
MONASH University
MONASH
INFORMATION
TECHNOLOGY
FIT5192 Lecture 5: Introduction to Java Server Faces
GROUP
OF EIGHT
AUSTRALIA

MONASH University

Last Lecture

- Quick revision of important database concepts
- Data persistence in Java
- Using JPA to communicate with a relational database (stand-alone application)

GROUP OF EIGHT AUSTRALIA

MONASH University

This Lecture

- Get a quick introduction into the Java Server Faces Framework (JSF) and how we can build web interfaces with it.
- Understand the purpose of **Managed Beans** and how we can work with data in different scopes.
- Understand how Expression Language can be used to retrieve values from managed beans.
- Review the processes involved in the JSF framework that make up the lifecycle of a deployed web application.

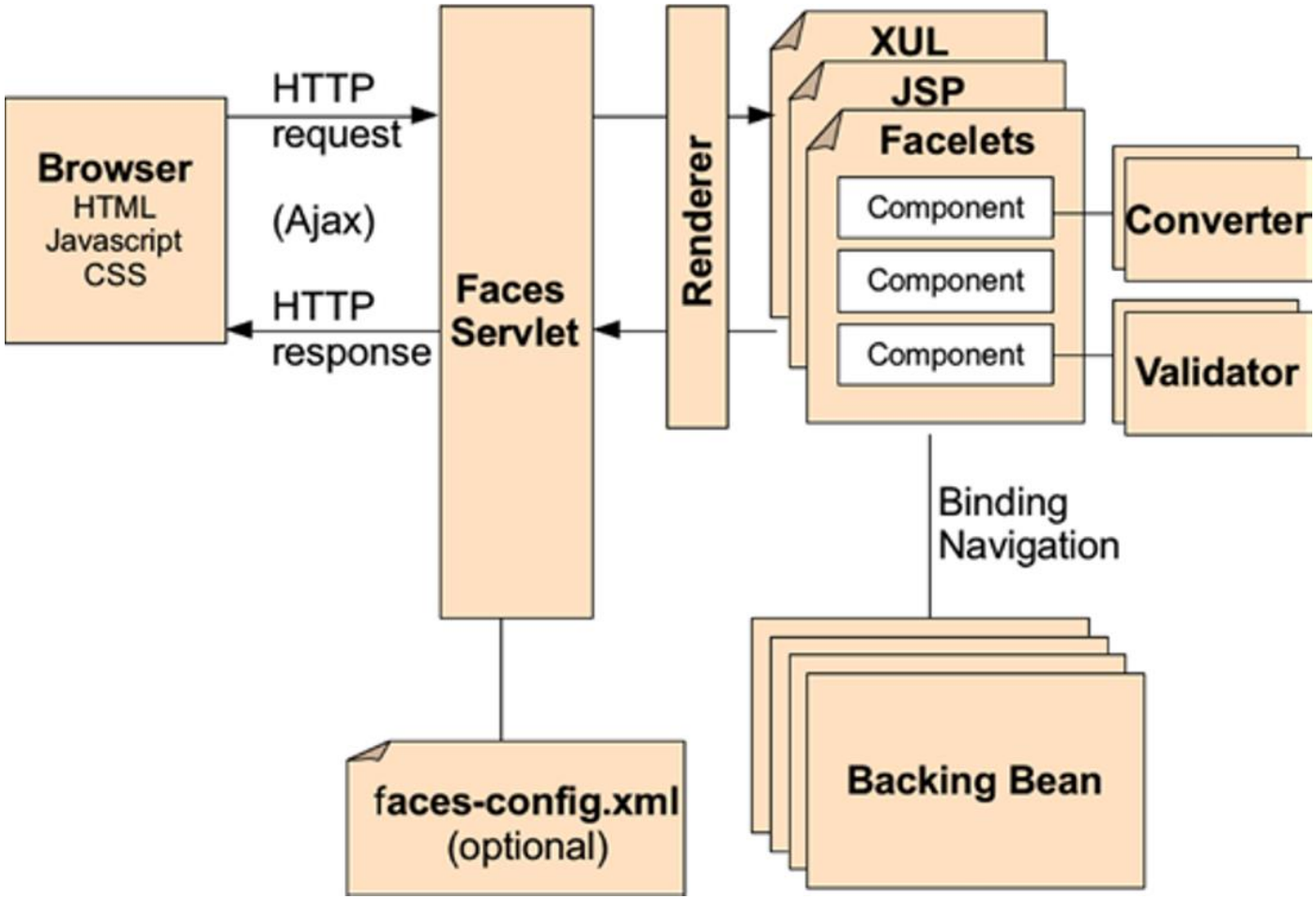
GROUP OF EIGHT AUSTRALIA

Java Server Faces

What is Java Server Faces (JSF)?

