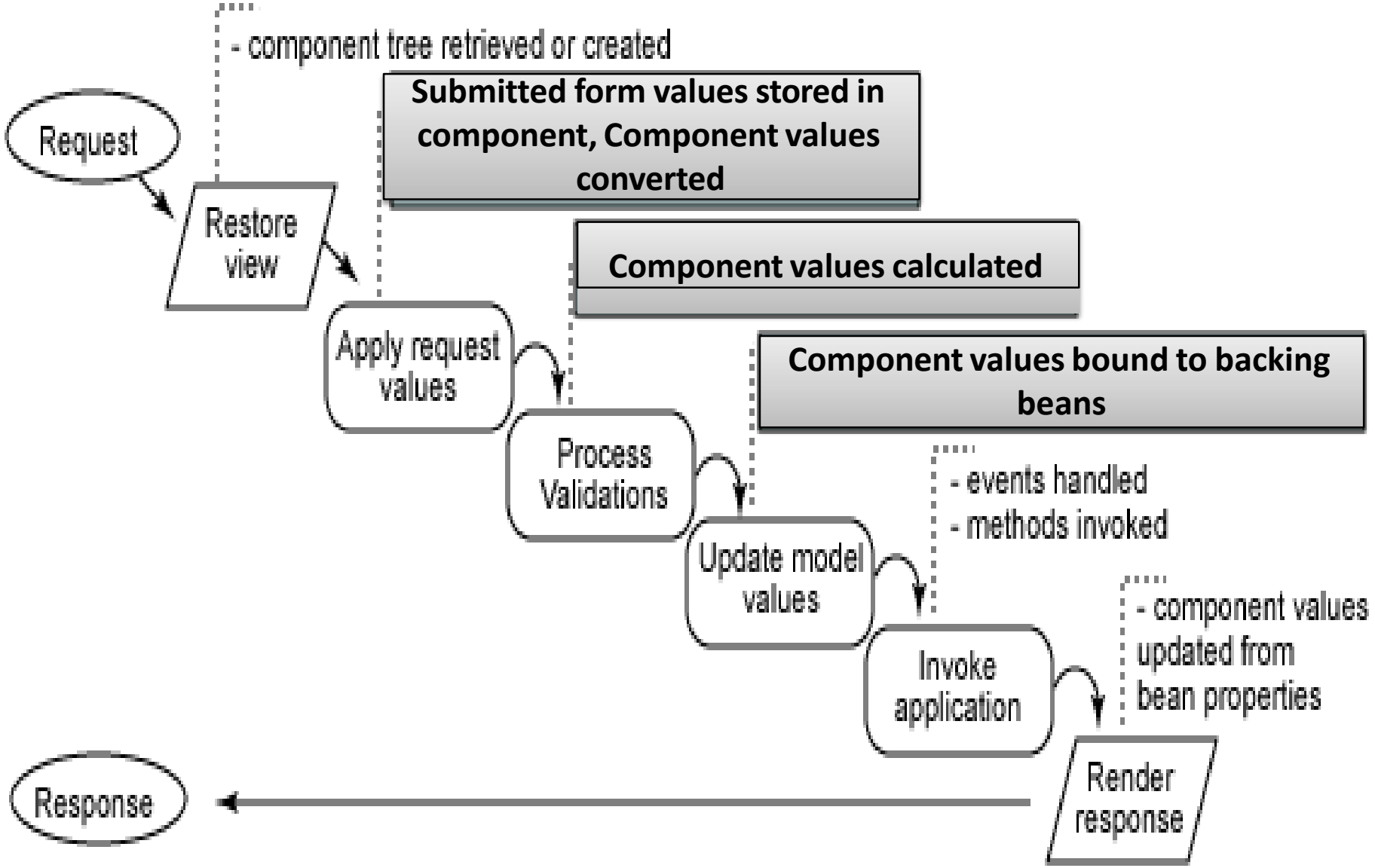
JSF elements

- UI Component
- Renderer
- Validator
- Backing Beans
- Convertor
- Event and Listeners
- Message
- Navigation

JSF LIFE CYCLE:



JSF LIFE CYCLE:

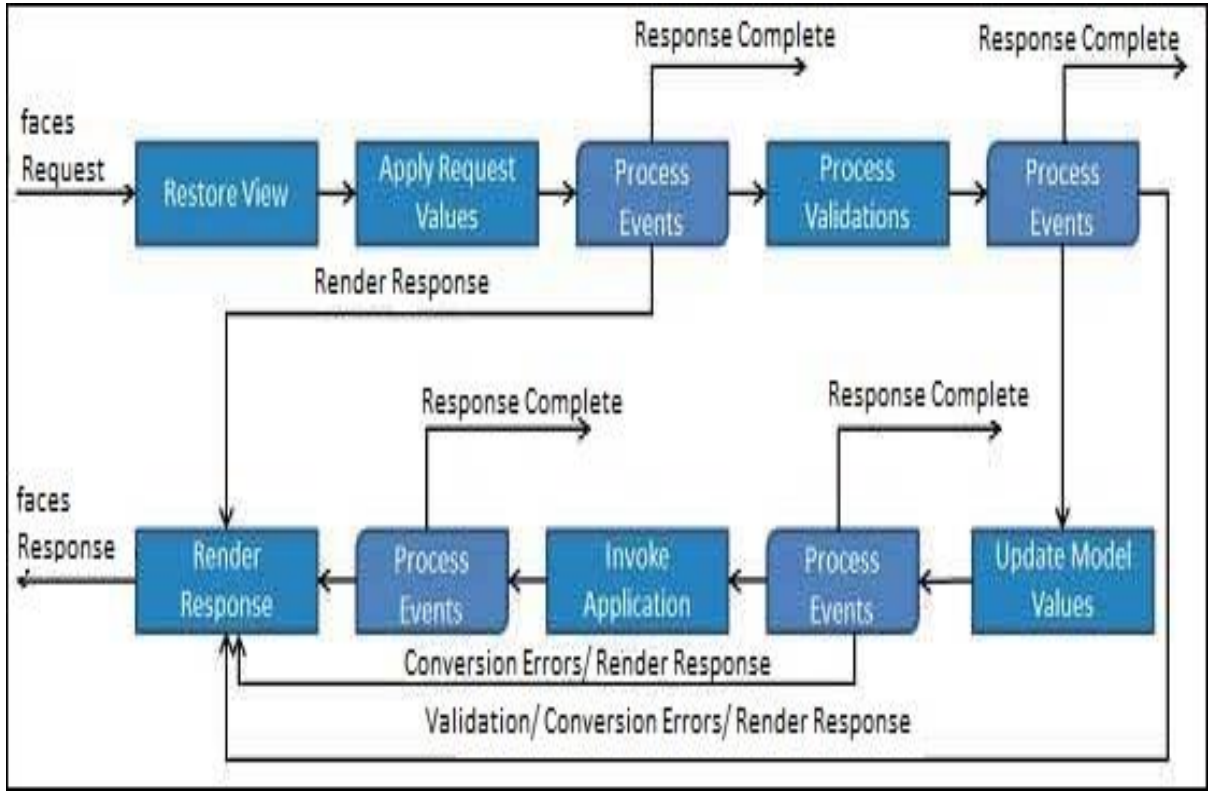


Six phases of JSF request processing life-cycle

1. The Restore View Phase
2. The Apply Request Values Phase
3. The Process Validation Phase
4. Update Model View Phase
5. The invoke Application Phase
6 The Render Response Phase

