Web Applications Developments

ITWT413

LECTURE 9: JAKARTA EE WEB PROFILE

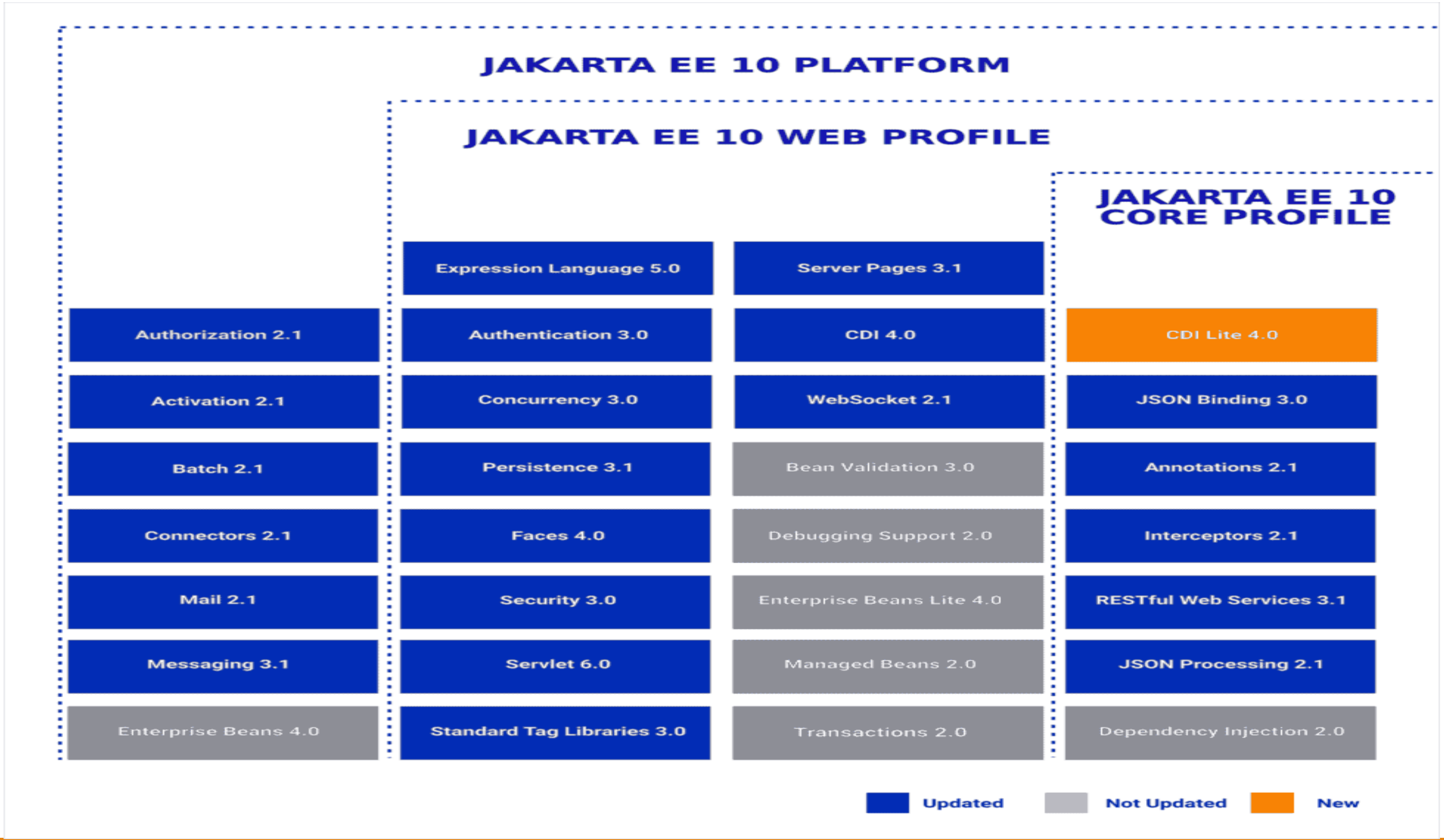
JAKARTA FACES TECHNOLOGY

DR. HALA SHAARI

Jakarta EE Tutorial- Structure

Basic Platform	Jakarta EE Core Profile	Jakarta EE Web Profile	Jakarta EE Platform
☐ Resource Creation	☐ Jakarta CDI Lite	☐ Jakarta CDI Full	☐ Jakarta Mail
☐ Injection	☐ Jakarta REST	☐ Jakarta Validation	☐ Jakarta Messaging
☐ Packaging	☐ Jakarta JSON	☐ Jakarta Security	☐ Jakarta Batch
		☐ Jakarta Servlets	
		☐ Jakarta Faces	
		☐ Jakarta WebSocket	
		☐ Jakarta Persistence	
		☐ Jakarta Enterprise Beans Lite	

Jakarta EE 10



Lecture Agenda

- ❑Introduction to Jakarta Faces Technology
- ❑Introduction to Facelets

Introduction to Jakarta Faces Technology

Jakarta Faces technology consists of the following:

- An API for representing components and managing their state; handling events, server-side validation, and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features