- JSF is a server side component framework or architecture for generating dynamic and interactive web pages.
- **It consists of:** Page Description Languages (PDLs, such as JavaServer Pages and Facelets);
- Backbeans or Managed Beans to implement business logic;
- Many other supporting elements.
- **Generated web page content is sent to the client / browser for displaying and providing an interface to send data back.**

JSF Architecture



Source: Figure 10-5, Beginning Java EE 7, p. 314

JSF Important Pieces

Architecture on previous slide represents several important pieces of JSF that make its architecture rich and flexible:

- **FacesServlet and faces-config.xml**: FacesServlet is the main Servlet for the application and can optionally be configured by a faces-config.xml descriptor file.
- **Pages and components**: JSF allows multiple PDLs but Facelets is the recommended one since JSF 2.0.
- **Renderers**: These are responsible for displaying a component and translating a user's input into the component property values.
- **Converters**: These convert a component's value (Date, Boolean, etc.) to and from markup values (String).
- **Validators**: These are responsible for ensuring that the value entered by a user is valid (most of the validation can be delegated to). Bean Validation
- **Backing beans and navigation**: The business logic is made in backing beans, which also control the navigation between pages.
- **Ajax support**: JSF 2.2 comes with built-in support for Ajax.
- **Expression language**: EL is used in JSF pages to bind variables and actions between a component and a backing bean.

JSF: History

- **2001: JSF 1.0 created.**
- **2004: JSF 1.1 maintenance release**
  - **2006: JSF 1.2 released and introduced to Java EE platform** Challenge was to provide backwards compatibility and integration with Expression Language.
  - **2009: JSF 2.0 released with Java EE 6** Major release focusing on ease of use, performance and enhanced functionality.
- **2010: JSF 2.1 maintenance release**
  - **2013: JSF 2.2 released with Java EE 7** Improved on functionality with features such as page flow

Some of the new features in JSF 2.2

- **HTML5 Friendly Markup** Features such as pass-through attributes and elements make it easier to work with more involved HTML5 content.
- **Faces Flows** A major improvement enabling better management of navigating the user between a series of linked pages.
- **Queue control for AJAX requests** Better handling of multiple AJAX requests made in a short period of time.
- **Stateless Views**
- **General security enhancements**

Java Server Pages/ Facelets/ Managed Beans

JavaServer Pages (JSP)

- **JavaScript (JS) can generate dynamic web pages but runs on the browser / client. It was thought as not interact with business logic and access external databases directly.** JS is starting to become useful for backend development with solutions such as **NodeJS**.
- **Java Server Pages (JSP) provided a way to write dynamic web pages such as HTML5 documents, without being a traditional programmer.** Enables Java code to be embedded within HTML markup.
- **JSPs are executed on supported server middleware such as GlassFish or Apache Tomcat,** providing dynamic output as they can execute Java instructions tailored specifically to individual requests made by clients.

Facelets

- **Web template system and the default view handler for JavaServer Faces.** Uses XML documents (via XHTML) to declare the components within a view.
- Much easier to define rich web interfaces than the previous JSP approach (depreciated as view handler from JSF 2.0).
- **Facelets has support for all JSF UI components and helps structure the JSF component tree (the view).** Also provides support for templating and use of Expression Language.

Facelets; background

- Facelets began life as an open source alternative to JSP.
- Unlike JSP, EL, and JSTL, it didn't have a JSR and was not part of Java EE.
- Facelets was a replacement for JSP and provided an XML-based alternative (XHTML) for pages in a JSF-based application.
- Facelets was designed with JSF in mind, and this is why it provided a simpler programming model than JSP's.
- Because of that, Facelets got specified in JSF 2.0 and comes today in Java EE 7 as the preferred PDL for JSF.

