Developing Applications Using Java Web Frameworks

Session - 6

Expression Language, Facelets, and Data Table

Objectives

- Describe the features of JSF 1.2 and JSF 2.0
- Explain Expression language and its types in JSF
- Explain the use of facelets View in JSF 2.0
- Explain JSF 2.0 DataTables
- Explain JSF 2.0 Event Handling
- Explain the code to integrate JSF 2.0 with JDBC API

Introduction 1-4

- JSF provides enhanced UI components for development of View tier.
- JSF uses a powerful, component based UI development framework that simplifies J2EE Web development.
- JSF can also be viewed as MVC framework for:

Building Web forms

Validating the input

Invoking business logics from model

Displaying results

Introduction 2-4

- Some of the major features introduced in the JSF versions are:

JSF 1.2

- It introduced Unified Expression Language (EL).
- The new unified EL represents the union of JSP and JSF expression languages.
- The new EL is more pluggable and flexible.
- EL allows developers to use simple expressions to dynamically access data from JavaBeans components.

JSF 2.0

- JSF 2.0 introduced Facelets as the official view technology.
- Facelets allows the developers to create components using XML markup language rather than writing Java code.
- JSF 2.0 allows developers to use annotation for configuring the JSF components.
- This removes the need for `faces-config.xml` from the JSF application.
- Various annotations are provided for configuring navigation, managed bean, and page transition for the application.

Introduction 3-4

- The main features of JSF 2.0 are as follows:

Includes bookmarking for URLs.
Expands the existing lifecycle mechanism.
Gives more support for the AJAX requests.
Allows development of custom component with little or no Java coding.
Provides default exception handling mechanisms.
Provides a mechanism to access persistent store.
Eliminates the need to author a JSP tag handler when writing JSF components.
Easily creates CRUD-based applications.
Separates the 'build the tree' and 'render the tree' processes into two separate lifecycle phases.
Supports bundling and delivering static resources associated with images, style sheets, scripts, and so on.
Provides a mechanism to minimize the 'Lost Update' and 'Duplicate Button Press' problems.
Allows partial tree traversal during lifecycle execution with the help of Ajax.

Introduction 4-4

- Leverages annotations to declare components, managed beans, navigation rules, and so on to the runtime.
- Provides Date Picker, Tree, Tab View, File Upload components to the Standard HTML Render Kit.
- Streamlines the rendering process through caching.
- Improves the interceptor mechanism delivered through the Phase Listener feature.
- Specifies command line interface for authoring JSF applications.
- Allows JSF application resources access through REST.
- Enables components that publish events through RSS/Atom.
- Improves the UI component specification to increase the interoperability of UI component libraries from different vendors.
- Adds support for REST.
- Gives support for passing values from page to page.

Introduction to Expression Language

- JSF framework has created its own Expression Language (EL).
- **Need for EL in JSF**
  - UI components are evaluated in different phases when the Web page containing them is rendered for the first time.
  - Once the user enters the value in the UI components and submits the page, these values are converted, validated, and transmitted to server-side data objects.
  - The component events are processed.
  - JSF divides these activities into phases to handle these tasks.
  - Also, JSF components need to get data from the server-side objects during the rendering phase and set the data in the server-side objects during postback phase.
  - JSF components need to invoke methods during their various life cycle stages.

To satisfy the mentioned tasks of validating data and handling events, JSF developed a powerful EL.

