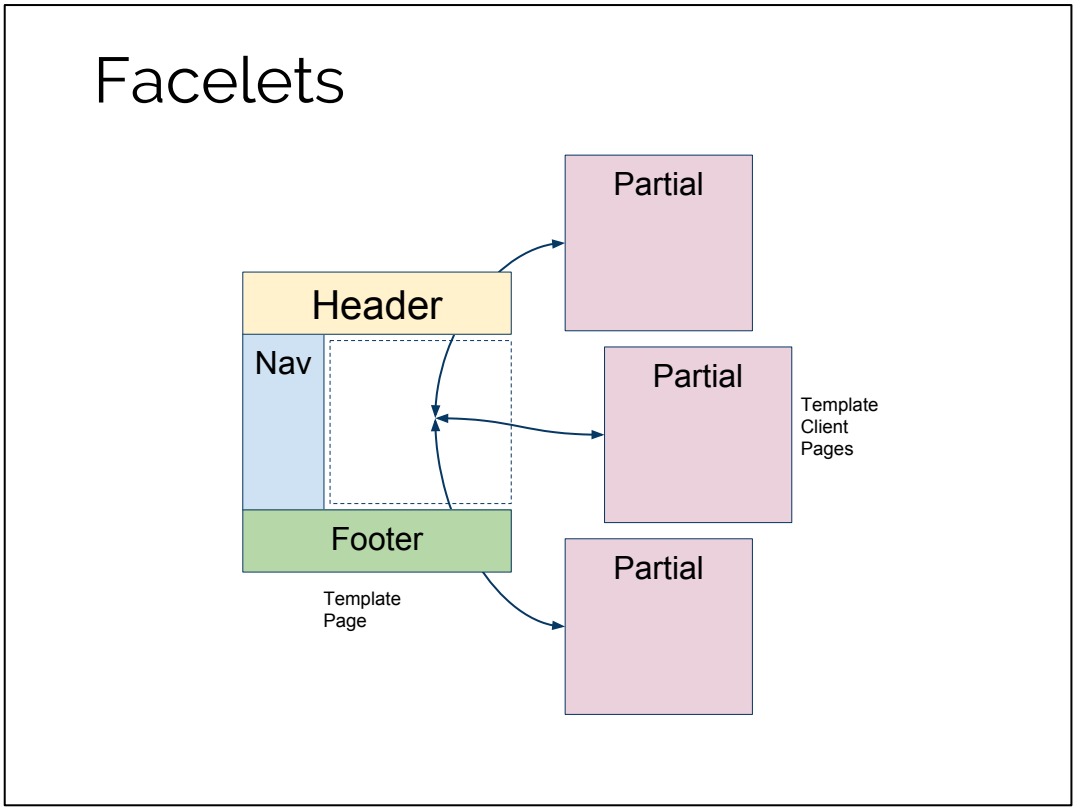
Facelets, Conversion, Validation, and Navigation

JSF Slides #3

Content

- Facelets
- Templates and Template Clients
- Conversion
- Validation
- Exceptions
- Navigation
- PRG-pattern
- JSF Request Cycle



Facelets refers to the view declaration language for JavaServer Faces technology

- So the JSF pages are if fact Facelets pages
  - I say JSF pages
- Facelets "specific" tags, <ui: … > more to come …
  - Facelets tag reference
- Possible to compose pages (modular pages) using template mechanism
  - Template page, the overall layout and style (CSS), defines where to insert content
    - Holds common parts (main menu, …)
  - Template client page
    - The content to insert
    - We say partials (as before)
  - Multilevel templating possible
  - Data transfer between template and client

Our main aim is the templating mechanism

Template Page

Set common style

```
<h:outputStylesheet library="css"
                    name="default.css" />
```

Define the slots

```
<div id="left">
   <ui:insert name="left"/>
</div>
<div id="content" class="left_content">
   <ui:insert name="content" />
</div>
```

Template file

- The overall layout and style (CSS), defines where to insert content
  - If using a top level directory “resources” possible to reference as library-attribute
- Holds common parts (menu, ...)
- Define the "named" slots, ... where to insert
- Also possible to just "include" some content (possible dynamic) compare JSP's
  - Using src attribute
- User never access template file (page) directly
  - Possible hide template below WEB-INF
  - Request URL is the client file (page)
  - Facelets will handle the composition of template and client