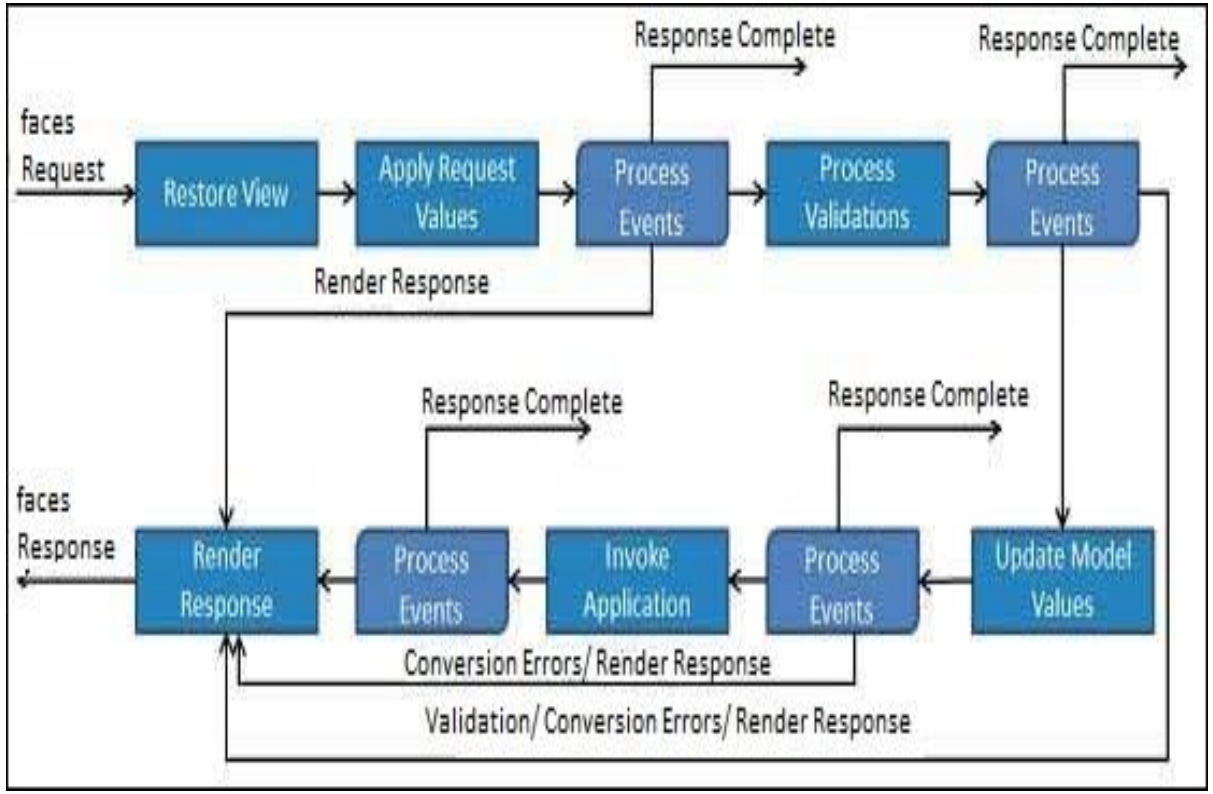
JSF LIFE CYCLE:

1. **Restore View:** This is the **first phase** in JSF Request processes life cycle. This phase is **used for constructing view to display the front end.** This view is stored in FacesContext instance and using this information request can be processed.
2. **Apply request Values:** After the **component tree is created**, each component in the component tree extracts its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors.



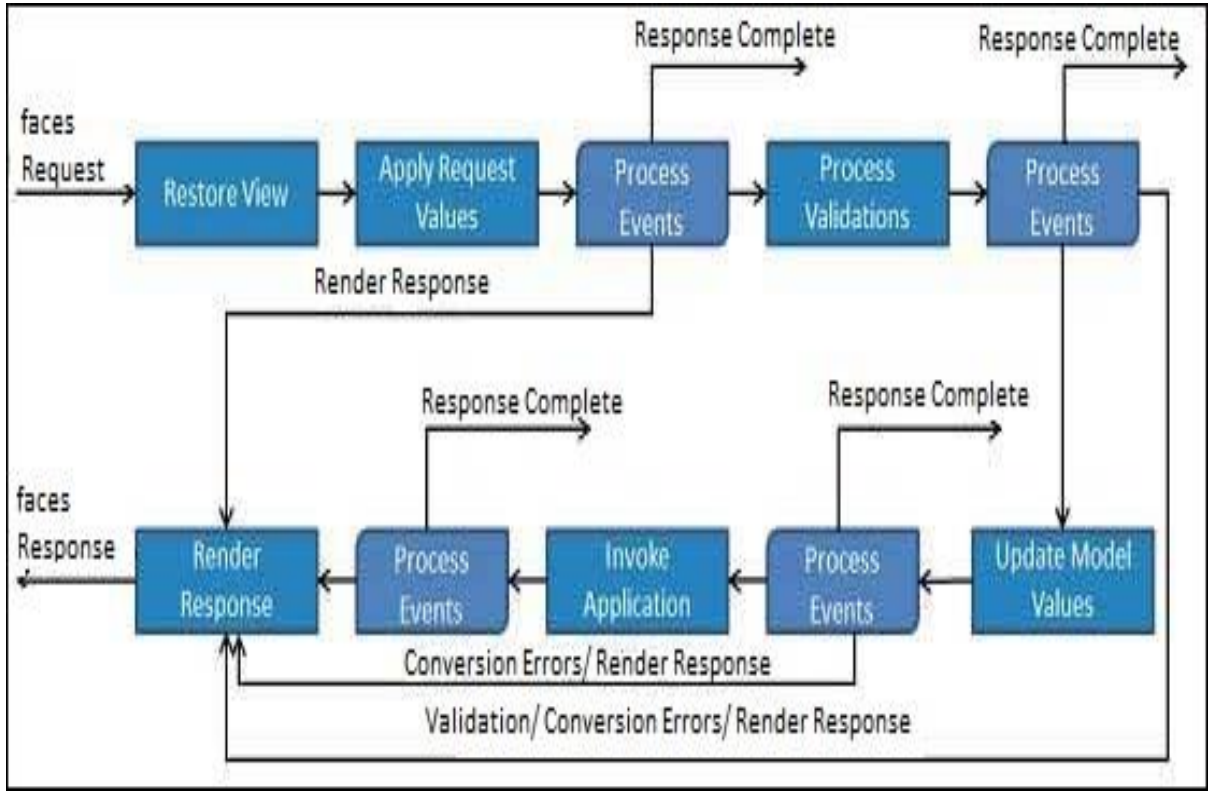
JSF LIFE CYCLE CONTINUED…

3. **Process validation :** During this phase, **JSF processes all validators** registered on the component tree.
4. **Update model values:** After the JSF checks that the data is valid, it walks over the component tree and **sets the corresponding server-side object properties** to the components local values.