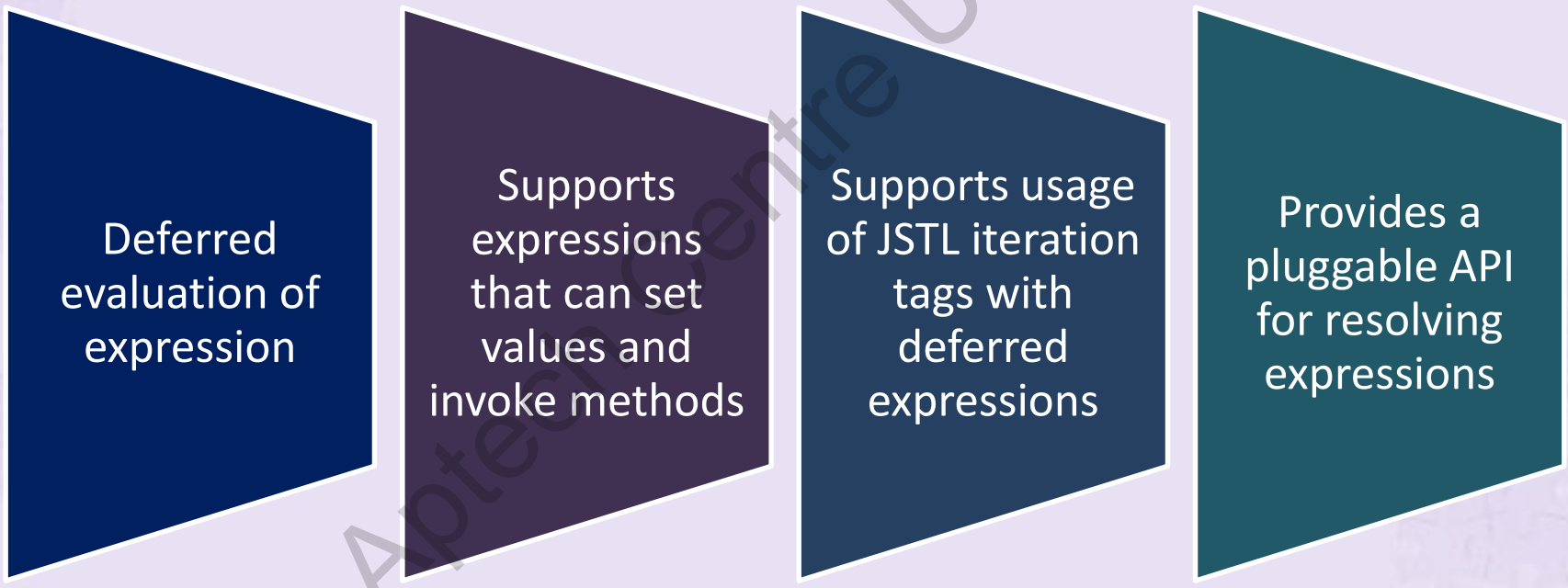
Expression Language in JSF 1-3

- The new unified EL introduced in JSF 1.2 is represented as follows:

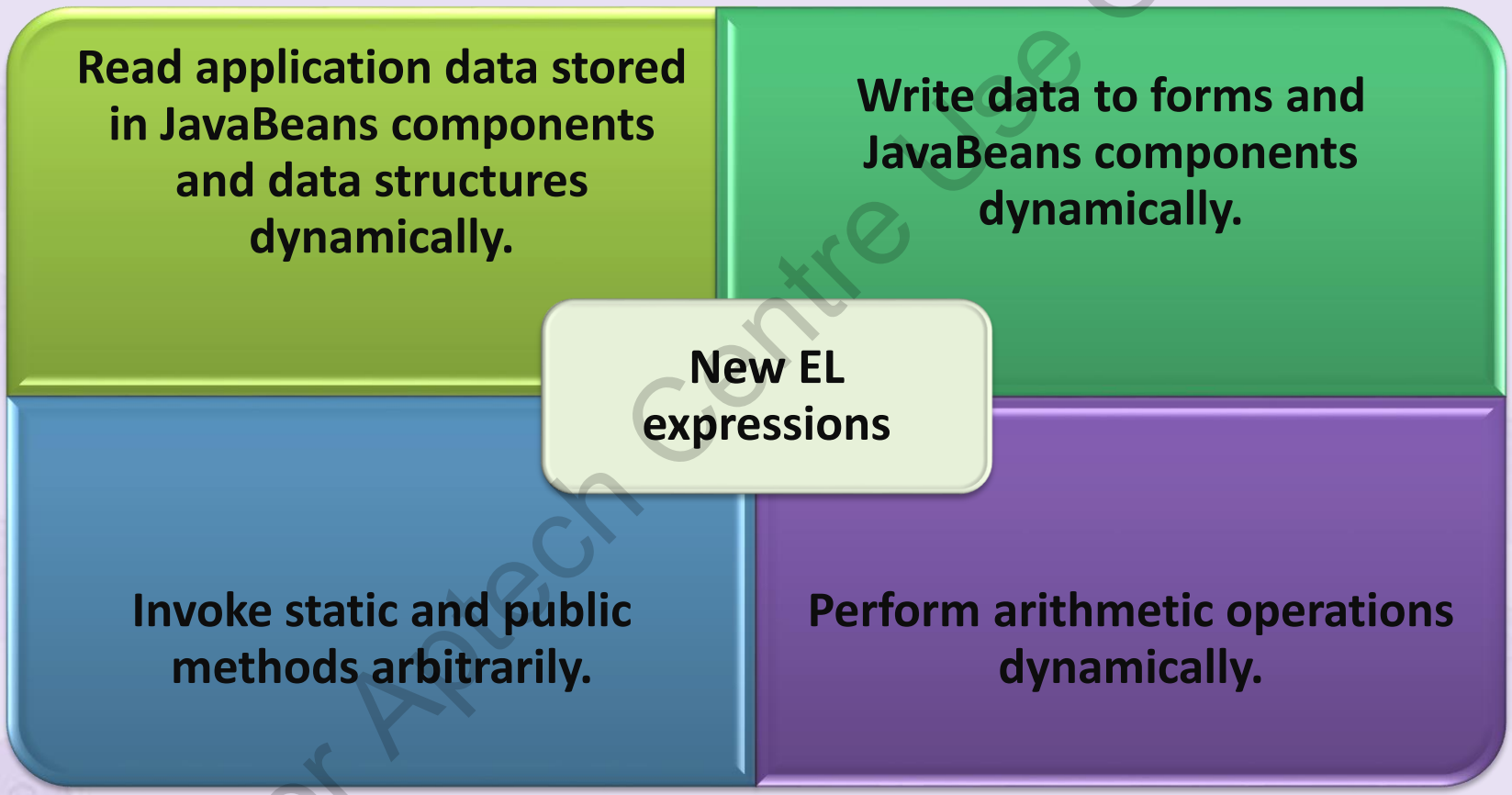
- JSF 2.0 EL allows developers to use simple expressions to dynamically access data from JavaBeans components.

Expression Language in JSF 2-3

- Following are the features of the new unified EL:

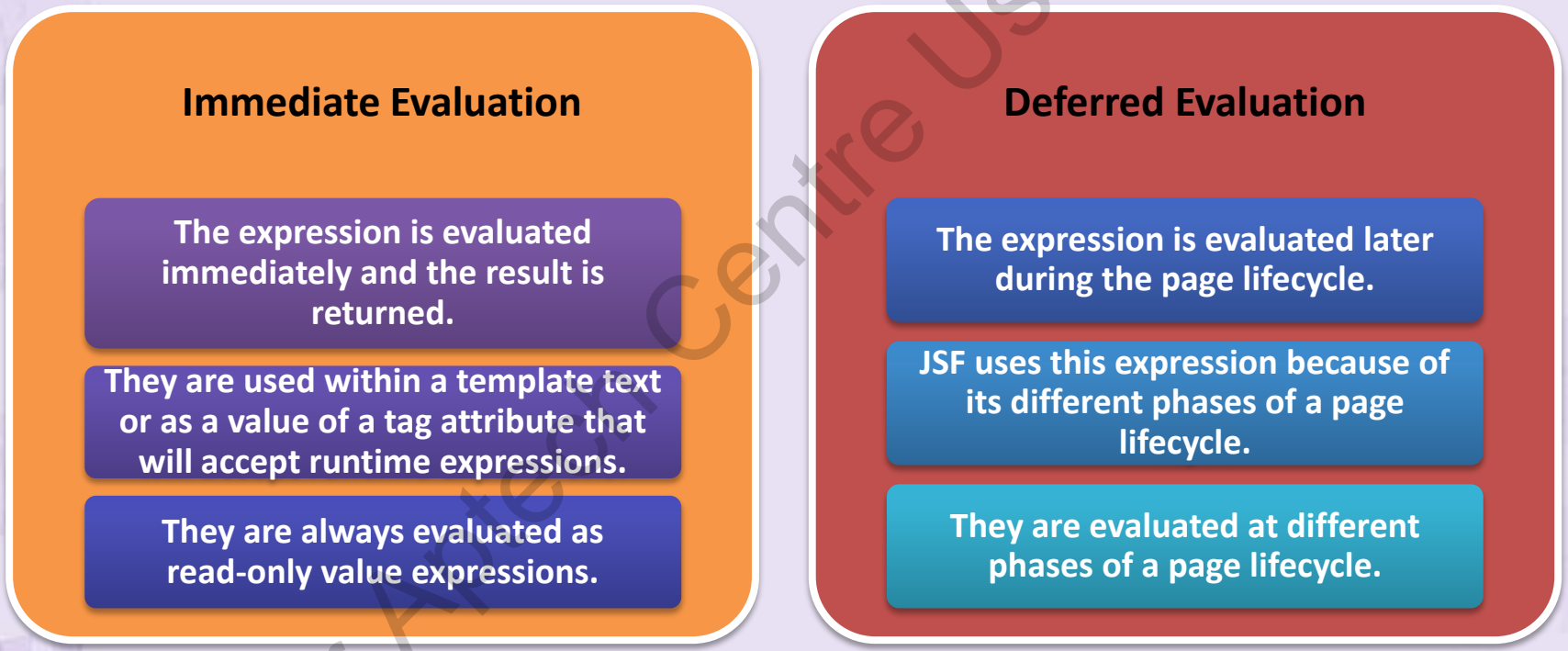


Expression Language in JSF 3-3



Immediate and Deferred Evaluation 1-2

- The new EL supports both immediate and deferred evaluation of expressions.



- The `${}` and `#{}` syntax are used to represent immediate and deferred evaluation of EL expression.

Immediate and Deferred Evaluation 2-2

- Following code snippet shows the use of deferred evaluation expression:

```
<h:inputText id="empname"
                              value="#{employee.name}" />
```

- When an initial request is made for the page containing this tag, JSF evaluates the expression `#{employee.name}` during the render-response phase of the lifecycle.

Types of EL Expressions 1-6

- Two types of expressions that the unified EL supports are value expressions and method expressions.

Value Expressions

- It refers to data present in a bean in the form of property or other data structure or as a literal value.
- It can be used to both read and write.
- It can either return a value or set a value.
- It allows to associate the name of the attribute or property with a value expression using the `setValueExpression()` method.
- It can be used to dynamically compute attributes and properties.

Types of EL Expressions 2-6

- Following code snippet shows the use of value expressions:

```
...
<h:outputText rendered="#{user.manager}"
              value="#{employee.salary}"/>
...
```

- Value expressions can be categorized into `rvalue` and `lvalue` expressions.
  - `rvalue` expressions can only read data and are evaluated using the `${}` delimiter.
  - `lvalue` expressions can both read as well as write data and evaluated using the `#{}` delimiter.

Types of EL Expressions 3-6

- Following code snippet shows the use of value expression:

```
...
<h:inputText value="#{employee.number}"/>
...
```

  - In the code:
    - The expression is evaluated as an `rvalue` when the page is rendered and the **getNumber** method present in the **employee** JavaBean is invoked.
    - The result is displayed as the default value in the text field.

Types of EL Expressions 4-6

Method Expressions

- It invokes an arbitrary public method of an arbitrary JavaBean object by passing a specified set of parameters.
- It renders the returned results on the page containing the expression.
- For example, the component tag uses method expressions to invoke methods that do some processing for the component.

Types of EL Expressions 5-6

- Following code snippet shows the use of method expressions:

```
<h:form>
<h:inputText
    id="name"
    value="#{employee.name}"
    validator="#{employee.validateName}"/>
<h:commandButton id="submit"
                 action="#{employee.submit}" />
</h:form>
```

- In the code, the `inputText` tag is a text field and the `validator` attribute of this tag invokes the method named, **`validateName`** in the **`employee`** JavaBean.