Template Client Page

Define content of slots

```
<ui:composition template="template.xhtml">

   <ui:define name="left">
     <!--Any left content here -->
   </ui:define>

   <ui:define name="content">
      <!-- Any content here -->
   </ui:define>
</ui:composition>
```

Define specific content to insert into slots

- Specify which template to use
- Match the name of the slot in template file

Data Transfer

Send from client page

```
<ui:composition template="template.xhtml">
     <ui:param name="age" value="1" />
 </ui:composition>
```

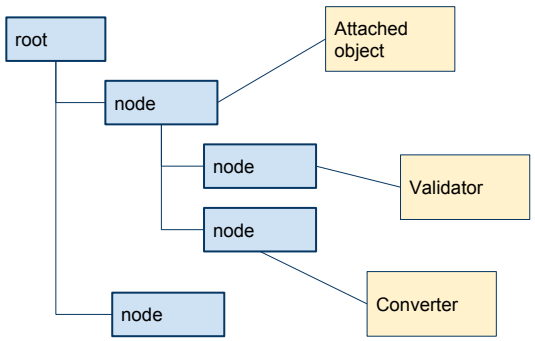
Access in template page

```
 <p>#{age}</p>
```

Possible to pass data from client to template

- Common data to be displayed on all pages (Logged in as …)

Attached Objects



Server side
Component tree

Components in server side UI tree may have attached objects, i.e. non-UIComponents attached to UIComponents

- Converters
- Validators

Incoming data must be converted and should validated before stored in UIComponents

- Normally automagically

Conversion

Standard conversion

```
<h:inputText id="date" value="#{order.date}" required="true">
     <f:convertDateTime type="time" dateStyle="long"/>
</h:inputText>
```

Custom conversion

```
@FacesConverter(value = "pnumbConverter")
public class PhoneNumberConverter implements Converter {
  public Object getAsObject(FacesContext context, UIComponent
                            component, String value) {...}
 public String getAsString(FacesContext context,UIComponent
                            component, Object value) {...}
}

<h:inputText value="..." ....>
    <f:converter converterId="pNumbConverter"/>
</h:inputText>
```

If specific needs needs

- Standard converters for: DateTime, Number, Boolean, Byte, Character, Double, Float, Integer, Long, Short, BigDecimal, BigInteger or …
- … Custom converters (classes)

Standard converters

- Using tags in pages

Custom converters

- Classes implementing Converter

Validation

Standard validation

```
<h:inputText validatorMessage="Must be 1-10 (incl)">
     <f:validateLongRange minimum="1" maximum="10"/>
</h:inputText>
```

Bean Validation (in beans)

```
@Digits(integer=2, fraction=0)
@Min(value = 1, message = "To small")
@Max(value = 10, message = "To big")
private int value;
@NotNull
@Size(min = 1, max = 8, message = "Must use 1-8 chars")
private String data;
@Pattern(regexp = "[A_Z][a-z]*")
private String pattern;
```

Principle: All layers should validate input data, validate client-side and server side

- Client side : HTML5/CSS3/JavaScript client side validation
- JEE server side: 3 different ways to validate :-(
  - We use only 2!

JSF validation

- Standard validators and …
- … custom validators (use @FacesValidator), not used in course
  - Concept as converters
- Part of JSF specification

Beans Validation a different specification

- JSF validation targeting just JSF
- Bean Validation targets "any" kind of Java application (also Java SE)
  - Works seamless with JSF, use annotations on class attribute
    - Validation errors will end up in <h:message(s)../> tag
  - Prefer …
- Bean validation very customizable (validate complex objects, validate in different stages, …

To bypass validation (and conversion) use attribute "immediate"!