- Tag libraries for adding components to web pages and for connecting components to server-side objects

Jakarta Faces Technology

Jakarta Faces technology provides a well-defined programming model and various tag libraries. The tag libraries contain tag handlers that implement the component tags. These features significantly ease the burden of building and maintaining web applications with server-side user interfaces (UIs). With minimal effort, you can complete the following tasks.

- Create a web page.
- Drop components onto a web page by adding component tags.
- Bind components on a page to server-side data.
- Wire component-generated events to server-side application code.
- Save and restore application state beyond the life of server requests.
- Reuse and extend components through customization.

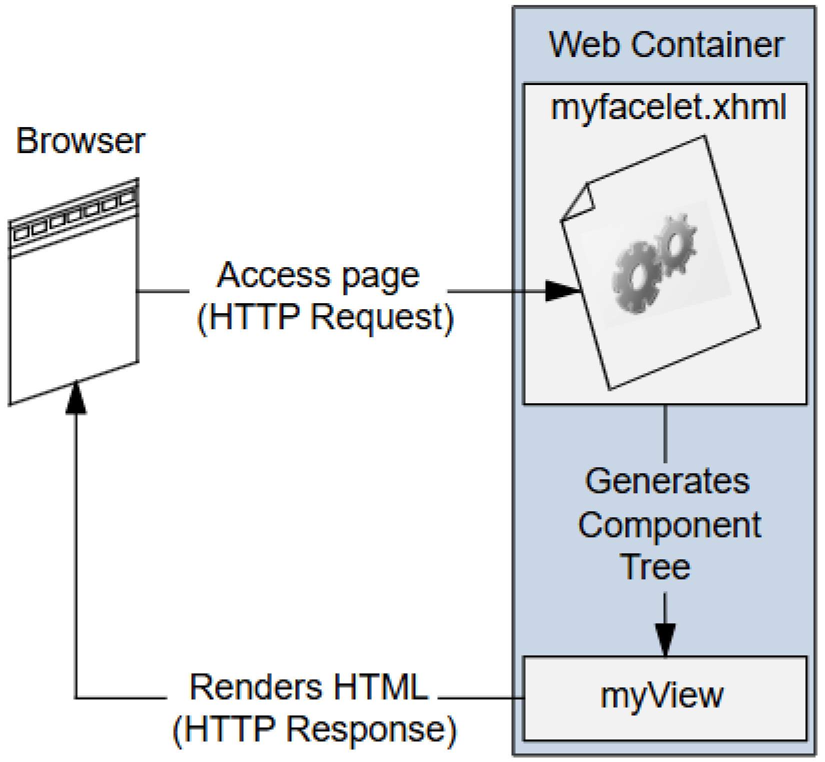
What Is a Jakarta Faces Application?

A typical Jakarta Faces application includes the following parts.