Types of EL Expressions 6-6

- Different ways in which the method expressions can be used in tag attributes are as follows:

Single Expression Construct	In this, the method expression is written as `<some:tagvalue="#{bean.method}" />`.
Text Only	In this, the method expression is written as `<some:tag value="sometext"/>`.

JSTL Tag 1-3

- Integration of JSTL tags provides `forEach` tag with JSF components.
- In JSF,
  - JSTL defines the attribute named, `items` of the `forEach` and `forTokens` tag.
  - Two cases for execution of attribute, **`items`**:

When runtime expression is specified

- Expression is evaluated immediately.

When a deferred value expression is specified

- The tag handler adds a mapping for the `var` attribute into an EL `VariableMapper` instance during each iteration.

JSTL Tag 2-3

- Following code snippet shows the use of `forEach` tag:

```
...
<table>
  <tr><th>Book Name</th> <th>Book Price</th>
  <th>Quantity</th></tr>
  <c:forEach var="book" books="#{shoppingCart.books}">
 <tr>
   <td><h:outputText value="#{book.name}" /></td>
   <td><h:outputText value="#{book.price}" /></td>
   <td><h:inputText value="#{book.quantity}" /></td>
 </tr>
</c:forEach>
<h:commandButton value="update quantities"
                 action="update" />
</table>
. . .
```

JSTL Tag 3-3

- Similar to iteration tags, the `set` tag now accepts deferred value expression.
- Following code snippet shows the implementation of the `set` tag:

```
<c:set var="d" value="#{handler.everythingDisabled}"/>
...
   <h:inputText id="i1" disabled="#{d}"/>
   <h:inputText id="i2" disabled="#{d}"/>
...
```

Facelets Declaration Language for JSF 2.0 1-3

- JSF 2.0 introduced facelets as a replacement for JSP as a View declaration language.

- Facelets:
  - Is a powerful lightweight page declaration language that is used to build JSF views using HTML style templates.
  - Are used to create XHTML-based views for the application.

Facelets Declaration Language for JSF 2.0 2-3

- Features of Facelets are as follows:

Provide a server-side template facility that allows the developer to compose a single View page out of several separate files.

Supports XHTML syntax for creating Web pages.

Provides an extended tag library.

Enforces clear separation of MVC by restricting the use of Java code in markup pages.

Code reusability that is achieved using templates and composite components.

Provides high performance rendering.

Reduces the time and effort required for development and deployment.

Facelets Declaration Language for JSF 2.0 3-3

- Following code snippet shows how to build a single logical view for the user:

```
<!-- This is the main page for the application -->
<f:view>
  <include name="menubar" file="menubar.xml" user="#{currentUser}"/>
  <include name="sidebar" file="sidebar.xml" user="#{currentUser}"/>
  <include name="summary" file="summary.xml" user="#{currentUser}"/>
</f:view>
```

- The code:
  - Creates a single page by combining three parts: **menubar**, **sidebar**, and **summary**.
  - The URI, `http://java.sun.com/jsf/facelets` is the JSF tag library that contains templating tags for the Facelets.
  - The EL expressions are used to reference properties and methods of managed beans, bind component objects, or to assign values to methods or properties of managed beans.

Facelet Tags 1-2

- Facelet tags are used to create a page that acts as the base or template for the other pages in the JSF application.
- Following table shows the Facelets tags that are used to perform templating in JSF:

Tag	Syntax	Description
`ui:composition`	`<ui:composition template="optionalTemplate">`	• It is used in files that are acting as a template client. • It is used to enable templating in Facelets. • It adds the component as the direct child of the `UIViewRoot`.
`ui:decorate`	`<ui:decorate template="requiredTemplate">`	• The tag provides features similar to `ui:composition`. • It also includes the content surrounding the `<ui:decorate>` tag. • The template attribute is required for this tag.