JSF LIFE CYCLE CONTINUED…

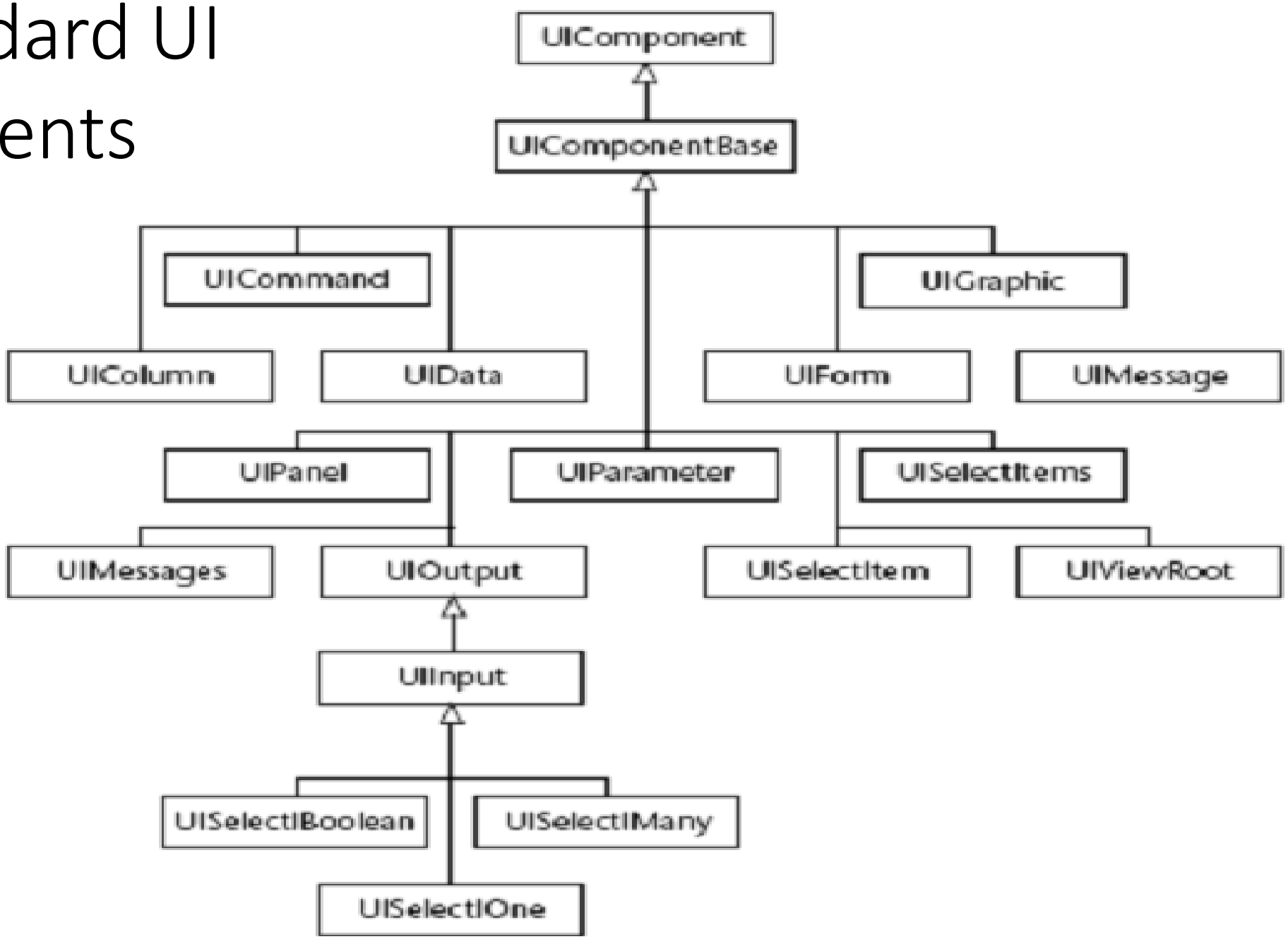
5. **Invoke application:** During this phase, JSF handles any application-level events, such as **submitting a form/linking to another page.**
6. **Render response:** During this phase, JSF asks container / application server to render the page if the application is using JSP pages. After the content is rendered.



JSF UI Component Model

- JSF provides the developers with the capability to create Web application from collections of UI components that can render themselves in different ways for multiple client types (for example - HTML browser, wireless, or WAP device).
- JSF provides
  - Core library
  - A set of base UI components - standard HTML input elements
  - Extension of the base UI components to create additional UI component libraries or to extend existing components
  - Multiple rendering capabilities that enable JSF UI components to render themselves differently depending on the client types

JSF standard UI components



Two types of JSF Tag library:

- **1.Core Tag Library:**
- Defines the tag that perform core actions.This tag library does not depend upon specific render kit.
- We can use these tag libraries in JSP page as follows:
- <%@taglib uri=" http://java.sun.com/jsf/core" prefix=" f"
- %>
- For these tags you need to use the following namespaces of URI in html node.
- <html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="http://java.sun.com/jsf/core" >

Two types of JSF Tag library (cont.):

- **2.Html Tag Library:**
- Defines the tags that represent general HTML UI components.
- These html components include:
- HtmlInputText, HtmlInputTextarea, HtmlForm and HtmlCommandButton.
- <%@taglib uri=" http://java.sun.com/jsf/html" prefix="h" %>
- JSF provides a standard HTML tag library. These tags get rendered into corresponding html output.
- For these tags you need to use the following namespaces of URI in html node.
- <html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" >

2.Html Tag Library:

- Defines the tags that represent general HTML UI components.These html components include HtmlInputText,HtmlInputTextarea,HtmlForm and HtmlCommandButton.