***Note**: This slide is for information only*

Managed Beans

- **A managed bean is a regular bean registered with JSF.** A Java Bean is a convention for a class where all properties are *private*, constructor has *no arguments* and implements *Serializable*.
- Works as the *model* for the JSF user interface components.
- **Managed beans are accessible from JSF pages.**
- **JSF automatically looks after Managed Beans:** Insanitation (constructors must have no arguments)
- Lifecycle (Scoping)
- Calling accessor / mutator methods (via Expression Language)

Declaring Managed Beans

- We use the @Named annotation in a class to identify a bean that will be managed by JSF:

```
@Named
@SessionScoped
public class User extends Serializable { ... }
```

- Using annotations enables us to not worry about configuring instances in faces-config.xml (Old approach!).
- Scope can also be defined within the class to determine how the application should store the bean.
- Annotations also available for functions such as validation which we will explore more closely next week.

Faces Servlet

- FacesServlet is internal and part of JSF.
- The only way it can be configured is by using external metadata.
- Up to JSF 1.2, the only source of configuration was the faces-config.xml file.
- Today, with JSF 2.2, this file is optional, and most metadata can be defined through annotations (on backing beans, converters, components, renderers, and validators).

***Note**: This slide is for information only*

Managed Bean Scopes (Part 1)

- Managed beans can be instantiated in different scopes.
- **Application:** @ApplicationScoped Longest life span, available for application runtime duration.
  - Available to all request / responses – needs to be thread-safe via synchronised keyword!
- **Session:** @SessionScoped Available for any request / response tied to the client's session.
  - State persisted between requests until session invalidated.
- **Request:** @RequestScoped **Default** scope, creates a new instance for each HTTP request

Managed Bean Scopes (Part 2)

- **View**: @ViewScoped Lives as long as you are interacting with the same JSF view in the client (ideal for AJAX requests).
- **Flash** Temporary objects designed for short-lifespans.
  - Can only be used programmatically.
- **Flow:** @FlowScoped Introduced with JSF 2.2
  - Useful for establishing a scope for a *combination* of pages (E.g. checkout process)

Managed Bean Scopes (Part 3)

- **Dependent:** @Dependent Useful for when a managed bean is referencing another managed bean.
  - If a bean is defined as dependent, it will be instantiated each time it is referenced so it does not get retained in any scope.