Facelet Tags 2-2

Tag	Syntax	Description
`ui:define`	`<ui:define name="requiredName">`	• The tag defines the content that is inserted into a page by a template. • It is used inside the `ui:composition` tag. • It defines the region that will be inserted at the location specified with the `ui:insert` tag.
`ui:insert`	`<ui:insert name="optionalName">`	• The tag inserts content into a template. • It is used in the template file and the `name` attribute specifies the location where the template client needs to be inserted.
`ui:include`	`<ui:include src="requiredFilename">`	• The tag `ui:include` is used to combine and reuse content for multiple pages.
`Ui:remove`	`<ui:remove>`	• The tag is mainly used during development to 'comment out' a portion of the markup.
`Ui:debug`	`<ui:debug hotkey="optionalHotKey" />`	• The tag enables a hot key that pop ups a new window displaying the component tree.

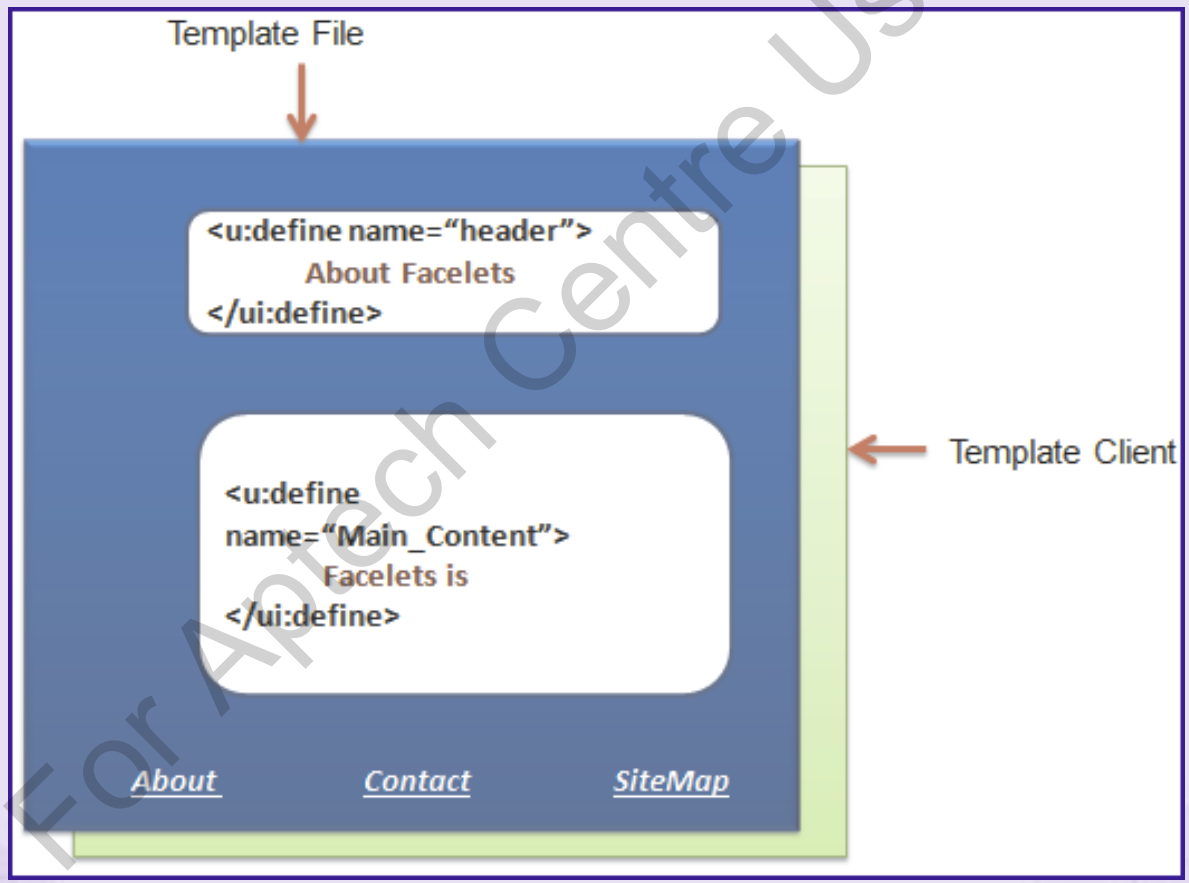
Templating with Facelets 1-6

- The facelet template consists of two main files such as template file and template client file.