- Example: Click Cancel button
- See JSF request cycle

Exceptions

In Bean

```
try {
   ...
} catch (Exception e) {
    FacesContext.getCurrentInstance()
      .addMessage(null, new FacesMessage
   ("Message..."));
}
```

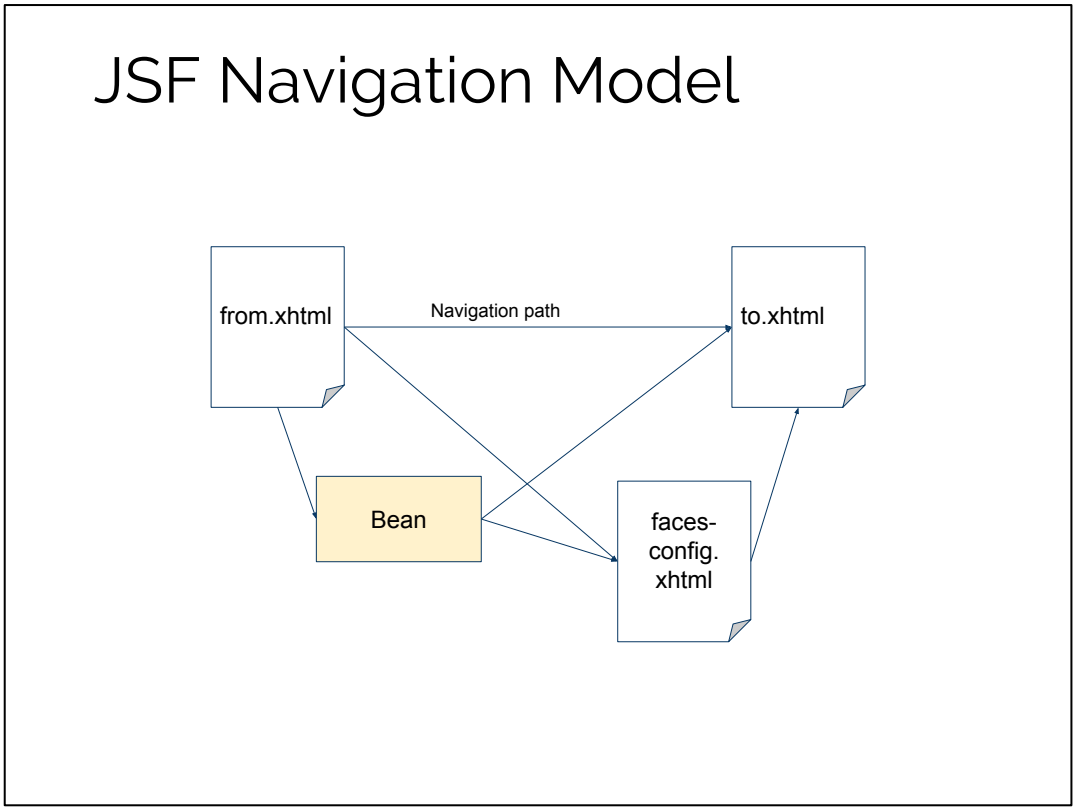
In page

```
<h:messages showSummary="true"
                      showDetail="false"/>
```

Use FacesContext to add messages

- Displayed in <h:message(s) />
  - <h:message..>, for specific component
  - <h:messages..> (ending 's') all messages
- No needs for if using Bean Validation

All messages in localized message bundles (application external)



JSF has sophisticated build in navigation model

Implicit navigation
- In pages or Java code

Rule based navigation
- Defined in faces-config.xml
- Variations
  - Static
  - Dynamic
  - Conditional

Navigation system will try to resolve all navigational cases
- If not possible produce error message (will show up in page)

Implicit Navigation

Plain link

```
<h:link value="Edit" outcome="personDetail">
```

Using action explicit

```
<h:commandButton … action="to"  ../>
```

Using action and result from bean

```
<h:commandButton … action="#{bean.navigate}" ../>
```

```
public String navigate(){
          ...
       return "to"; // If null or empty no
                    //navigation i.e. same page
}
```

Strings interpreted as filename (no xhtml suffix needed)

- Have seen this …

Rule Based Navigation

In faces-config.xml

```
<navigation-rule>
   <from-view-id>/viewProducts.xhtml</from-view-id>
       <navigation-case>
            <from-outcome>onViewCart</from-outcome>
            <to-view-id>/viewCart.xhtml</to-view-id>
        </navigation-case>
    <!-- More cases →

</navigation-rule>
```

Rules in WEB-INF/faces-config.xml
- NetBeans can generate

Rules applied in written order, first success wins
Variations
- Static, hard coded
- Dynamic, from bean method result
- Conditional, possible to avoid “first success wins”