<%@taglib uri=" http://java.sun.com/jsf/html" prefix=" h" %>

- JSF provides a standard HTML tag library. These tags get rendered into corresponding html output.
- For these tags you need to use the following namespaces of URI in html node.
- <html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://java.sun.com/jsf/html" >

JSF STANDARD COMPONENT:

1. JSF <h:inputText>Tag
2. JSF <h:outputText> Tag
3. JSF <h:form> Tag
4. JSF <h:commandButton> Tag

1 JSF <h:inputText>Tag

- The JSF <h: inputText> tag is used to render an input field on the web page.
- It is used within a <h: form> tag to declare input field that allows user to input data.

<h:inputText id="sid-id" value="#{student.id}"/>

Attribute name	Description
Id	It is an identifier for this component. This id must be unique. You can use it to access HTML element in CSS and JS file.
Value	It is used to collect present value of the inputText.
maxlength	The maximum number of characters that may be entered in this field.
size	The number of characters used to determine the width of this field .

2 JSF - <h:selectOneMenu>

- The h:selectOneMenu tag renders an HTML input element of the type "select" with size not specified.

```
<h:selectOneMenu value = "#{student.city}">
        <f:selectItem itemValue = "1" itemLabel = "UNJHA" />
        <f:selectItem itemValue = "2" itemLabel = "MEHSANA" />
        <f:selectItem itemValue = "3" itemLabel = "KADI" />
        <f:selectItem itemValue = "4" itemLabel = "VISHNAGAR" />
</h:selectOneMenu>
```

3 JSF <h:outputText> Tag

- It is used to render a plain text.

<h:outputText value="hello"></h:outputText>

Attribute	Description
Value	It holds current value of this component.
Id	It is an identifier for this component. This id must be unique. You can use it to access HTML element in CSS and JS file.
style	It is used to apply CSS for the component.
class	It gives class name to the component. It is used to access component from CSS and JS file.
lang	It is used to specify language. It helps to make web page localized.

4 JSF <h:form> Tag

- The <h:form> tag represents an input form.
- It includes child components that can contain data which is either presented to the user or submitted with the form.
- It can also include HTML markup to lay out the components on the page.

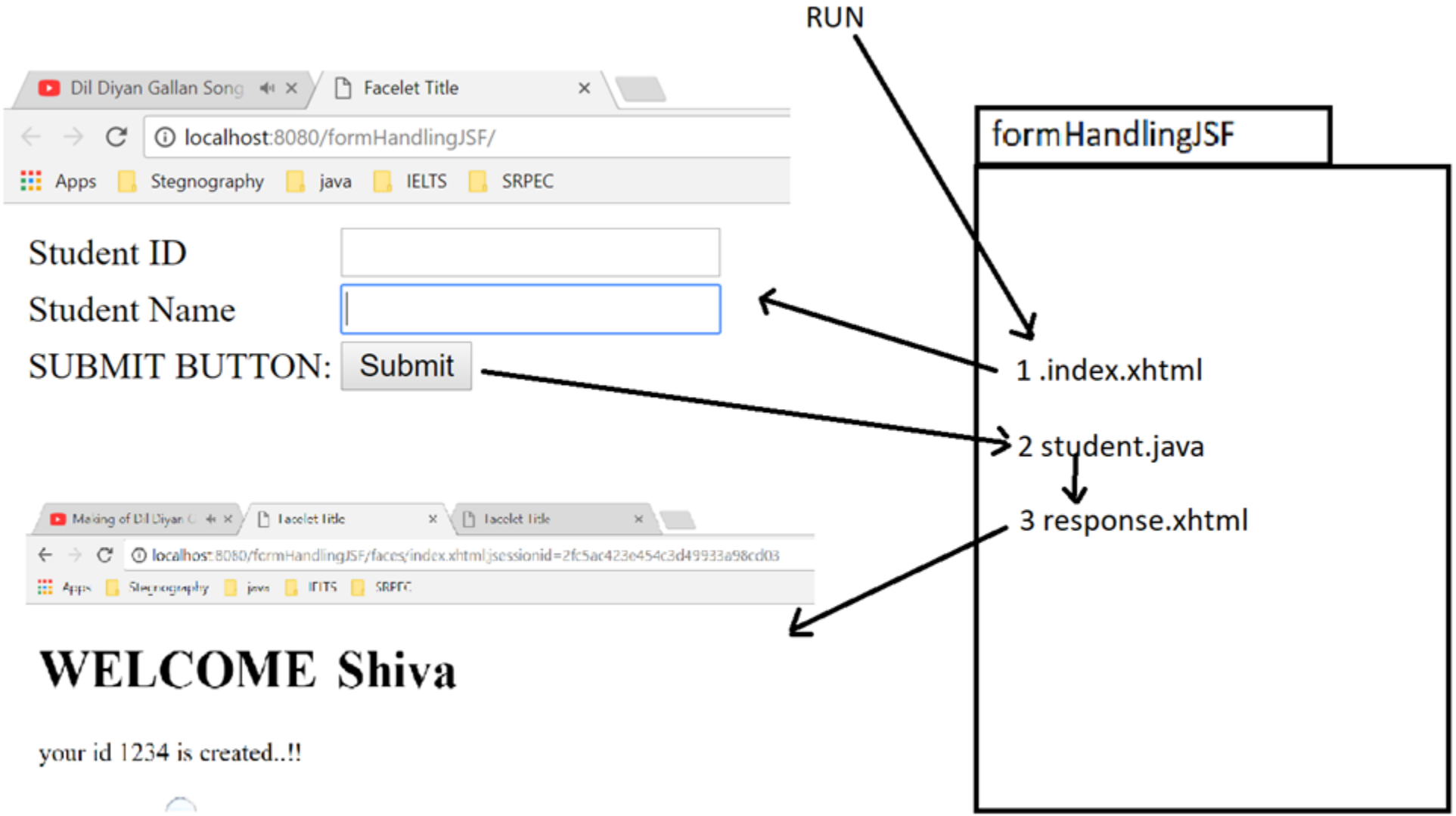
```
<h:form>
<!-- form elements -->
</h:form>
```

5 JSF <h:commandButton> Tag

- It creates a submit button and used to submit a application form.
- You can create it by using the following syntax.

```
<h:commandButton value="Submit“ action="response.xhtml">
</h:commandButton>
```

PROG 0 : JSF FORM HANDLING EXAMPLE:



PROG 0: FILES NEED TO BE CREATED

- Total 3 file need to be created
  1. index.xhtml
  2. Student.java
  3. response.xhtml

1. Index.xhtml

```
<?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">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h:form>
            <table>
                <tr>
                <td>Student ID</td>
                <td><h:inputText id="sid-id" value="#{student.id}"/></td>
                </tr>
```

1. Index.xhtml

```
            <tr>
            <td>Student Name</td>
            <td><h:inputText id="sname-id" value="#{student.name}"/></td>
            </tr>
            <tr>
            <td>SUBMIT BUTTON:</td>
            <td><h:commandButton value="Submit"
action="response.xhtml"></h:commandButton></td>
            </tr>
          </table>
       </h:form>
    </h:body>
</html>
```