Example: Simple Managed Bean

```java
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;

@Named(value = "user")
@SessionScoped
public class User implements Serializable {

    private String name;
    private String password;

    public User() { }

    public String getName() {
        return name;
    }

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

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Expression Language

Expression Language

- **JSP pages enable the combination of Java, HTML and textual content.** Mixing a verbose language such as Java can make it significantly harder to read code.
- Expression Language (EL) was created to perform similar functions to Java code but in a more markup friendly format.
- **The binding between a JSF page and a supporting Managed Bean is done via EL which enables:**
  - Output of values
  - Invoking managed bean methods

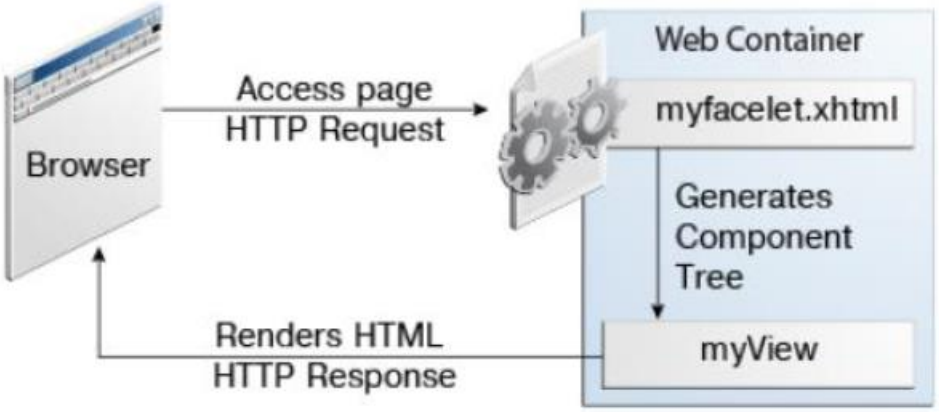
EL: Supported Operators

- **General**
  . and []
- **Arithmetic**
  +, -, *, / (div), % (mod)
- **Logical**
  && (and), || (or), ! (not)
- **Relational**
  == (eq), != (ne), < (lt), > (gt), <= (le), >= (ge)
- **Conditional**
  A ? B : C (if A is true, then evaluate B, else evaluate C)

Expression Language: Examples

- Basic syntax for EL expression: #{expr}
- **Access *password* property from *UserBean*:**
  #{userBean.password}
- **Check if *username* property from UserBean is empty:**
  #{empty userBean.username} // returns true/false
- **Access *defendingTeam* key from *teams map:***
  #{teams[defendingTeam]}
- **Output odd or even depending on dice value:**
  #{dice.num % 2 ? "Even" : "Odd"}

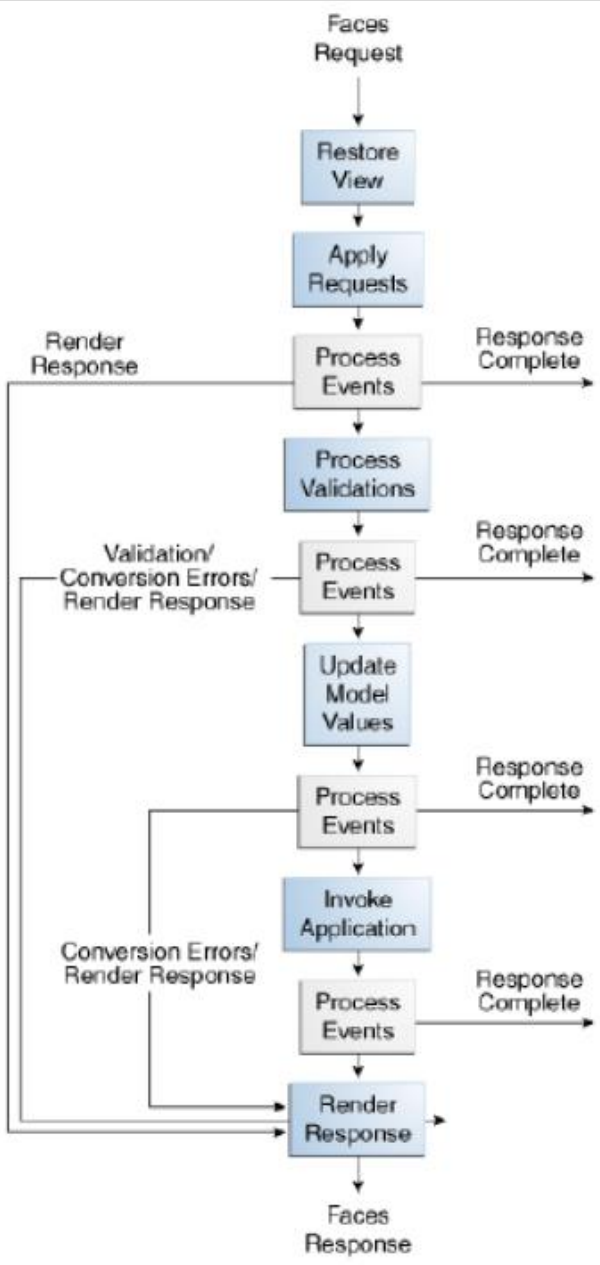
JSF: Responding to Client Requests



- The client (web browser) requests for a specific page (myfacelet.xhtml) via HTTP request.
- Web server processes the request via the JSF Lifecycle (next slide) and generates the view to be sent back.
- **Web server returns the processed view via HTTP Response.** E.g. HTML5 document

JavaServer Faces Lifecycle

- **Two types of requests are handled:**
  - *Initial Request:* First request to a page
  - *Postback Request:* Submitting form back to previously rendered view
- **Lifecycle comprised of six key phases:**
  - Restore View
  - Apply Request Values
  - Process Validations
  - Update Model Values
  - Invoke Application
  - Render Response



Phase: Restore View

- When a request is made for a JSF page, the lifecycle begins with this phase.
- **Actions performed in phase:** Page view is established.
- Setup event handlers and validators.
- Saves view in the *FacesContext* instance (contains all information necessary to process request).
- **If requesting page for the first time, JSF establishes a blank view and the lifecycle advances to the *Render Response* phase.** If a postback request, information is already stored in *FacesContext* so JSF restores the view using saved data.

Restore View; technical details

- **Restore view:** JSF finds the target view and applies the user's input to it.