- A set of web pages in which components are laid out.
- A set of tags to add components to the web page.
- A set of managed beans, which are lightweight, container-managed objects (POJOs). In a Jakarta Faces application, managed beans serve as backing beans, which define properties and functions for UI components on a page.
- A web deployment descriptor (web.xml file).
- Optionally, one or more application configuration resource files, such as a faces-config.xml file, which can be used to define page navigation rules and configure beans and other custom objects, such as custom components.
- Optionally, a set of custom objects, which can include custom components, validators, converters, or listeners, created by the application developer.
- Optionally, a set of custom tags for representing custom objects on the page.

Responding to a Client Request for a Jakarta Faces Page

- The web page, myfacelet.xhtml, is built using Jakarta Faces component tags. Component tags are used to add components to the view (represented by myView in the diagram), which is the server-side representation of the page. In addition to components, the web page can also reference objects, such as the following:
  - Any event listeners, validators, and converters that are registered on the components
  - The JavaBeans components that capture the data and process the application-specific functionality of the components
- On request from the client, the view is rendered as a response. Rendering is the process whereby, based on the server-side view, the web container generates output, such as HTML or XHTML, that can be read by the client, such as a browser.



Jakarta Faces Technology Benefits

Facelets technology offers several advantages.

- Code can be reused and extended for components through the templating and composite component features.
- You can use annotations to automatically register the managed bean as a resource available for Jakarta Faces applications.
- Most important, Jakarta Faces technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.

User Interface Component Model

- In addition to the lifecycle description, an overview of Jakarta Faces architecture provides better understanding of the technology.
- Jakarta Faces components are the building blocks of a Jakarta Faces view. A component can be a user interface (UI) component or a non-UI component.
- Jakarta Faces UI components are configurable, reusable elements that compose the user interfaces of Jakarta Faces applications. A component can be simple, such as a button, or can be compound, such as a table composed of multiple components.

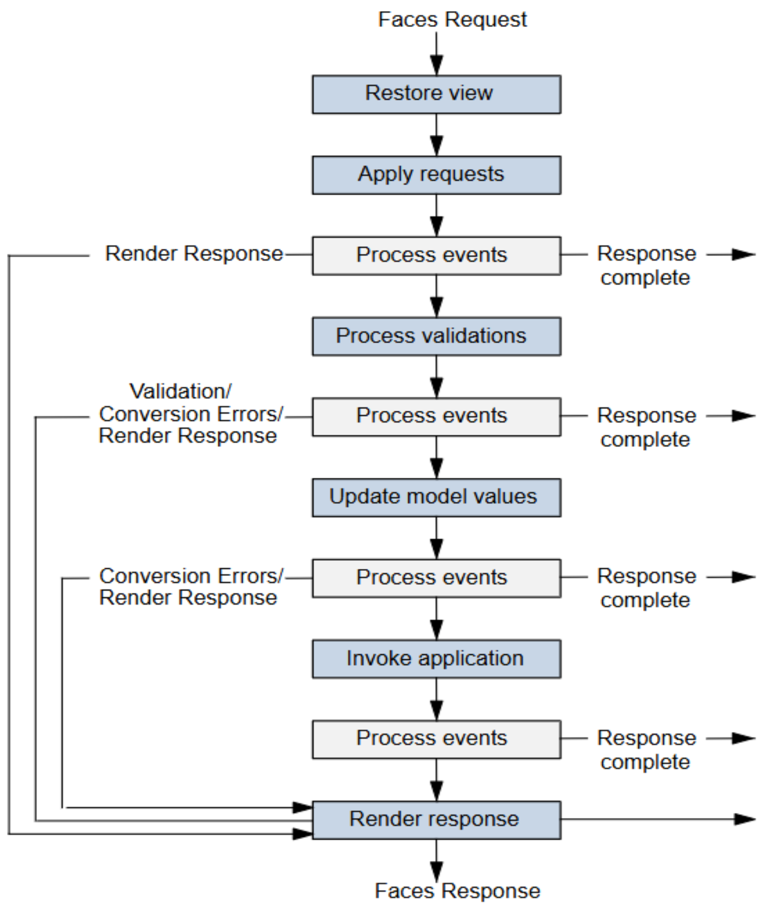
User Interface Component

- Jakarta Faces technology provides a rich, flexible component architecture that includes the following:
  - A set of jakarta.faces.component.UIComponent classes for specifying the state and behavior of UI components
  - A rendering model that defines how to render the components in various ways
  - A conversion model that defines how to register data converters onto a component
  - An event and listener model that defines how to handle component events
  - A validation model that defines how to register validators onto a component