Possible to specify redirects

Redirects and View Parameters

In Page

```
<h:commandButton action="to?faces-redirect=
                  true&amp;includeViewParams=true" ... />
```

In Bean

```
return "to?faces-redirect=true&amp;includeViewParams=true"
```

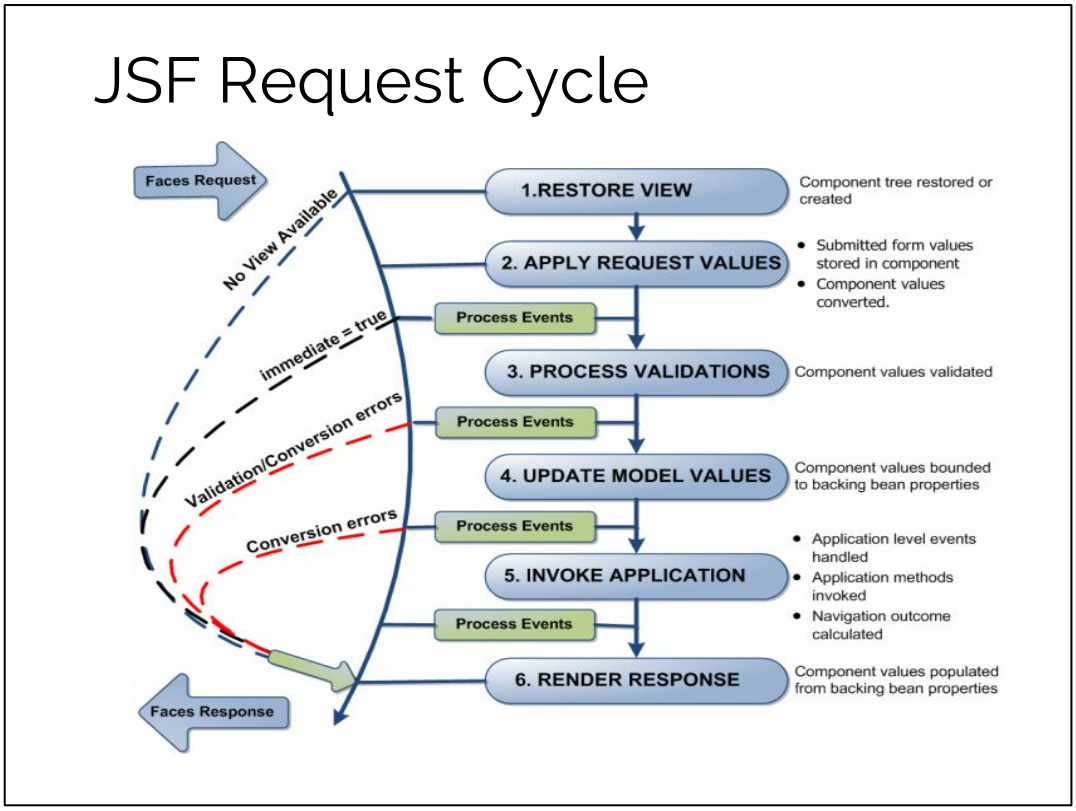
faces-config.xml

```
<navigation-case>
    <from-outcome>onViewCart</from-outcome>
    <to-view-id>/viewCart.xhtml</to-view-id>
    <redirect/>
</navigation-case>
```

Useful for PRG pattern!

Parameters don't survive redirects (default)

- Possible to alter using includeViewParams=true
- Parameters to include specified on target page!



To emulate desktop programming (but having a network in between) things must be done in an orderly fashion

- The JSF request cycle and more about the same, will handle this
- The cycle is composed of 6 phases
  - Cycle may abrupt …
- NOTE: JSF views are stateful, … possibly a problem

Initial request vs Postback

- Initial request is the first request for a page.
  - Will execute phase 1 and then 6
- Postback is a post request to the same page (NOTE: <h:form> has no action attribute)
  - Will execute full cycle

POST and GET

- Post always trigger full cycle (if not using "immediate"-attribute)
- Get only will shortcut to 1 → 6
- Get + view params will trigger full cycle

Validation phases

- JSF validation and Bean validation in different phases
  - JSF First

Browser Behaviour

Combination of state in

- client side DOM state
- server side component tree state
- beans …
- … may be very confusing.
- Do some experiments …!