Java Server Faces, JSF Pages, Managed Beans and Request Cycle

JSF Slides #2

JavaBeans

Recurring term: **JavaBeans**, session bean, managed bean, enterprise bean...

Basically

- Plain old Java class (POJO)
- Private attributes and read/write methods for (relevant) attributes (attribute +set + get is called **a property**)
- Naming conventions for set/get-methods

```
private String data;
public String getData();
public void setData( String str );
```

- Non private default constructor
- Serializable

.. sadly it's a bit of an antipattern... (immutable hard...)

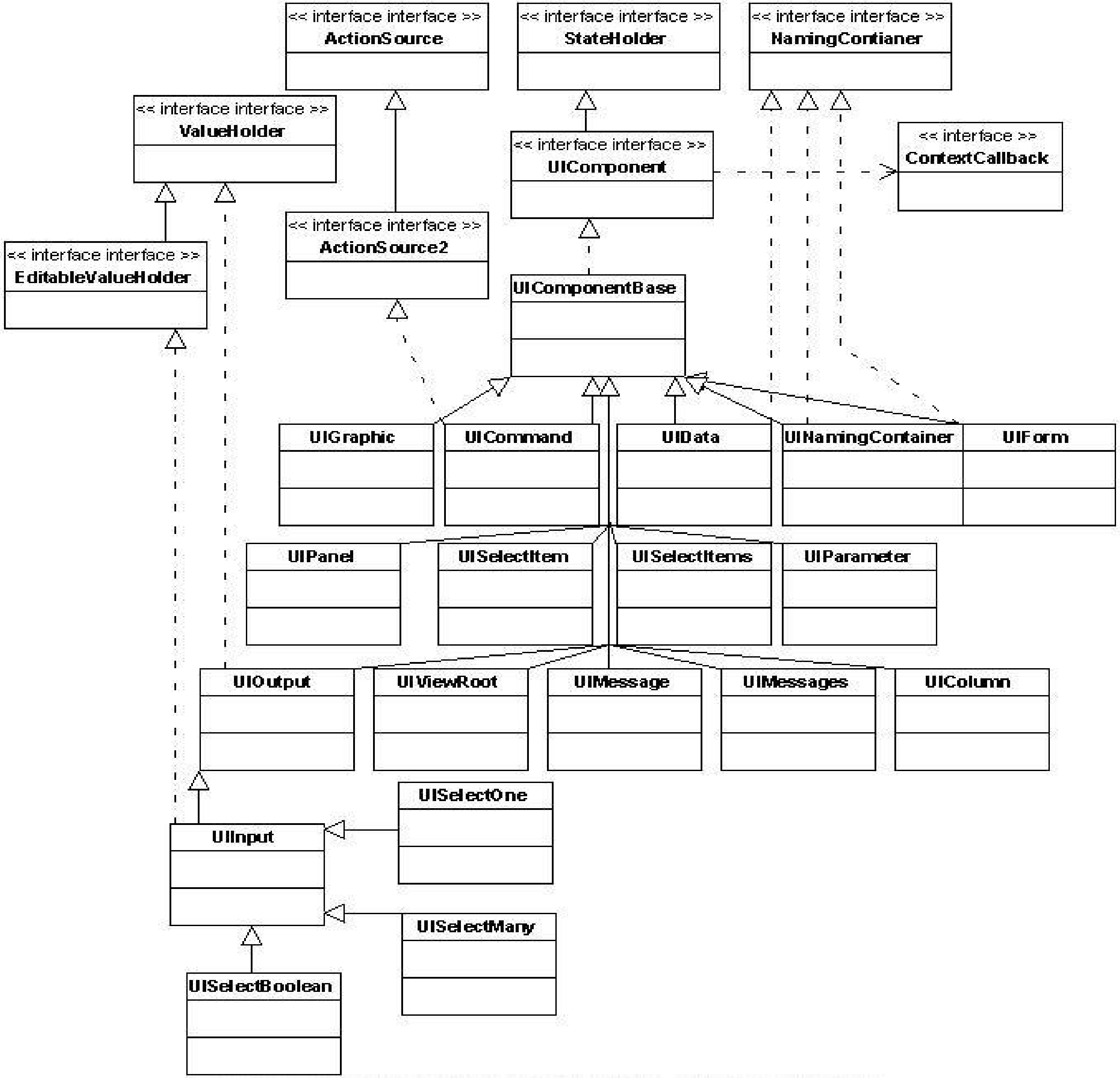
Java Server Faces

JEE technology for server-side user interfaces

Basically

- Server side component classes (inputs, buttons, …)
- Tag sets to describe component trees (using XHTML files)
  - A **JSF Page** (confusing also called Facelets tags/page)
- Facelets, a templating system for composing JSF pages (other tags)
- A renderkit to translate component trees to renderable format (spec. requires a HTML renderkit). Handled transparently by JSF
- **Managed beans** to hold GUI-data and act as GUI-listeners (adhere to JavaBeans)
- A request cycle to handle request and responses
- Conversion, validation and navigation

JSF Component Classes



<< interface interface >> ActionSource
<< interface interface >> StateHolder
<< interface interface >> NamingContianer
<< interface interface >> ValueHolder
<< interface interface >> UIComponent
<< interface >> ContextCallback
<< interface interface >> ActionSource2
<< interface interface >> EditableValueHolder
