15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

JSF-Hacks

Tipps und Problemlösungen mit und für JSF

Sascha Groß
Christian Beranek

Agenda

- Tipps und Tricks
  - Allgemeines
  - Konfiguration
  - Navigation
  - Komponenten und Attribute
  - Validator und Konverter
- Hacks
  - Facelets – ResourceResolver
  - JSF Komponentenbaum – TreeWalker
  - Validierung mal anderes

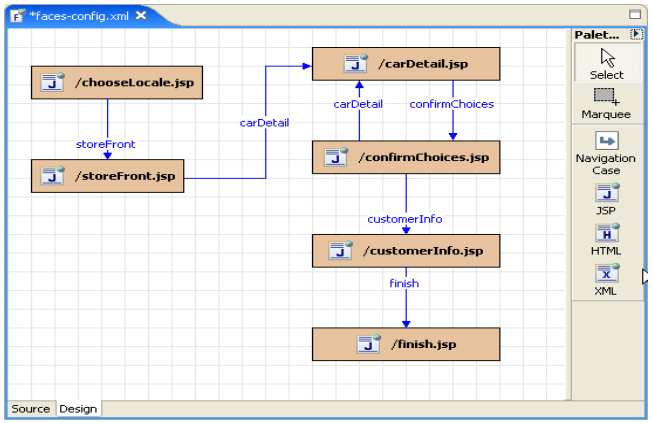
Allgemeines – IDE

- Unterstützung bei der JSF-Entwicklung durch:
  - Komponentenübersicht
  - WYSIWYG-Editoren
  - Code-Completion
    - Tags
    - Attribute
    - Bindings
  - Faces-Config
    - Erstellung von Backing Beans
    - Navigation