The Lifecycle of a Jakarta Faces Application

- The lifecycle of an application refers to the various stages of processing of that application, from its initiation to its conclusion. All applications have lifecycles. During a web application lifecycle, common tasks are performed, including the following.
  - Handling incoming requests
  - Decoding parameters
  - Modifying and saving state
  - Rendering web pages to the browser

Overview of the Jakarta Faces Lifecycle



What Is Facelets?

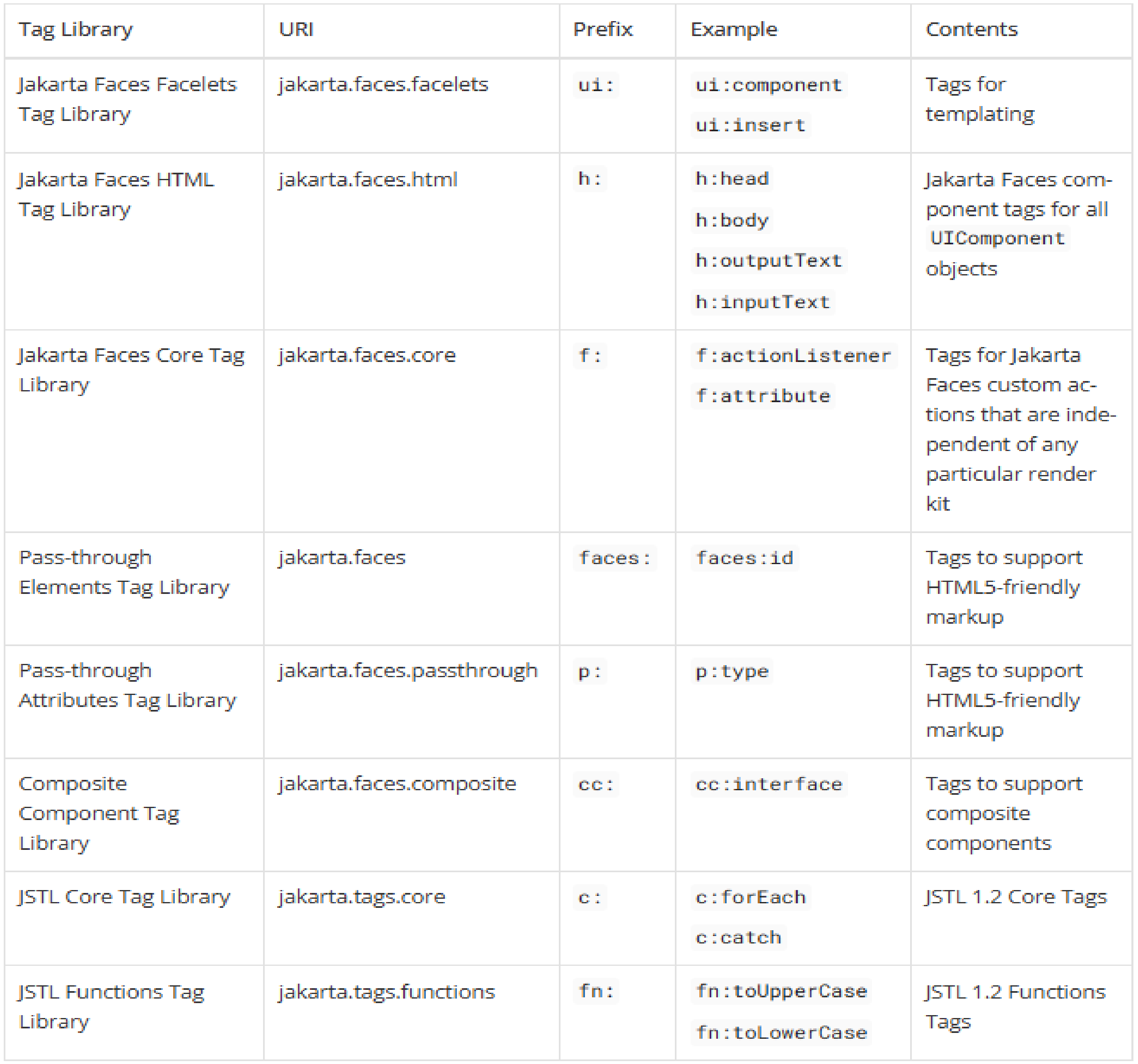
- Facelets is a powerful but lightweight page declaration language that is used to build Jakarta Faces views using HTML style templates and to build component trees. Facelets features include the following:
  - Use of XHTML for creating web pages
  - Support for Facelets tag libraries in addition to Jakarta Faces and JSTL(JSP - Standard Tag Library) tag libraries
  - Support for the Expression Language (EL)
  - Templating for components and pages