UIComponentBase
UIGraphic
UICommand
UIData
UINamingContainer
UIForm
UIPanel
UISelectItem
UISelectItems
UIParameter
UIOutput
UIViewRoot
UIMessage
UIMessages
UIColumn
UISelectOne
UIInput
UISelectMany
UISelectBoolean

JavaDoc

JSF Pages

Possible JSF Page content

- XML prolog (mandatory)
- The XHTML tags (static content). Well-formed!
- JSF tags <h: … > (html elements>, <f: … > (core), …
- Facelets tags <u: … >, more to come …
- The Expression Language (EL)
  - Same as JSP but a bit different use, upcoming …

Again: Pages are not returned to the client they are used to construct the component tree

EL in JSF

Same as in JSP but uses deferred syntax

- **#**{ … expression … } **#**=deferred (JSP used immediate ${…})
- Immediate syntax is read only
- Deferred is both read and write (depends on expression location (input vs output component)

JSF Page Example

Note: Namespaces have changed.
Possibly problematic (bug).
New namespaces
xmlns:f="http://xmlns.jcp.org/jsf/core"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:ui="http://xmlns.jcp.org/jsf/facelets"

NetBeans will use new (possible have to change to old)!

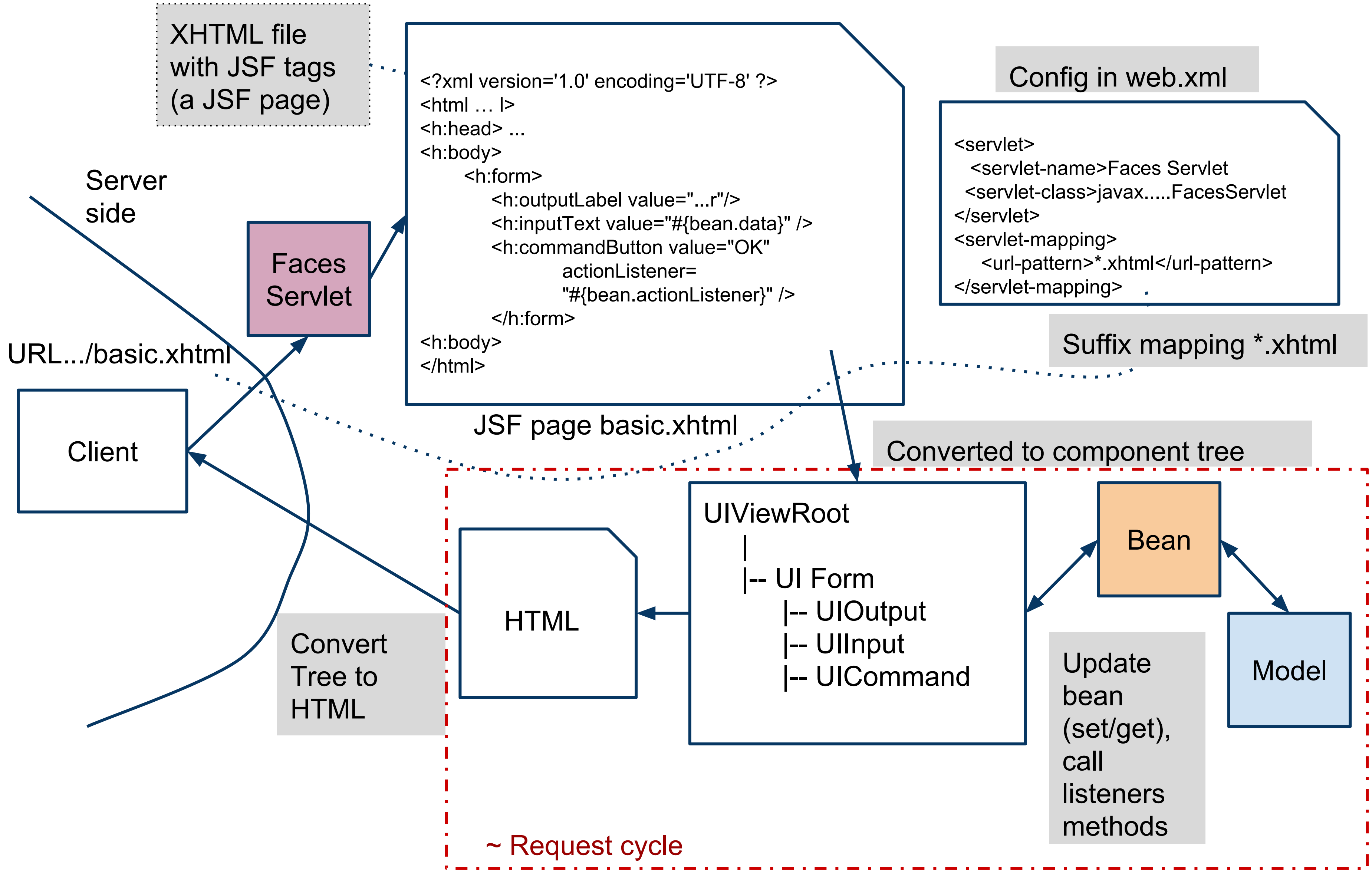
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "htt
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
        <title>Basic</title>
    </h:head>
    <h:body>
          <h1>Basic</h1>
            <p>Page sometimes reacts strange, because of simpleness (resends POSTs) </p>
            <p>Also watch GlassFish output </p>
               <h:form>
                    <h:panelGrid columns="2" class="table">