Allgemeines – IDE

- MyEclipse

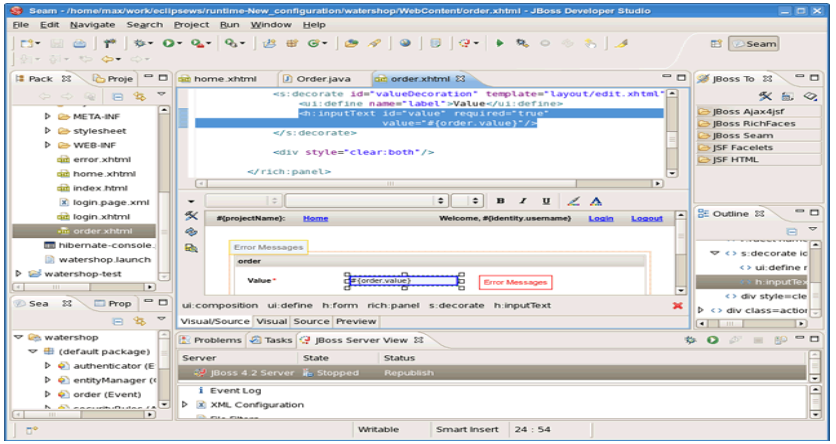
  http://www.myeclipseide.com



- JBoss Developer Studio

  http://www.jboss.com/products/devstudio

  http://www.jboss.org/tools/



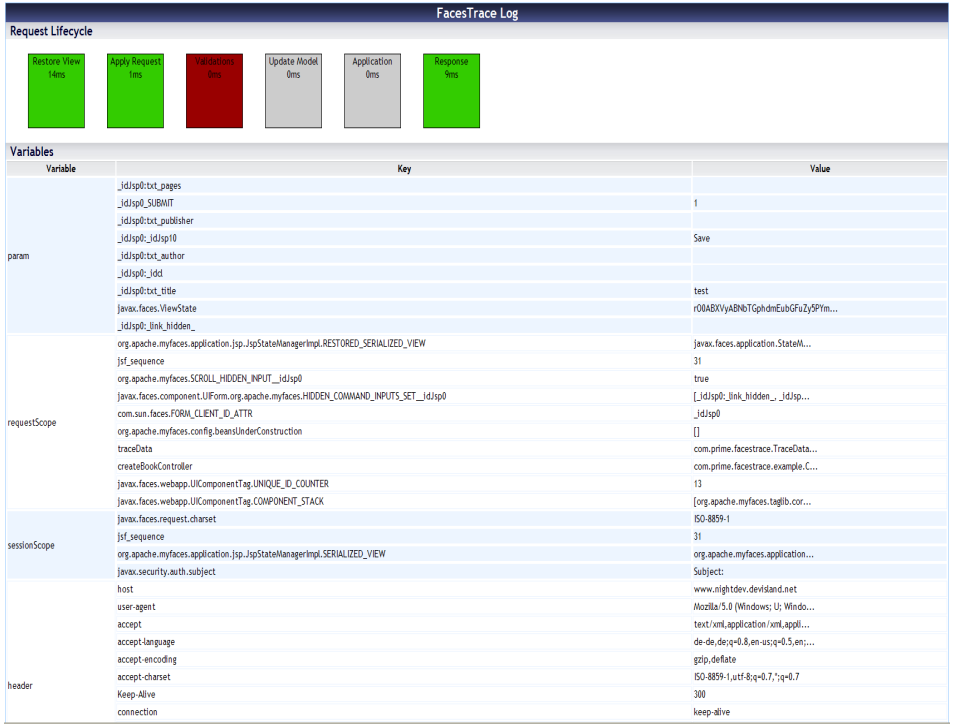
Allgemeines – Erweiterungen

- Facelets
  - https://facelets.dev.java.net/
  - XHTML-basierte Seitenentwicklung
  - Templating Framework
  - einfache Komponentenentwicklung
  - Debugging, JSTL-Support

Allgemeines – Erweiterungen

- Faces Trace
  - http://facestrace.sourceforge.net/
  - Tracing-Tool für die Analyse von JSF-Anwendungen
  - Performance-Tracker
  - Lifecycle-Tracker
  - Variables
  - Faces-Messages
  - Logs
  - Component-Tree

```
<ft:trace>
```



Allgemeines – Erweiterungen

- Apache Tomahawk
  - http://myfaces.apache.org
  - Komponentenbibliothek
  - sortierbare Tabellen, Fileupload, DataScroller, ...
- JBoss RichFaces
  - http://www.jboss.org/jbossrichfaces/
  - neue AJAX-Komponenten
  - AJAX-Erweiterungsmöglichkeit vorhandener Komponenten
- ICEfaces
  - http://www.icefaces.org/
  - viele AJAX-Komponenten: Menüs, DnD, Autocompletion

Konfiguration – Faces-Config

- Splitten der Faces-Config
- Auslagern der Navigation, Beans

```
<web-app>
…
  <context-param>
       <param-name>javax.faces.CONFIG_FILES</param-name>
       <param-value>WEB-INF/navigation.xml,
            WEB-INF/beans.xml</param-value>
  </context-param>
…
</web-app>
```

Konfiguration – Message Bundles

- zentrales Auslagern der Messagestrings
  - Übersichtlichkeit
  - Lokalisierung der Anwendung
- JSF 1.1:

messages.properties:

```
sampleMessage=Beispielausgabe
```

```
<f:loadBundle basename="de.mathema.messages" var="msg" />
<h:outputText value="#{msg.sampleMessage}"/>
```

Konfiguration – Message Bundles

- seit JSF 1.2

```
<application>
  <resource-bundle>
    <base-name>de.mathema.messages</base-name>
    <var>msg</var>
  </resource-bundle/>
</application>
```

- Lokalisierung/Internationalisierung mittels:

```
de/mathema/messages_de.properties (ISO-639)
```

```
<f:view locale="de"/>
```

```
FacesContext.getCurrentInstance().getViewRoot().setLocale();
```

Konfiguration – Message Bundles

```
<faces-config>
  <application>
    <locale-config>
       <default-locale>de</default-locale>
       <supported-locale>en</supported-locale>
    </locale-config>
  </application>
</faces-config>
```