Advantages of Facelets

- The advantages of Facelets for large-scale development projects include the following:
  - Support for code reuse through templating and composite components
  - Functional extensibility of components and other server-side objects through customization
  - Faster compilation time
  - Compile-time EL validation
  - High-performance rendering

In short, the use of Facelets reduces the time and effort that needs to be spent on development and deployment.

Tag Libraries Supported by Facelets



Tag Library	URI	Prefix	Example	Contents
Jakarta Faces Facelets Tag Library	jakarta.faces.facelets	`ui:`	`ui:component` `ui:insert`	Tags for templating
Jakarta Faces HTML Tag Library	jakarta.faces.html	`h:`	`h:head` `h:body` `h:outputText` `h:inputText`	Jakarta Faces com-ponent tags for all `UIComponent` objects
Jakarta Faces Core Tag Library	jakarta.faces.core	`f:`	`f:actionListener` `f:attribute`	Tags for Jakarta Faces custom ac-tions that are inde-pendent of any particular render kit
Pass-through Elements Tag Library	jakarta.faces	`faces:`	`faces:id`	Tags to support HTML5-friendly markup
Pass-through Attributes Tag Library	jakarta.faces.passthrough	`p:`	`p:type`	Tags to support HTML5-friendly markup
Composite Component Tag Library	jakarta.faces.composite	`cc:`	`cc:interface`	Tags to support composite components
JSTL Core Tag Library	jakarta.tags.core	`c:`	`c:forEach` `c:catch`	JSTL 1.2 Core Tags
JSTL Functions Tag Library	jakarta.tags.functions	`fn:`	`fn:toUpperCase` `fn:toLowerCase`	JSTL 1.2 Functions Tags