                          <h:outputLabel value="Enter a number"/>
                          <h:inputText value="#{basicbean.data}" />
                          <h:outputLabel value="Data in bean is now"/>
                          <h:outputLabel value="#{basicbean.data}"/>
                      </h:panelGrid>
                      <h:commandButton value="OK"
                                       actionListener="#{basicbean.actionListener}" />
                      <h:commandButton value="Clear"
                                       actionListener="#{basicbean.setData(0)}" />
                      <h:messages/>
               </h:form>
     </h:body>
</html>
```

The Faces Servlet

JSF is a subsystem of the container

- To invoke the JSF subsystem, request must go through the **FacesServlet** (supplied by JSF)
  - Could be an JSP/JAX-RS request, server may handle many applications/resources
- Servlet "invisible" to developer
- Configuration of FacesServlet in web.xml
- URL pattern to trigger Servlet (*.xhtml or other, ...), specify in web.xml (possible multiple patterns)

JSF Application Overview



XHTML file with JSF tags (a JSF page)

```
<?xml version='1.0' encoding='UTF-8' ?>
<html … l>
<h:head> ...
<h:body>
        <h:form>
            <h:outputLabel value="...r"/>
            <h:inputText value="#{bean.data}" />
            <h:commandButton value="OK"
                        actionListener=
                        "#{bean.actionListener}" />
            </h:form>
<h:body>
</html>
```

JSF page basic.xhtml

Config in web.xml

```
<servlet>
   <servlet-name>Faces Servlet
  <servlet-class>javax.....FacesServlet
</servlet>
<servlet-mapping>
     <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