Konfiguration – Messages

Variable Messages

```
<h:outputFormat value="#{msg.score}">
  <f:param value="#{AccoutBean.score}" />
</h:outputFormat>
```

```
score=Ihr Kontostand beträgt
 {0,choice,0#Null Punkte|1#Einen Punkt|2#{0} Punkte}
```

Konfiguration - ErrorHandling

keine unschönen Stracktraces im Fehlerfall

```
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/errorPage.jsp</location>
</error-page>
<error-page>
  <error-code>404</error-code>
  <location>/pageNotFound.jsp</location>
</error-page>
```

Vorsicht, wenn Fehler auf der Fehlerseite auftritt!

Konfiguration – ErrorHandling

Fehlerseite

```
<body>
  <f:form>
    <p>In der Anwendung ist ein schwerwiegender Fehler aufgetreten</p>
    <p>Detaillierte Fehlermeldung für technischen Support:</p>
    <h:inputTextarea value="#{errorBean.stackTrace}"/>
  </f:form>
</body>
```

```
Map<String, Object> request =
  context.getExternalContext().getRequestMap();
request.get("javax.servlet.error.exception");
```

Navigation

```
<navigation-rule>
  <from-view-id>/xhtml/application/resul .xhtml</from-view-id>
  <navigation-case>
    <from-outcome>reload</from-outcome>
    <to-view-id>/xhtml/application/res .xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- Seitenreload mit **`null`** als return-Wert in Actionmethode

- Verwendung von Wildcards

```
<from-view-id>/registration/*</from-view-id>
```

- Problem: mehrere separate Actions mit dem gleichen String

```
<from-action>#{sampleBean.loginAction}</from-action>
```

Navigation

- JSF 1.1: ReturnType `String` verpflichtend für Action-Methoden
- seit JSF 1.2: `Enum` als ReturnType möglich

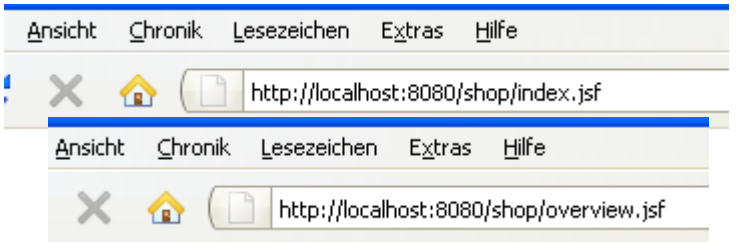
```
<h:commandButton action="#{sampleBean.sampleAction}"
  value="Absenden">
```

```
public Object sampleAction() {
   return NavigationEnum.test;
}
```

Navigation

- Bookmarking wichtiger Seiten
- Redirection-Element in der Navigation =>
  Browser erhält Http-Redirect =>
  Adressfeld wird aktualisiert

```
<navigation-case>
  <from-outcome>overview</from-outcome>
  <to-view-id>/overview.jsp</to-view-id>
  <redirect/>
</navigation-case>
```



- Daten im Request-Scope gehen verloren!!!

Komponenten und Attribute

Conditional Rendering:

- Ein- und Ausblenden bestimmter Elemente
  - Verwendung der JSTL-Tags `<c:if>`
  - Besser: Verwendung des **`rendered`**-Attributs

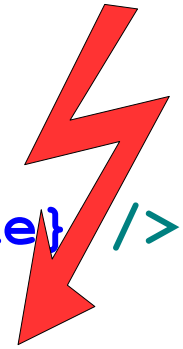
```
<h:outputText value="#{sampleBean.versionInfo}"
  rendered="#{userBean.loggedIn}"/>
```

  - Funktioniert auch für Blöcke und mit Operatoren.