- If this is the client's first visit to the page, JSF creates the view as a **UIViewRoot** component (root of the component tree that makes up a particular page.)
- If this is a subsequent request, the previously saved **UIViewRoot** is retrieved for processing the current HTTP request.

***Note**: This slide is for information only*

Phase: Apply Request Values

- After the **component tree** (view) is restored, each component will inherit their new values via request parameters.
- Values that have come from the request (such as input fields from a submitted form) are applied to various components within the tree.
- UI components are focused on updating their state, not the managed bean that models the data.

Apply Request Values; technical details

- **Apply request values**: Values that have come from the request (from input fields in a web form, from cookies, or from request headers) are applied to the various components of the page.
- Note that only UI components update their state—not the backing bean that composes the model.

***Note**: This slide is for information only*

Phase: Process Validations

- **All specified validators registered on components in the view are processed.** Examines attributes that define the validation rules and compares to the value stored with the component.
- **If validation and conversion is successful for all components, lifecycle continues to the *Update Model Values* phase.**
- **If validation or conversion of values fails, an error is recorded in *FacesContext*.** Lifecycle moves to the ***Render Response*** phase with the error and conversion messages.

Process Validations; technical details

- **Process validations:** After the preceding steps, the UI components have a value set.
- In the validation processing, JSF traverses the component tree and asks each component to ensure its submitted value is acceptable.
- If both conversion and validation are successful for all components, the life cycle continues to its next phase.
- Otherwise, the life cycle goes to the Render response phase with the appropriate validation and conversion error messages.

***Note**: This slide is for information only*

Phase: Update Models Values

- Once JSF determines that components have been validated successfully, values are then bound to the associated managed beans.
- **If the values cannot be converted to the type specified in the managed beans, lifecycle proceeds to the *Render Response* phase.** Enables error messages to be outputted to the user.

Update Models Values; technical details

- **Update model values**: All the validated component values are bound to the associated backing beans.

***Note**: This slide is for information only*

Phase: Invoke Application

- Business logic can now be performed.
- **Any application-level events such as form submission or linking to an application resource (such as a page or stylesheet) are processed.** If the application needs to handle other resources or generate responses not reliant on any JSF component, *FacesContext*.responseComplete() is called.
- JSF moves to the *Render Response* phase once all processing has been completed.

Invoke Application; technical details