Suffix mapping *.xhtml

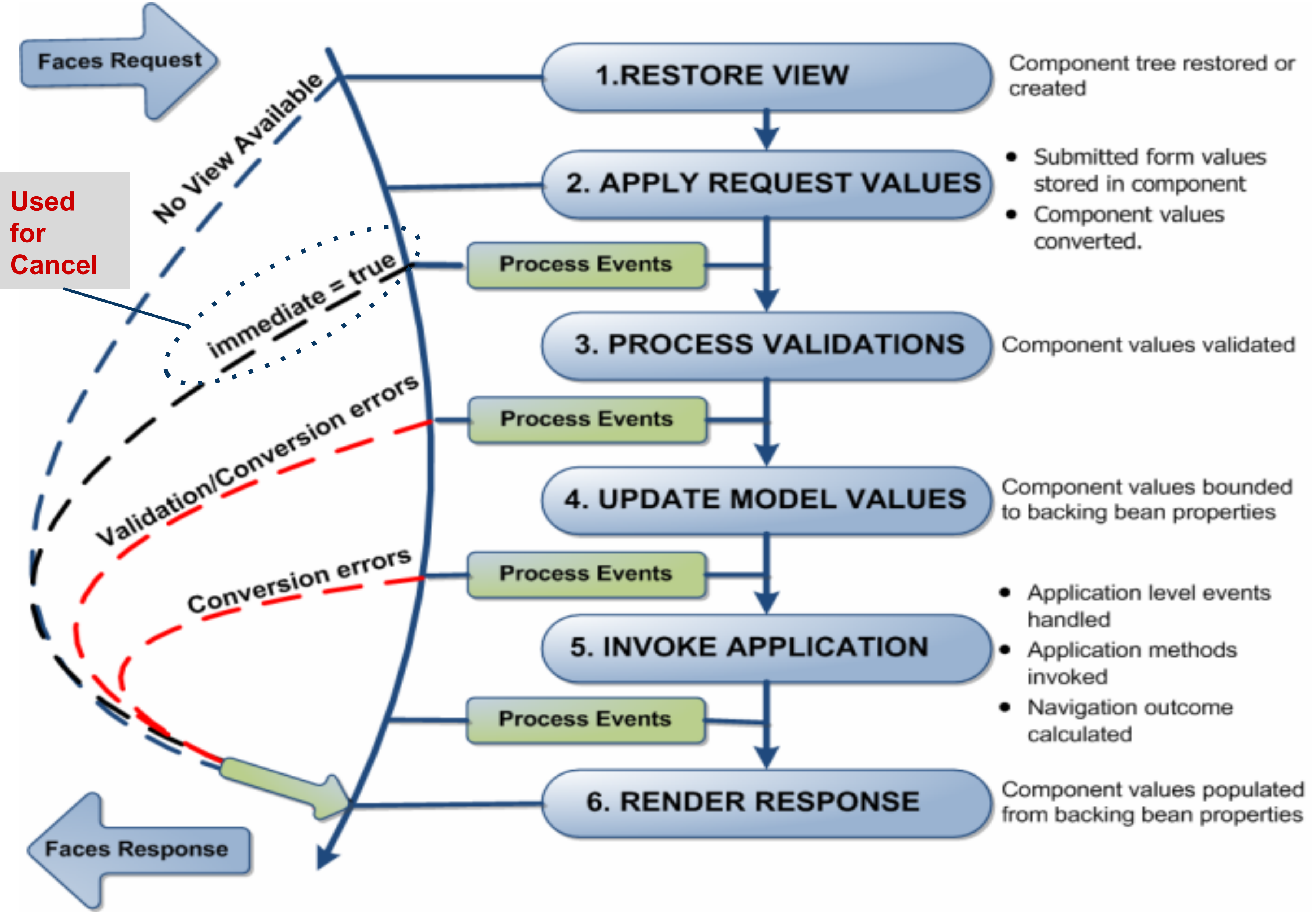
Converted to component tree

Convert Tree to HTML

Update bean (set/get), call listeners methods

~ Request cycle

The JSF Request Cycle


Faces Request
1.RESTORE VIEW
Component tree restored or created
2. APPLY REQUEST VALUES
• Submitted form values stored in component
• Component values converted.
Process Events
3. PROCESS VALIDATIONS
Component values validated
Process Events
4. UPDATE MODEL VALUES
Component values bounded to backing bean properties
Process Events
5. INVOKE APPLICATION
• Application level events handled
• Application methods invoked
• Navigation outcome calculated
Process Events
6. RENDER RESPONSE
Component values populated from backing bean properties
Faces Response
No View Available
immediate = true
Validation/Conversion errors
Conversion errors
Used for Cancel

Request Cycle and HTTP Requests

Post request (h:commandButton, h:commandLink)

- Will trigger a full cycle
- Except if using attribute "immediate" in <h: ... /> (useful for "Cancel" button)
- Except if configuration/exception issues
- Must be inside <h:form ...>

Get request (h:link)