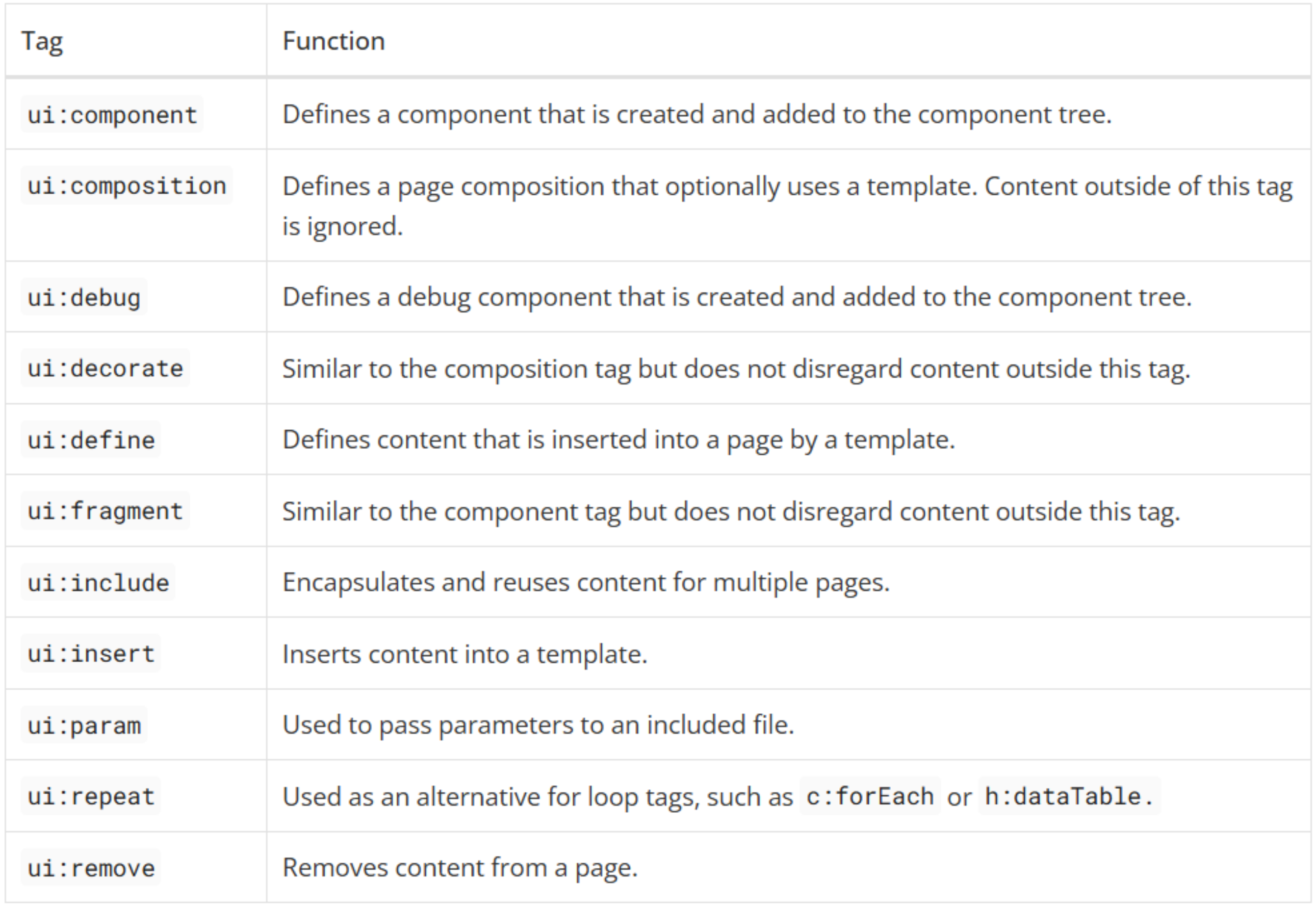
The Lifecycle of a Facelets Application

1. When a client, such as a browser, makes a new request to a page that is created using Facelets, a new component tree or jakarta.faces.component.UIViewRoot is created and placed in the FacesContext.
2. The UIViewRoot is applied to the Facelets, and the view is populated with components for rendering.
3. The newly built view is rendered back as a response to the client.
4. On rendering, the state of this view is stored for the next request. The state of input components and form data is stored.
5. The client may interact with the view and request another view or change from the Jakarta Faces application. At this time, the saved view is restored from the stored state.
6. The restored view is once again passed through the Jakarta Faces lifecycle, which eventually will either generate a new view or re-render the current view if there were no validation problems and no action was triggered.
7. If the same view is requested, the stored view is rendered once again.
8. If a new view is requested, then the process described in Step 2 is continued.
9. The new view is then rendered back as a response to the client.

Using Facelets Templates

- Jakarta Faces technology provides the tools to implement user interfaces that are easy to extend and reuse.
- Templating is a useful Facelets feature that allows you to create a page that will act as the base, or template, for the other pages in an application.
- By using templates, you can reuse code and avoid recreating similarly constructed pages.
- Templating also helps in maintaining a standard look and feel in an application with a large number of pages.
- Facelets Templating Tags lists Facelets tags that are used for templating and their respective functionality.