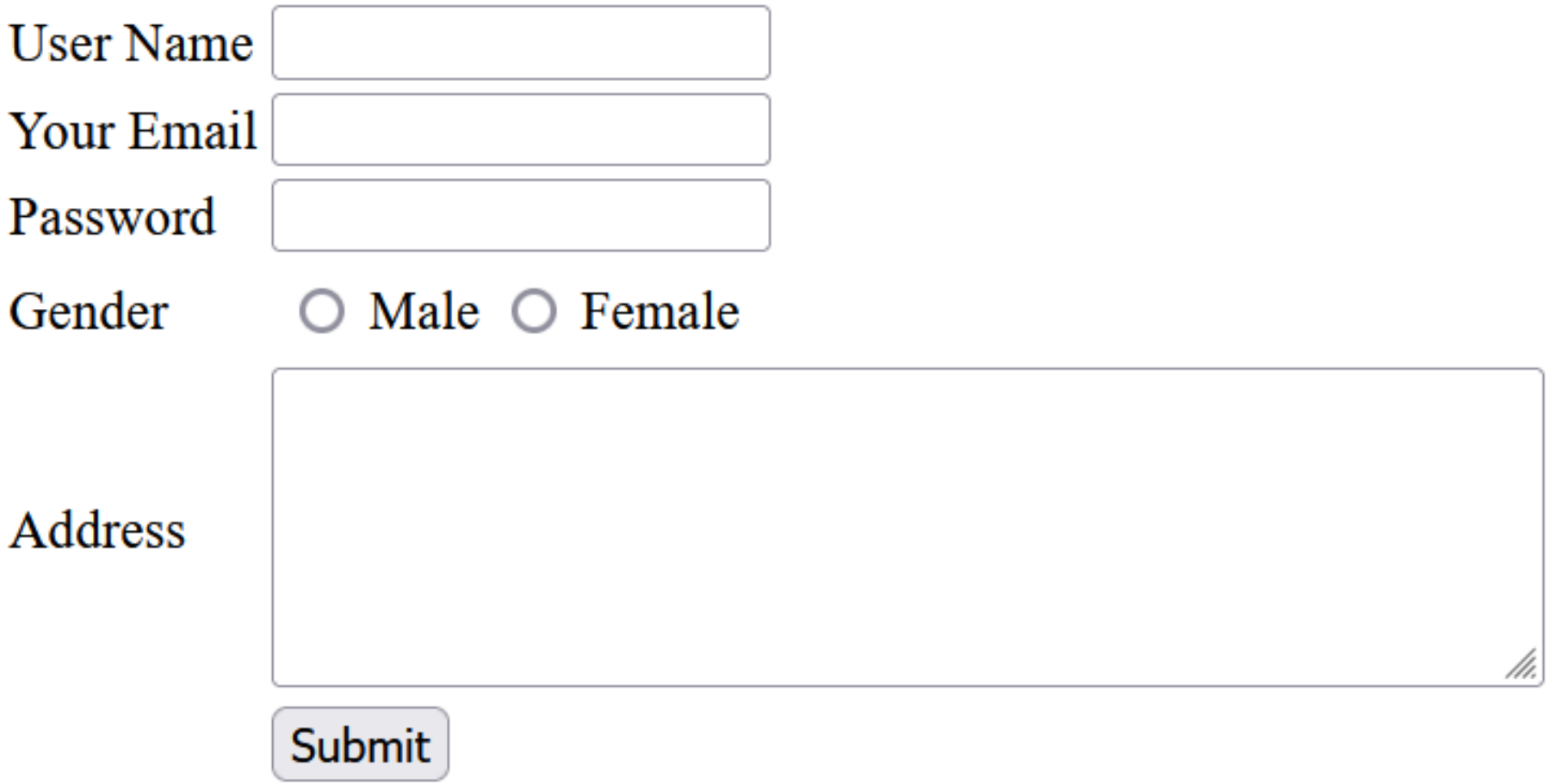
2. student.java
JSF MANAGED BEAN: JAVA CLASS

```
import javax.annotation.ManagedBean;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;

@Named("student")
@ManagedBean
@ApplicationScoped
public class student {
   int id;
   String name;
   public int getId() {
      return id;
 }
public void setId(int id)
   {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
}
```

3 response.xhtml

```
<?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">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h1> WELCOME <h:outputText value="#{student.name}"> </h:outputText></h1>

        your id <h:outputText value="#{student.id}"> </h:outputText> is created..!!
    </h:body>
</html>
```

PROG 1: JSF COMPONENTS



JSF EXPRESSION LANGUAGE:

- JSF expression language **allows the user to access the data dynamically form the java Bean component**.
- JSF provides rich expression language.
- We can provide normal operation using #{operation-expression} notation
- These expression can be of two types
- Property expression:

#{<beanName>.<property>}

- Method expression:

#{<beanName>.<method>}

JSF EXPRESSION LANGUAGE ADVANTAGES

- Can reference bean properties where bean can be an object stored in request, session or application scope or is a managed bean.
- Provides easy access to elements of a collection which can be a list, map or an array.
- Provides easy access to predefined objects such as a request.
- Arithmetic, logical and relational operations can be done using expression language.
- Automatic type conversion.
- Shows missing values as empty strings instead of NullPointerException.

JSF EXPRESSION LANGUAGE EXAMPLE:

UserData.java

```
import java.io.Serializable;
import java.util.Date;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "userData", eager = true)
@SessionScoped
public class UserData implements Serializable {
    private static final long serialVersionUID = 1L;
    private Date createTime = new Date();
    private String message = "Hello World!";

    public Date getCreateTime() {
        return(createTime);
    }

    public String getMessage() {
        return(message);
    }
}
```