- **Invoke application:** Now you can perform some business logic.
- Whatever action has been triggered will be executed on the backing bean, and this is where the navigation comes into effect, as its return will determine the render response.

***Note**: This slide is for information only*

Phase: Render Response

- JSF creates the view with all the processed information.
- **Initial requests will have JSF components in the page added to the component tree (view).** If not an initial request, components have already been added so do not need to be reprocessed.
- **Errors encountered in a postback request during either the *Apply Request Values, Process Validations* or *Update Model Values* phases requires the page to be re-rendered.** Errors are outputted to any message tags specified.
- **After view has been rendered, response state is saved so that requests may access it.**

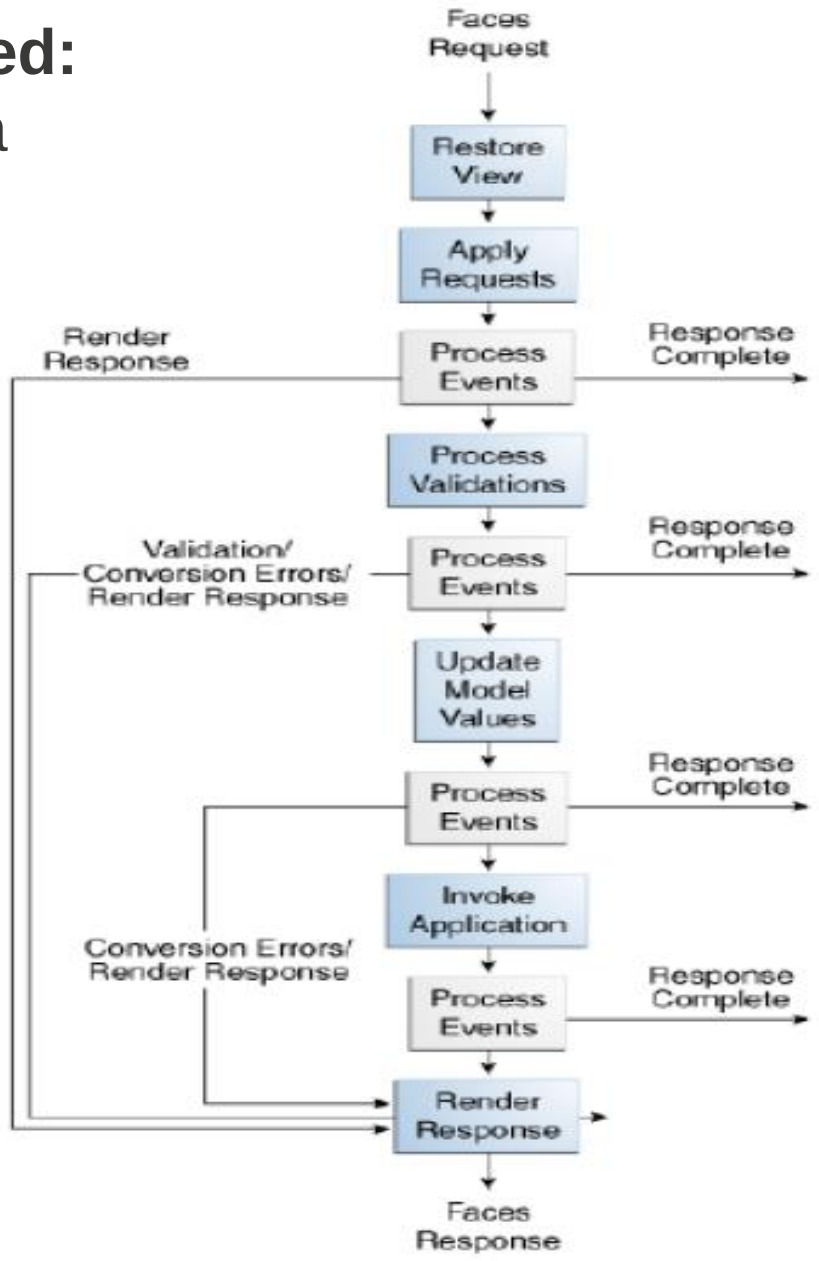
Render Response; technical details

- **Render response:** Causes the response to be rendered to the client.
- The secondary goal of the phase is to save the state of the view so that it can be restored in the restore view phase if the user requests it again.

***Note**: This slide is for information only*

Summary of JavaServer Faces Lifecycle

- **Two types of requests are handled:**
  - *Initial Request:* First request to a page
  - *Postback Request:* Submitting form back to previously rendered view
- **Lifecycle comprised of six key phases:**
  - Restore View
  - Apply Request Values
  - Process Validations
  - Update Model Values
  - Invoke Application
  - Render Response



JavaServer Faces Lifecycle; technical details

- The thread of execution for a request/response cycle can flow through each phase or not, depending on the request and what happens during the processing; if an error occurs, the flow of execution transfers immediately to the render response phase.
- Note that four of these phases can generate messages: apply request values, process validations, update model values, and invoke application.
- With or without messages, it is the render response phase that sends output back to the user.

***Note**: This slide is for information only*

Summary

- Get a quick introduction into the Java Server Faces Framework (JSF) and how we can build web interfaces with it.
- Understand the purpose of Managed Beans and how we can work with data in different scopes.
- Understand how Expression Language can be used to retrieve values from managed beans.
- Review the processes involved in the JSF framework that make up the lifecycle of a deployed web application.

Next Lecture

- **Developing Web Interfaces with JavaServer Faces** Using the MVC pattern approach
- Working with data validation
- Working with AJAX and JSON Processing

MONASH University

See you in the Studio !

Readings

MONASH University

- ***Recommended:*** Chapter 10: JavaServer Faces in *Beginning Java EE 7, Antonio Goncalves*
- ***Background:*** Part III The Web Tier in The Java EE 7 Tutorial – Chapter 6 to 13, covers most of the topics we have looked at