Facelets Templating Tags



Tag	Function
`ui:component`	Defines a component that is created and added to the component tree.
`ui:composition`	Defines a page composition that optionally uses a template. Content outside of this tag is ignored.
`ui:debug`	Defines a debug component that is created and added to the component tree.
`ui:decorate`	Similar to the composition tag but does not disregard content outside this tag.
`ui:define`	Defines content that is inserted into a page by a template.
`ui:fragment`	Similar to the component tag but does not disregard content outside this tag.
`ui:include`	Encapsulates and reuses content for multiple pages.
`ui:insert`	Inserts content into a template.
`ui:param`	Used to pass parameters to an included file.
`ui:repeat`	Used as an alternative for loop tags, such as `c:forEach` or `h:dataTable.`
`ui:remove`	Removes content from a page.

Composite Components

- Jakarta Faces technology offers the concept of composite components with Facelets. A composite component is a special type of template that acts as a component.
- Any component is essentially a piece of reusable code that behaves in a particular way. For example, an input component accepts user input. A component can also have validators, converters, and listeners attached to it to perform certain defined actions.
- A composite component consists of a collection of markup tags and other existing components. This reusable, user-created component has a customized, defined functionality and can have validators, converters, and listeners attached to it like any other component.
- With Facelets, any XHTML page that contains markup tags and other components can be converted into a composite component. Using the resources facility, the composite component can be stored in a library that is available to the application from the defined resources location.

Composite Component Tags:

Tag	Function
`composite:interface`	Declares the usage contract for a composite component. The composite component can be used as a single component whose feature set is the union of the features declared in the usage contract.
`composite:implementation`	Defines the implementation of the composite component. If a `composite:interface` element appears, there must be a corresponding `composite:implementation`.
`composite:attribute`	Declares an attribute that may be given to an instance of the composite component in which this tag is declared.
`composite:insertChildren`	Any child components or template text within the composite component tag in the using page will be reparented into the composite component at the point indicated by this tag's placement within the `composite:implementation` section.
`composite:valueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ValueHolder` suitable for use as the target of attached objects in the using page.
`composite:editableValueHolder`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `EditableValueHolder` suitable for use as the target of attached objects in the using page.
`composite:actionSource`	Declares that the composite component whose contract is declared by the `composite:interface` in which this element is nested exposes an implementation of `ActionSource2` suitable for use as the target of attached objects in the using page.

An example shows a composite component that renders an form field with a label, input and message component

```
<ui:component xmlns:ui="jakarta.faces.facelets"
              xmlns:cc="jakarta.faces.composite"
              xmlns:h="jakarta.faces.html">
    <cc:interface>
        <cc:attribute name="label" required="true" />
        <cc:attribute name="value" required="true" />
    </cc:interface>

    <cc:implementation>
        <div id="#{cc.clientId}" class="field">
            <h:outputLabel for="input" value="#{cc.attrs.label}" />
            <h:inputText id="input" value="#{cc.attrs.value}" />
            <h:message for="input" />
        </div>
    </cc:implementation>
</ui:component>
```

Using Jakarta Faces Technology in Web Pages

A typical Jakarta Faces web page includes the following elements:

- A set of namespace declarations that declare the Jakarta Faces tag libraries
- Optionally, the HTML head (h:head) and body (h:body) tags
- A form tag (h:form) that represents the user input components

To add the Jakarta Faces components to your web page, you need to provide the page access to the two standard tag libraries: the Jakarta Faces HTML render kit tag library and the Jakarta Faces core tag library. The Jakarta Faces standard HTML tag library defines tags that represent common HTML user interface components. The Jakarta Faces core tag library defines tags that perform core actions and are independent of a particular render kit.