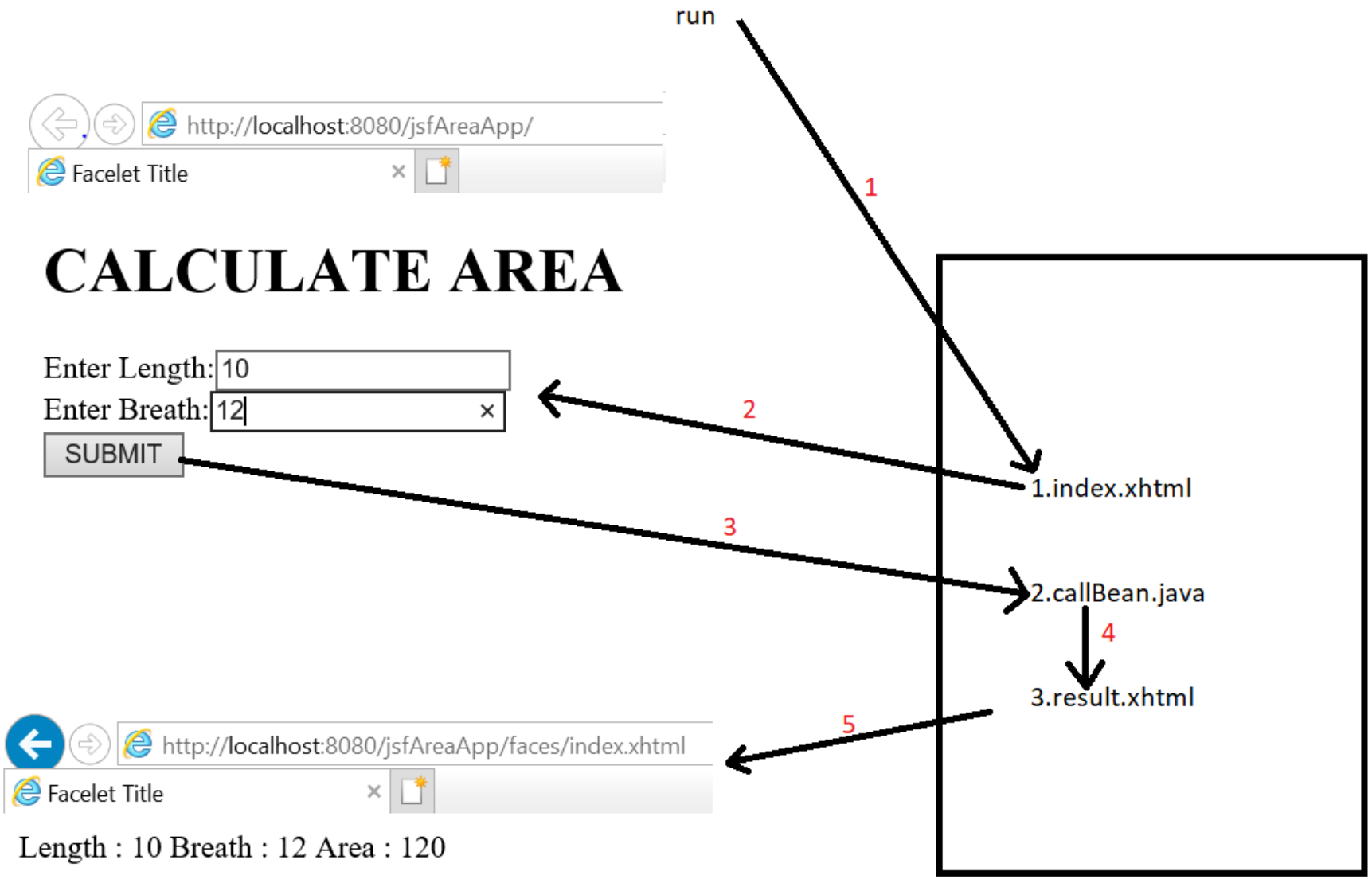
JSF EXPRESSION LANGUAGE EXAMPLE:

home.xhtml

```
<?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:f = "http://java.sun.com/jsf/core"
   xmlns:h = "http://java.sun.com/jsf/html">

   <h:head>
      <title>JSF Tutorial!</title>
   </h:head>

   <h:body>
      <h2>Expression Language Example</h2>
      Creation time:
      <h:outputText value = "#{userData.createTime}"/>
      <br/><br/>
      Message:
      <h:outputText value = "#{userData.message}"/>
   </h:body>
</html>
```

JSF EXPRESSION LANGUAGE EXAMPLE:

PROG 5:Create a JSF application that will calculate the area of rectangle. Make use of expression language.



Index.xhtml

```
<?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">
   <h:head>
      <title>Facelet Title</title>
   </h:head>
   <h:body>
      <h1>CALCULATE AREA</h1>
      <h:form>
         Enter Length:<h:inputText
value="#{calBean.len}"/><br/>
         Enter Breath:<h:inputText
value="#{calBean.br}"/><br/>
         <h:commandButton
value="SUBMIT" action="result.xhtml"/>
      </h:form>
   </h:body>
</html>
```

calBean.java

```
import javax.annotation.ManagedBean;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;

@Named(value = "calBean")
@ManagedBean
@ApplicationScoped

public class calBean {

    int len,br;
    public calBean() {
    }

    public int getLen() {
        return len;
    }

    public void setLen(int len) {
        this.len = len;
    }

    public int getBr() {
        return br;
    }

    public void setBr(int br) {
        this.br = br;
    }

    public int area(){
        int area = len*br;
        return area;
    }
}
```

Result.xhtml

```
<?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">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Length : #{calBean.len}
        Breath : #{calBean.br}
        Area : #{calBean.area()}
    </h:body>
</html>
```

JSF Convertor Tags

- JSF provides inbuilt convertors to convert its UI component's data to object used in a managed bean and vice versa. For example, these tags can convert a text into date object and can validate the format of input as well.
- For these tags you need to use the following namespaces of URI in html node.
- <html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="http://java.sun.com/jsf/core" >

JSF Convertor Tags

S.N.	Tag & Description
1	**f:convertNumber** Converts a String into a Number of desired format
2	**f:convertDateTime** Converts a String into a Date of desired format
3	**Custom Convertor** Creating a custom convertor

JSF Convertor Tags

1.<f:convertNumber >tag is used to convert a string value to a number of required format.

Syntax:

<f:convertNumber minFractionDigits="2" />

JSF Convertor Tags

S.N.	Attribute & Description
1	**type** number (default), currency , or percent
2	**pattern** Formatting pattern, as defined in java.text.DecimalFormat
3	**maxFractionDigits** Maximum number of digits in the fractional part
4	**minFractionDigits** Minimum number of digits in the fractional part
5	**maxIntegerDigits** Maximum number of digits in the integer part
6	**minIntegerDigits** Minimum number of digits in the integer part
7	**integerOnly** True if only the integer part is parsed (default: false)

JSF Convertor Tags

2.<f:convertDateTime> tag is used to convert a string value to a date of required format. It also acts as a validator a required date format.

Syntax:

<f:convertDateTime pattern="dd-mm-yyyy" />

JSF Convertor Tags

S.N.	Attribute & Description
1	**type** date (default), time, or both
2	**dateStyle** default, short, medium, long, or full
3	**timeStyle** default, short, medium, long, or full
4	**pattern** Formatting pattern, as defined in java.text.SimpleDateFormat
5	**locale** Locale whose preferences are to be used for parsing and formatting
6	**timeZone** Time zone to use for parsing and formatting

JSF - Validation Tags

- JSF provides inbuilt validators to validate its UI components. These tags can validate the length of the field, the type of input which can be a custom object.