Template file	• This file corresponds to a view Id, such as `greeting.xhtml`.
Template Client file	• A template client uses one or more developed template to achieve reuse of the page content.

Templating with Facelets 2-6

- Following figure shows Facelet view with template and template client:



Templating with Facelets 3-6

- Following code snippet shows a template that defines the structure for a page:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">
    <h:head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
        <h:outputStylesheet library="css" name="default.css"/>
        <h:outputStylesheet library="css" name="cssLayout.css"/>
        <title>Facelets Template</title>
    </h:head>
    <h:body>
        <div id="top" class="top">
            <ui:insert name="top">Top Section</ui:insert>
        </div>
        <div>
```

Templating with Facelets 4-6

```
      <div id="left">
            <ui:insert name="left">Left Section</ui:insert>
        </div>

        <div id="content" class="left_content">
            <ui:insert name="content">Main Content</ui:insert>
        </div>
        </div>
    </h:body>
</html>
```

- The code defines an XHTML page that is divided into three sections: a top, a left, and a main section.
- The `ui:insert` tag is used to define a default structure for a page.

Templating with Facelets 5-6

- Following code snippet shows the client page that invokes the template:

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

    <h:body>
        <ui:composition template="./template.xhtml">
            <ui:define name="top">
                Welcome to Template Client Page
             </ui:define>
```

Templating with Facelets 6-6

```
            <ui:define name="left">
                <h:outputLabel value="You are in the Left Section"/>
             </ui:define>

             <ui:define name="content">
                <h:graphicImage
value="#{resource['images:wave.med.gif']}"/>
                <h:outputText value="You are in the Main Content
Section"/>
             </ui:define>
        </ui:composition>
    </h:body>
</html>
```

- The client invokes the template by using the `ui:composition` tag.
- The client page invokes the template page and inserts the content using the `ui:define` tag.

Introduction to JSF 2.0 DataTable 1-7

- JSF 2.0 DataTable helps to render and format html tables.

- **DataTable**
  - Iterates over an array of values to display data.
  - Provides attributes to modify the data.
- Following code snippet shows the tag library to be used for DataTable:

```
<html
xmlns="http://www.xyz.org/2014/xhtml"
xmlns:h="http://java.sun.com/jsf/html">
</html>
```

Introduction to JSF 2.0 DataTable 2-7

- The most important DataTable operations in JSF 2.0 are:
  - **Display DataTable:** It displays a datatable.
  - **Add data:** It adds a new row in a datatable
  - **Edit data:** It edits a row in a datatable.
  - **Delete data:** It deletes a row in datatable.
  - **Using Data Model:** It displays row numbers in a datatable.

- **Attributes of DataTable**
  - The main attribute at the `h:dataTable` is the `value` attribute.
  - `value` attribute represents the data over which `h:dataTable` iterates.

Introduction to JSF 2.0 DataTable 3-7

- The data must be one of the following types:

A Java Object
An Array
An instance of `java.util.List`
An instance of `java.sql.ResultSet`
An instance of `javax.servlet.jsp.jstl.sql.Result`
An instance of `javax.faces.model.DataModel`

Introduction to JSF 2.0 DataTable 4-7

- Following code snippet shows the use of `h:dataTable` tag to loop over the array of `order` object:

```
<h:head>
<h:outputStylesheet library="css" name="table-style.css"  />
</h:head>
<h:body>

<h1>JSF 2 dataTable example</h1>

<h:dataTable value="#{order.orderList}" var="o"
             styleClass="order-table"
             headerClass="order-table-header" >

<h:column>
<!-- column header -->
<f:facet name="header">Order No</f:facet>
<!-- row record -->
#{o.orderNo}
</h:column>
```

Introduction to JSF 2.0 DataTable 5-7

```
<h:column>
<f:facet name="header">Product Name</f:facet>
#{o.productName}
</h:column>

<h:column>
<f:facet name="header">Price</f:facet>
#{o.price}
</h:column>

<h:column>
<f:facet name="header">Quantity</f:facet>
#{o.qty}
</h:column>
 </h:dataTable>
 </h:body>
</html>
```

Introduction to JSF 2.0 DataTable 6-7

- Following code snippet shows the `table-style.css` page:

```
.order-table{
border-collapse:collapse;
border:1px solid #000000;
}
.order-table-header{
text-align:center;
background:none repeat scroll 0 0 #E45EA5;
border-bottom:1px solid #000000;
padding:3px;
}
```

- The `table-style.css` page contains the properties to change the appearance of the data table displayed on the page.