- Default, short circuit cycle: Restore View -> Render Response
- Possible to invoke cycle using **view parameters**

View Parameters

Using GET request to set values in beans

- Add link with query parameters source page

```
<!-- Add a link with query param (index.xhtml?data=100000) -->
<h:link outcome="index" value="Send data" >
       <f:param name="data" value="100000"  />
 </h:link>
```

- Retreive in target page

```
<!-- Target page index.xhtml, use viewParams to get
     parameter “data” and set value in bean (add before <head> )-->
<f:metadata>
   <f:viewParam name="data" value="#{bean.data}"/>
</f:metadata>
```

- Will trigger full request cycle
- Note: Get are for reads. <u>Use to specify read criteria</u>

JSF Implicit Objects

Mostly same as JSP implicit objects

Some special
- #{facesContext}, FacesContext instance for current request, upcoming…
- #{view}, viewroot for current component tree

Possible to get the current viewId (possible later use, "to get back")
- #{view.viewId}

Not much used …

HTML5 Friendly JSF

Pass-through elements allow you to use HTML5 tags and attributes but to treat them as equivalent to JavaServer Faces components

```
// If jsf: attribute element will be passed through (can use EL)
<input type="email" jsf:id="email" name="email"
           value="#{reservationBean.email}" required="required"/>
```

Pass through attributes allow you to pass attributes that are not JavaServer Faces attributes through to the browser without interpretation

```
// JSF tag use p: to pass through
<h:inputText id="nights" p:type="number" value="#{bean.nights}"
p:min="1" p:max="30" p:required="required"
             p:title="Enter a number between 1 and 30 inclusive.">
```

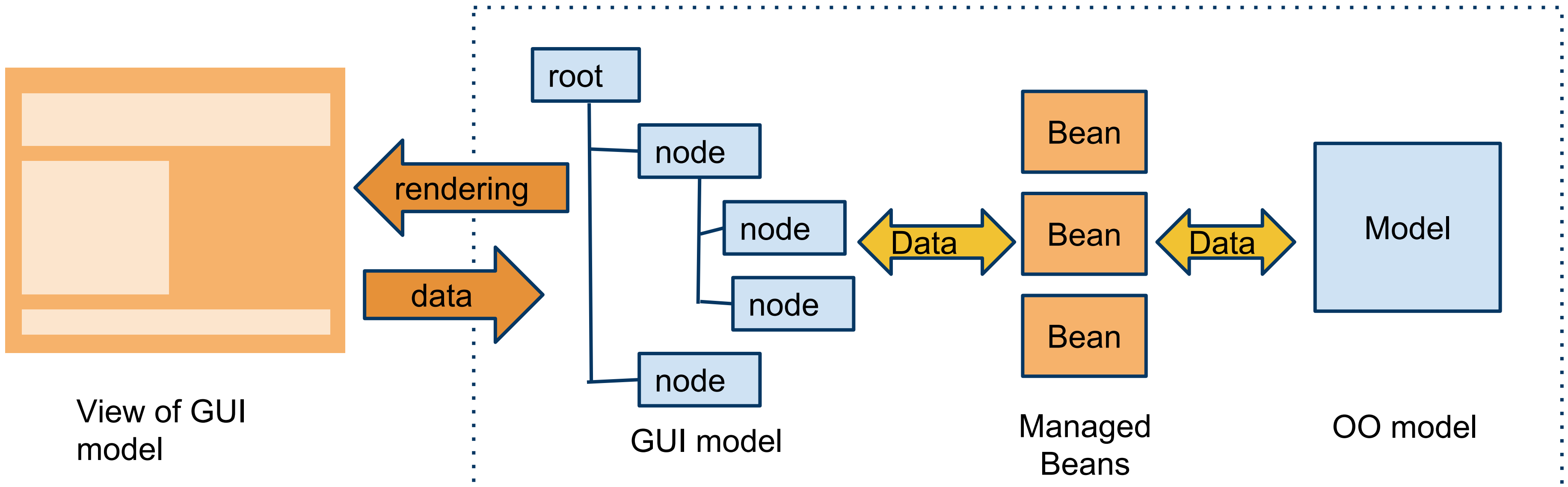
Managed Beans

Managed Beans used as
- Managed by container
- Glue between server side GUI (component tree) and model
- Receiver of incoming data (get from components in GUI-tree, set in bean), possible handle over to model or subsystem
- Supplier of outgoing data (get from bean, set in components in GUI-tree), possible read from model or subsystem
- Possible (often) as event listener
- Possible (often) define navigation (depending on processing outcome)
- Must run in some container (that can manage)

Design: Beans have various "roles", to be continued...