S.No	Tag & Description
1	**f:validateLength**: Validates the length of a string
2	**f:validateLongRange**: Validates the range of a numeric value
3	**f:validateDoubleRange**: Validates the range of a float value

JSF - Validation Tags

S.N.	Tag & Description
1	**f:validateLength** Validates length of a string
2	**f:validateLongRange** Validates range of numeric value
3	**f:validateDoubleRange** Validates range of float value
4	**f:validateRegex** Validate JSF component with a given regular expression.
5	**Custom Validator** Creating a custom validat

JSF - f:validateLength

- f:validateLength tag is used to validate the length of a string value in a particular range.

JSF Tag

```
<f:validateLength minimum = "5"
maximum = "8" />
```

Tag Attributes

S.No	Attribute & Description
1	**minimum** A String with a minimum number of characters
2	**maximum** A String with a maximum number of characters

JSF - f:validateLongRange

- f:validateLongRange tag is used to validate the long value in a particular range.
- JSF Tag

```
<f:validateLongRange minimum = "5"
maximum = "200" />
```

- Tag Attributes

S.No	Attribute & Description
1	**minimum** Minimum long value within an optional range
2	**maximum** Maximum long value within an optional range

JSF - f:validateDoubleRange

- f:validateDoubleRange tag is used to validate a value to a range of float values.
- JSF Tag

```
<f:validateDoubleRange minimum = "1000.50"
maximum = "10000.50" />
```

- Tag Attributes

S.No	Attribute & Description
1	**minimum** Minimum long value within an optional range
2	**maximum** Maximum long value within an optional range

JSF - f:validateRegex

•**<f:validateRegex>** tag is used to validate a string value to a required format.

- Syntax:

```
<f:validateRegex pattern="((?=.*[a-z]).{6,})" />
```

Custom validator

- We can create our own Custom validator in JSF.
- Defining a custom validator in JSF is a three step process

Step No.	Description
1	Create a validator class by implementing *javax.faces.validator.Validator* interface.
2	Implement validate() method of above interface.
3	Use Annotation @FacesValidator to assign a unique id to the custom validator.

•Step 1: Create a validator class : UrlValidator.java

public class UrlValidator implements Validator { ... }

Step 2: Implement Validator interface methods : UrlValidator.java

public class UrlValidator implements Validator {

@Override public void validate(FacesContext facesContext, UIComponent component, String value) throws ValidatorException

{ ... }

}

- Step 3: Annotate to register the validator : UrlValidator.java

@FacesValidator("com.tutorialspoint.test.UrlValidator") public class UrlValidator implements Validator { }

Use the validator in JSF page

<h:inputText id="urlInput" value="#{userData.data}" label="URL" >
<f:validator validatorId="com.UrlValidator" />
</h:inputText>

JSF Validation Tags

- JavaServer Faces technology provides a set of standard classes and associated tags that you can use to validate elements data.

Validator class	Tag
BeanValidator	validateBean
DoubleRangeValidator	validateDoubleRange
LengthValidator	validateLength
LongRangeValidator	validateLongRange
RegexValidator	validateRegex
RequiredValidator	validateRequired

JSF Validation Tags (cont.)

Validator class	Tag	Function
BeanValidator	validateBean	It is used to registers a bean validator for the component.
DoubleRangeValidator	validateDoubleRange	It is used to check whether the local value of a component is within a certain range or not. The value must be floating-point or convertible to floating-point.
LengthValidator	validateLength	It is used to check whether the length of a component's local value is within a certain range or not. The value must be a java.lang.String.
LongRangeValidator	validateLongRange	It is used to check whether the local value of a component is within a certain range or not. The value must be any numeric type or String that can be converted to a long.
RegexValidator	validateRegex	It is used to check whether the local value of a component is a match against a regular expression from the java.util.regex package or not.
RequiredValidator	validateRequired	It is used to ensure that the local value is not empty on an EditableValueHolder component.

JSF FACELETS TAGS:

- JSF provides special tags to **create common layout** for a web application called facelets tags. These tags provide flexibility to manage common parts of multiple pages at one place.
- For these tags, you need to use the following **namespaces** of URI in html node

```
<html xmlns = "http://www.w3.org/1999/xhtml"
      xmlns:ui = "http://java.sun.com/jsf/facelets" >
```

JSF FACELETS TAGS:

- It is a light weight page declaration language which is used to build JavaServer Faces views using HTML style.
- It includes the following features:
- It uses XHTML for creating web pages.
- It supports Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries.
- It supports the Expression Language (EL).
- It uses templating for components and pages.

Advantages of Facelets

- It supports code reusability through templating and composite components.
- It provides functional extensibility of components and other server-side objects through customization.
- Faster compilation time.
- It validates expression language at compile-time.
- High-performance rendering.

FACELET TAGS

S.N.	Tag & Description
1	**Templates** We'll demonstrate how to use templates using following tags <ui:insert> <ui:define> <ui:include> <ui:composition>
2	**Parameters** We'll demonstrate how to pass parameters to a template file using following tag <ui:param>
3	**Custom** We'll demonstrate how to create custom tags.
4	**Remove** We'll demonstrate capability to remove JSF code from generated HTML page.

FOUR FACELET TAGS USED TO CREATE THE TEMPLATE:

S.No	Tag & Description
1	ui:insert Used in template file. It defines contents to be placed in a template. ui:define tag can replaced its contents.
2	ui:define Defines the contents to be inserted in a template.
3	ui:include Includes contents of one xhtml page into another xhtml page.
4	ui:composition Loads a template using template attribute. It can also define a group of components to be inserted in xhtml page.

Creating Template steps

Step 1: Create Header file: header.xhtml

- Use **ui:composition** tag to define a default content of Header section.

Code:

```
<ui:composition>
<h1>Default Header</h1>
 </ui:composition>
```

Step 2: Create Footer file: footer.xhtml

- Use **ui:composition** tag to define a default content of Footer section.

```
<ui:composition> <h1>Default Footer</h1>
</ui:composition>
```

Step 3: Create Content file: contents.xhtml

- Use **ui:composition** tag to define a default content of Content section.

```
<ui:composition> <h1>Default Contents</h1>
</ui:composition>
```

- Step 4: Create a Template: common.xhtml

Use **ui:insert** and **ui:include** tag to include header/footer and content file in template file. Name each section in **ui:insert** tag.

- **name** attribute of **ui:insert** tag will be used to replace contents of corresponding section.
- Code:

```
<h:body>
 <ui:insert name="header" >
 <ui:include src="header.xhtml" /> </ui:insert>
 <ui:insert name="content" >
 <ui:include src="contents.xhtml" />
 </ui:insert>
<ui:insert name="footer" >
<ui:include src="footer.xhtml" />
</ui:insert> </h:body>
```

2.Using ui:param tag, we can pass parameters to template file or an included file

Example:

```
<ui:insert name="header" >
 <ui:include src="/templates/header.xhtml" >
<ui:param name="defaultHeader"
value="Default Header" /> </ui:include>
</ui:insert>
```

3.ui:remove tag is used the prevent the JSF specific code to be rendered on client side. It is used especially to prevent commented out code to be rendered on client side.