Introduction to JSF 2.0 DataTable 7-7

- Following code snippet shows the `OrderBean` that will create a list of orders to be displayed in the data table component:

```
@ManagedBean(name = "orderbean", eager = true)
@SessionScoped
public class OrderBean implements Serializable {
  private static final ArrayList<Order> orderList
         = new ArrayList<Order>(Arrays.asList(
          new Employee("10", "Beverages", 30,5),
          new Employee("20", "Stationary", 35,3),
          new Employee("30", "Appliances", 25,25)
)) ;
.  . .
}
```

- The code creates a managed bean named, `OrderBean` and initializes an `ArrayList` object with the instances of `Order` class.
- The data from the `Order` are displayed in the data table.

Introduction to JSF 2.0 Event Handling 1-2

- JSF event handling is based on the JavaBeans event model.
- JavaBeans Event Model is based on:
  - Event classes
  - Event listener interfaces
- Some of the examples of events in an application includes:
  - Clicking a button.
  - Selecting an item from a menu or list.
  - Changing a value in an input field.

Introduction to JSF 2.0 Event Handling 2-2

- The important event handlers in JSF 2.0 are as follows:

ValueChangeListener	ActionListener	Application Events
• Value change events is invoked when user make changes in input components.	• Action event is invoked when user clicks a button or link component.	• Application is invoked during JSF lifecycle: `PostConstructApplicationEvent`, `PreDestroyApplicationEvent`, `PreRenderViewEvent`.

Integrating JDBC with JSF 2.0 1-3

- Following code snippet shows the integration of JSF 2.0:

```
@ManagedBean(name = "customerData")
@SessionScoped
public class CustomerData implements Serializable {
public List<Author> getCustomers(){
ResultSet rs = null;
PreparedStatement pst = null;
Connection con = getConnection();
String stm = "Select * from customers";
List<Customer> records = new ArrayList<Customer>();
t r y {
pst = con.prepareStatement(stm);
pst.execute();
rs = pst.getResultSet();
while(rs.next()){
Customer customer = new Customer();
author.setId(rs.getInt(1));
```

Integrating JDBC with JSF 2.0 2-3

```
author.setName(rs.getString(2));
records.add(customer);
}
} catch (SQLException e) {
e.printStackTrace();
}
return records;
}
public Connection getConnection(){
Connection con = null;
String url = "jdbc:sqlserver://localhost/testdb";
String user = "user1";
String password = "user1";
t r y {
con = DriverManager.getConnection(url, user, password);
System.out.println("Connection completed.");
} catch (SQLException ex) {
   . . .
}
}
```

Integrating JDBC with JSF 2.0 3-3

- Following code snippet shows the `cust.xhtml` page accessing customer data in a `DataTable` element:

```
<html
. . .
<h2>JDBC Integration</h2>
<h:dataTable value="#{customerData.customers}" var="c"
styleClass="CustomerTable"
headerClass="CustomerTableHeader" >
<h:column><f:facet name="header">Customer ID</f:facet>
     #{c.i d }
</h:column>
<h:column><f:facet name="header">Name</f:facet>
    #{c.name}
</h:column>
</h:dataTable>
</h:body>
</html>
```

Summary

- JSF 2.0 Expression Language helps to access the JavaBeans component in the JSF Web application.
- JSF helps to construct a UI from a set of reusable UI components and simplifies migration of application data to and from the UI.
- JSF permits custom UI components to be built and reused easily and gives an easy model for wiring client-generated events to server-side application code.
- The JSF 2.0 DataTable renders and formats html tables and iterates over an array of values to display data.
- JSF 2.0 Event Handling manages the events generated by components and is based on the JavaBeans event model.
- JSF 2.0 annotations are used in managed beans, registering listeners, resource rendering, and so on.