Managed Beans, cont

Application location of managed beans



Server side

Managed Beans Confusion

JEE is an evolving platform, things change...
- We will only (for now) use Context and Dependency Injection (CDI) managed beans (reference implementation **Weld**).

- There are also JSF managed beans, we don't use (replaced by CDI)
    - Many code samples on the web, watch out!
    - The annotation @ManagedBean should never occur in code!!!

```
@ManagedBean
public class HelloBean ... {
             // BAD, BAD, BAD, BAD...
```

CDI Managed Beans Packages

Packages used for CDI managed beans
- javax.annotation.*
- javax.inject.*
- javax.enterprise.context.*

Never
- javax.faces.*
- Except javax.faces.event.* and javax.faces.view.ViewScoped;

Warning, common mistake
- Importing java.awt.ActionEvent, BAD...should be
- javax.faces.event.ActionEvent!

CDI Scopes

The built -in scopes (again: This is from javax.enterprise.*)
-@RequestScoped
-@SessionScoped
-@ApplicationScoped
-@ConversationalScoped

Other scopes possible to use for beans
-@Singleton, a singleton (a pseudo scope), avoid
-@FlowScoped
-@ViewScoped (sadly placed in javax.faces.view.*)
-@FlashScope
- Dependant pseudo-scope, if no annotation, scope of owner

Warning don't confuse with javax.faces.* scopes check packages!!

A CDI Managed Bean

```
@Named // CDI annotation! Refer to bean by someName (in pages)
@SessionScoped   // CDI scope!! Not JSF
public class SomeBean implements Serializable {

   private int data;

   ... set/get for data
}
```

For this to work must have (almost empty) file WEB-INF/beans.xml,

- NetBeans will normally create (or show lightbulb as a warning, click lightbulb to generate file)

If other name preferred use @Named("myName")

Bean Scopes Usage

Some bean scopes connect to MVC roles
- There are different opinions, … recommended

- **Backing bean** @RequestScoped, set/get data for specific page. A 1:1 relation with page. UI logic

- **Controller bean** @RequestScoped, administrate the interaction between GUI and model (call model methods) and then navigate. (see action, in Navigation slides). A 1:1 relation with a backing bean

Bean Scopes Usage, cont

Non MVC roles

- **Support Bean**
- @SessionScoped usage
  - Support one <u>or more</u> views for single user (populate select boxes, etc …)
  - Session data for a user
- @ApplicationScoped usage
  - Support one or more views for all user
  - General utility data for all users

Other scopes are for special (not so common cases)

- @ViewScoped, Set/get data for <u>specific page using AJAX</u>
- @FlowScoped, for a flow of forms sharing the same object(s)

Passivation

“A bean is called passivation capable if the container is able to temporarily transfer the state of any idle instance to secondary storage.” (for example SessionScoped)

“A managed bean is passivation capable if and only if the bean class is serializable”

“If a managed bean which declares a passivating scope: [...] has a non-transient injected field, bean constructor parameter or initializer method parameter that does not resolve to a passivation capable dependency [...] the container automatically detects the problem and treats it as a deployment problem” // CDI Spec

Exceptions may occur ....

CDI Dependency Injection

Injection as before (compare AngularJS).
- Container create objects and assigns references to (via proxy object)
- Life cycle handled
- Possible to inject interface types (loose coupling)
- Possible to inject "any managed into any other managed" (often CDI beans into CDI bean), but must have an **injection point**

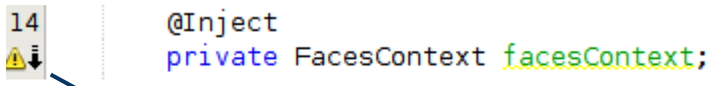
Injection points
- Constructor injection, prefer…(can only have one)
- Initializer method injection, ok …
- Field (attribute) injection, … bad (makes it hard to test)

CDI Injection Points

```
// Constructor. Note: Must have non private default ctor (also)
public class Checkout {
private final ShoppingCart cart; // Other CDI bean
@Inject
public Checkout(ShoppingCart cart) {
this.cart = cart;
}}
```

```
// Initializer method
public class Checkout {
private ShoppingCart cart;
@Inject
void setShoppingCart(ShoppingCart cart) {
this.cart = cart;
}}
```

```
// Field, avoid cant mock-up
public class Checkout {
@Inject
private ShoppingCart cart;
}}
```



```
@Inject
private FacesContext facesContext;
```

Warning! No bean matches the injection point!