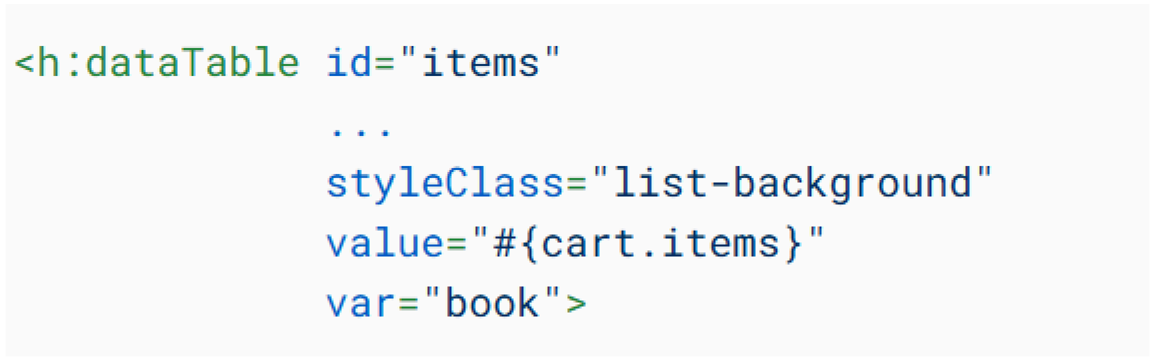
Jakarta Faces tags usage

- To use any of the Jakarta Faces tags, you need to include appropriate directives at the top of each page specifying the tag libraries.
- For Facelets applications, the XML namespace directives uniquely identify the tag library URI and the tag prefix.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="jakarta.faces.html"
      xmlns:f="jakarta.faces.core">
```

Adding Components to a Page Using HTML Tag Library Tags

- The tags defined by the Jakarta Faces standard HTML tag library represent HTML form components and other basic HTML elements.
- These components display data or accept data from the user.
- This data is collected as part of a form and is submitted to the server, usually when the user clicks a button.



```
<h:dataTable id="items"
             ...
             styleClass="list-background"
             value="#{cart.items}"
             var="book">
```

Using Converters, Listeners, and Validators

This slide provides information on adding more functionality to the components through converters, listeners, and validators.

- Converters are used to convert data that is received from the input components. Converters allow an application to bring the strongly typed features of the Java programming language into the String-based world of HTTP servlet programming.

```
<h:inputText value="#{loginBean.age}">
    <f:converter converterId="jakarta.faces.Integer" />
</h:inputText>
```

- Listeners are used to listen to the events happening in the page and perform actions as defined.

```
<h:commandLink id="Duke" action="bookstore">
    <f:actionListener
        type="ee.jakarta.tutorial.dukesbookstore.listeners.LinkBookChangeListener
    <h:outputText value="#{bundle.Book201}"/>
</h:commandLink>
```

- Validators are used to validate the data that is received from the input components. Validators allow an application to express constraints on form input data to ensure that the necessary requirements are met before the input data is processed.

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

Managed Beans in Jakarta Faces Technology

- Creating a Managed Bean: A managed bean is created with a constructor with no arguments, a set of properties, and a set of methods that perform functions for a component. Each of the managed bean properties can be bound to one of the following:
  - A component value
  - A component instance
  - A converter instance
  - A listener instance
  - A validator instance

Managed bean methods

- The most common functions that managed bean methods perform include the following:
  - Validating a component’s data
  - Handling an event fired by a component
  - Performing processing to determine the next page to which the application must navigate

Using Ajax with Jakarta Faces Technology

- Ajax functionality can be added to a Jakarta Faces application in one of the following ways:
  - Adding the required JavaScript code to an application
  - Using the built-in Ajax resource library

Using Ajax with Facelets

- Jakarta Faces technology supports Ajax by using a built-in JavaScript resource library that is provided as part of the Jakarta Faces core libraries. This built-in Ajax resource can be used in Jakarta Faces web applications in one of the following ways.

  - By using the f:ajax tag along with another standard component in a Facelets application. This method adds Ajax functionality to any UI component without additional coding and configuration.

  - By using the JavaScript API method faces.ajax.request() directly within the Facelets application. This method provides direct access to Ajax methods and allows customized control of component behavior.

  - By using the <h:commandScript> component to execute arbitrary server-side methods from a view. The component generates a JavaScript function with a given name that when invoked, in turn invokes, a given server-side method via Ajax.

Questions?