Example:

```
<ui:remove>
 <h:commandButton value="Ok" />
</ui:remove>
```

JSF Event Handling

- When a user clicks a JSF button or link or changes any value in text field, JSF UI component fires event which will be handled by the the application code.
- To handle such event, event handler are to be registered in the application code or managed bean.
- When a UI component checks that a user event has happened, it creates an instance of the corresponding event class and adds it to an event list.
- Then, Component fires the event, i.e., checks the list of listeners for that event and call the event notification method on each listener or handler.
- JSF also provide system level event handlers which can be used to do some tasks when application starts or is stopping.

S.N.	Event Handlers & Description
1	**valueChangeListener** Value change events get fired when user make changes in input components.
2	**actionListener** Action events get fired when user clicks on a button or link component.
3	**Application Events** Events firing during JSF lifecycle: PostConstructApplicationEvent, PreDestroyApplicationEvent , PreRenderViewEvent.

Managed Beans

Managed Beans

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

Mapping Managed Beans

- Managed beans are mapped in the
- `faces-config.xml`

```
<managed-bean>
  <managed-bean-name>someName</managed-bean-name>
  <managed-bean-class>package.BeanClass</managed-bean-
    class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Mapping Elements

- `<managed-bean>` – enclosing element
- `<managed-bean-name>` – this element's value is the identifier used for the bean in our JSP pages
- `<managed-bean-class>` – the fully qualified name of the class of the bean
- `<managed-bean-scope>` – the bean's scope (request, session, application, none)

Binding Values

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

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

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

JSF Navigation Model

What Is Navigation?

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

Navigation Elements in `faces-config.xml`

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

Navigation Rules – Example

```
<navigation-rule>
  <from-view-id>/login-demo.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/welcome.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/login-demo.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>failed</from-outcome>
    <to-view-id>/login-failed.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Action Attribute in JSF Form

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

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

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

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

JSF Database Access

Program 6: DATABASE ACCESS

Index.xhtml

```
<?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">
   <h:head>
      <title>Facelet Title</title>
   </h:head>
   <h:body>
      <h1>JDBC JSF EXAMPLE</h1>
      <h:form>
         USER NAME: <h:inputText
id="uname“ value = "#{user.uname}"/>
         EMAIL: <h:inputText id="email“
value = "#{user.email}"/>
         <h:commandButton action =
"#{user.submit()}" value = "SUBMIT"/>
      </h:form>
   </h:body>
</html>
```

Program 6: DATABASE ACCESS

response.xhtml

```
<?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">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello <h:outputText value="#{user.uname}"/>
        Your Record has been Saved Successfully!
    </h:body>
</html>
```

Program 6: DATABASE ACCESS

User.java

```
import java.sql.*;
import javax.inject.Named;
import javax.annotation.ManagedBean;
import javax.enterprise.context.ApplicationScoped;

@Named(value = "user")
@ManagedBean
@ApplicationScoped
public class User {

    String uname,email;
    public User() {
    }
public String getUname() {
    return uname;
}
public void setUname(String uname) {
    this.uname = uname;
}
public String getEmail() {
    return email;
}
public void setEmail(String email) {
    this.email = email;
}
```

Program 6: DATABASE ACCESS

User.java

```
public boolean save(){
   int result = 0;
try{
   Class.forName("com.mysql.jdbc.Driver");
   Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/Emp","root","");
   PreparedStatement stmt =
con.prepareStatement("insert into user(name,email)
values(?,?)");
   stmt.setString(1, this.getUname());
   stmt.setString(2, this.getEmail());
   result = stmt.executeUpdate();
}catch(Exception e){
   System.out.println(e);
}
if(result == 1){
   return true;
}else
   return false;
}

public String submit(){
   if(this.save()){
      return "response.xhtml";
   }else{
      return "index.xhtml";
   }
}
}
```

JSF Database Access Example Links

- Example 1
- https://www.javatpoint.com/jsf-jdbc-connectivity

- Example 2 – CRUD Application
- https://www.javatpoint.com/jsf-crud-example

- Example 3
- https://www.tutorialspoint.com/jsf/jsf_display_datatable.htm

JSF PrimeFaces

- It is an UI (User Interface) library for JSF (JavaServer Faces) based applications.
- It is designed and developed by PrimeTek.
- It is Cross-platform, open source and written in Java programing language.
- It provides rich support of UI components, built-in ajax support, themes etc.
- It has became popular and supported by Oracle.
- It is default library in NetBeans IDE.

JSF PrimeFaces Features

- Rich UI Components
- Ajax Support
- Push Support
- Dialog Suppport
- Client Side Validation
- Mobile UI kit
- Skinning Framework

JSF PrimeFaces Features (cont.)

Rich UI Components

- It provides over 100 UI (User Interface) components. We can use that to create interective interface for JSF application. It includes HtmlEditor, Dialog, AutoComplete, Signature etc.

Ajax Support

- Primefaces provides built-in Ajax support. We can use it to perform Ajax call for the JSF application. It provides Ajax components like: counter, listener, event, poll etc.
- PrimeFaces Ajax Javascript API is powered by jQuery and optimized for JSF. Whole API consists of properly namespaced simple javascript functions **PrimeFaces.ajax.Request, PrimeFaces.ajax.Response.**

Push Support

- It provides Atmosphere framework that provides us push support. The Atmosphere Framework is the most popular asynchronous application development framework for enterprise Java. PrimeFaces Push 2.0 is based on Atmosphere as its predecessor and follows an annotation based approach this time.

JSF PrimeFaces Features (cont.)

Dialog Support

- PrimeFaces provides Dialog Framework which is used to open an external xhtml page in a dialog that is generated dynamically on runtime. The RequestContext provides methods to open and close dialog.

Client Side Validation

- PrimeFaces provides the most advanced Client Side Validation for JavaServer Faces and Java EE. It is used to validate data at client side. It is compatible with Server Side Implementation and provides Advanced Bean Validation Integration.

JSF PrimeFaces Features (cont.)

Mobile UI Kit

- It provides a mobile UI kit to create JSF application for mobile phones. It is default in the library. So, does not require any additional downloading. It is built on top of jQuery Mobile.
- It includes various features popular PrimeFaces components, ajax framework extensions, mobile ajax behavior events, integrated navigation model, lazy loading of pages, responsive widgets etc.

Skining Framework

- It provides lots of built-in themes and designer tools for visual themes. PrimeFaces is integrated with powerful ThemeRoller CSS Framework. Currently there are many pre-designed themes that we can preview and download from PrimeFaces theme gallery.

More on PrimeFaces

Link 1

- https://www.javatpoint.com/primefaces-configuration

Link 2

- https://www.javatpoint.com/primefaces-ajax