What Get Injected

Very many options to specify

- Simple case just one type possible
- Complex when many implementations of interface and more...see CDI ref.

Injection Timing

This is the order

1. Constructor run (if constructor injection, do inject)
2. Initializer methods and field injection
3. Life cycle callback executed, upcoming...

Beans Lifecycle Callbacks

@PostContruct
- Called after constructor
- <u>After injection done (i.e. no null pointer exception)</u>
- Only one method can have

```
@PostConstruct
public void postContruct() { // Any name
     // Do something...
}
```

@PreDestroy
- Called before destruction (release resources)

Uniqueness

After renaming NetBeans in between doesn't build clean

- Old bean still there (bean name ambiguous)
- Will not deploy
- Clean and build

JSF Events

Kinds of events

- **System events** (for specific points in request cycle), preRender, postValidate, ...
- **Application events** (we normally use)
  - **ActionEvent**, Emitted by a UICommand component when the user activates the corresponding user interface control (such as a clicking a button or a hyperlink).
  - **ValueChangeEvent**, Emitted by a UIInput component (or appropriate subclass) when a new local value has been created, and has passed all validations.

Managed Bean as Event Listeners

Managed beans may have listener methods, can act as event listener (method name doesn't matter, parameters do)

```
@Named("myBean") @RequestScoped
public class MyBean ... {
   //Callback for valueChangeListener attribute in <h: ...>
   public void valueChange(ValueChangeEvent evt){
    ...// name of method arbitrary
   }
   // Callback for actionListener attribute in <h: ...>
   public void buttonClicked(ActionEvent evt) {
    ...
   }
}
```

Event Listeners in JSF Page

Example (using bean from previous slide)

```
// Using action and actionListener (event object supplied
// automagically for actionListener)
<h:commandButton value="OK" …
   actionListener="#{myBean.buttonClicked}"  … />


// Using valueChangeListener
<h:inputText …  valueChangeListener="#{myBean.valueChange}" >
```

Arbitrary Method Calls

May call arbitrary method in CDI bean

```
 // Ok, passing a String
<..."#{anyName.someMethod('abc'}"../>

// Ok passing a number
<..."#{anyName.otherMethod(123)}"../>
```

A lot of things happens in background, type coercions, ...
- Note: EL doesn't support overloading

Component Binding

Possible to access UIComponents in GUI tree in managed bean, **component binding**

In page

```
<!-- Tag will create a component in server GUI -->
<...binding="#{bean.txtData}".../>
```

In bean

```
// txtData reference to component in tree
private UIInput txtData; // Also need set and get
public void buttonClicked(ActionEvent evt) {
     // Will show up in GUI
     txtData.setValue("Hello");
}
```

Faces Context

FacesContext object contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the rendering of the corresponding response. A entry to...

- ...the component tree, an event queue (storing events for later execution, due to the processing model), messages (possible failure messages to GUI)
- Also possible to access low level data (HTTP Session, ...)

```
// Globally accessible (not injectable)
FacesContext ctx = FacesContext.getCurrentInstance();
```

JSF Session Handling

There's an HTTP-session associated with each user
- javax.servlet.http.HttpSession (same as with Servlets)

On logout or similar we should invalidate the session

```
 // Accessing low level data
FacesContext.getCurrentInstance().
                     getExternalContext().invalidateSession();
```

Exceptions

Exceptions in control layer, use FacesContext

```
// In managed bean
try {
     // If login fail, exception
     cr.add(new Customer(login, passwd, "...", 10));
     return "login";
 } catch (Exception e) {
     // Will end up in <h:message(s) ../>
     FacesContext.getCurrentInstance().addMessage(
                 new FacesMessage("Bad Login name"));
 }
```

Exception Messages

In JSF Pages

Tags for error messages
- <h:message..>, for specific component
- <h:messages..> (ending 's') all messages

Connect message to component

```
  <h:inputText id="userName" ..../>
  <h:message for="userName"... />
```

Possible to set messages in Java code, but we prefer Bean validation, more to come...