```
<h:panelGrid rendered='#{sampleBean.selectedTab == "help"}'/>
```

Komponenten und Attribute

```
<h:dataTable values="${value}"
 var="var">
  <h:column>
    <c:if test="${var < 10}">
      <h:outputText value="${value}" />
    </c:if>
  </h:column>
</h:datatable>
```



Hier rendered verwenden!

Komponenten und Attribute

- seit JSF 1.2: (Initialisierung)

```
<f:view beforePhase="#{b.method}">

public void beforePhase(PhaseEvent
 e){
    ...
 }
```

- **immediate** für Cancel und Reset-Buttons, um die Validierung zu umgehen

Valididator und Konverter

- JSF verwendet standardmäßig bei der Konvertierung GMT-Zeit 

```
<f:convertDateTime type="date" pattern="dd.MM.yyyy"
  timeZone="CET"/>
```

- Alternativ: eigenen DateTimeConverter schreiben

```
public class MyDateTimeConverter extends DateTimeConverter {
public MyDateTimeConverter() {
  super();
   //Set Timezone
   setTimeZone(TimeZone.getDefault());
   //setPattern("M/d/yy");
}
```

Valididator und Konverter

- MyDateTimeConverter als DefaultConverter für Objekte vom Typ java.util.Date registrieren

```
<converter>
  <converter-for-class>java.util.Date</converter-for-class>
  <converter-class>de.mathema.MyDateTimeConverter</converter-
    class>
</converter>
```

Validator und Konverter

- Custom Error Messages
- Die Standard-Fehlermeldungen sind in der Datei Messages.properties im package javax.faces des jsf-impl.jar-Files definiert.

```
javax.faces.component.UIInput.REQUIRED={0}: Validation Error: Value
  is required.
```

- Durch Verwendung des jeweiligen Key in einer eigenen Properties-Datei können die Meldungen angepasst werden.

```
javax.faces.component.UIInput.REQUIRED={0}: Bitte Wert eingeben.
```

```
<application>
  <message-bundle>de.mathema.messages</message-bundle/>
</application>
```

Validator und Konverter

Problem: Navigation funktioniert (scheinbar) nicht.

`<h:messages>`-Element einfügen

- Seit JSF 1.2 gibt es neue Attribute:
  - converterMessage
  - validatorMessage
  - requiredMessage

Facelets

- `ViewHandler` Technologie
  - Kein JSP, sondern XML/XHTML
  - Einfacheres Erstellen von Komponenten
    - Auch ohne Programmierung
  - Templating
  - EL überall
  - Kein `<f:verbatime/>`
  - ...
- https://facelets.dev.java.net
- **Verwenden**

Facelets - `ResourceResolver`

- Interface `ResourceResolver`

  ```
  java.net.URL resolveUrl(java.lang.String path);
  ```

  - Liefert die URL der Resource
  - Default-Implementierung:

    `com.sun.facelets.impl.DefaultResourceResolver`

- Laden der Resourcen (*.xhtml) aus JAR
  - Vorteil: Single Source
  - Sinnvoll für Template, ansonsten Komponenten erstellen
- Laden der JSF-Seiten aus webapp-Verzeichnis des Entwickler
  - Schnellere Entwicklung ohne neues Deployment
- Laden der JSF-Seiten aus Datenbank (würde ich nicht machen)
- ...

Facelets – ResourceResolverImpl

```
public class ResourceResolverImpl implements ResourceResolver {

    private static final ResourceResolver
                    DEFAULT_RESOURCE_RESOLVER = new DefaultResourceResolver();

    public URL resolveUrl(String path) {
        URL url = DEFAULT_RESOURCE_RESOLVER.resolveUrl(path);

        if (url == null) {
            return resolveUrlFromJar(path);
        } else {
            return url;
        }
    }


    private URL resolveUrlFromJar(String path) {
        return Thread.currentThread().getContextClassLoader().getResource(path);
    }

}
```

Facelets – `ResourceResolver` Konfiguration in `web.xml`

```
<web-app>
  ...
  <context-param>
    <param-name>facelets.RESOURCE_RESOLVER</param-name>
    </param-value>.....ResourceResolverImpl</param-value>
  </context-param>
  <context-param>
    <param-name>facelets.REFRESH_PERIOD</param-name>
    </param-value>2</param-value>
  </context-param>
  ...
</web-app>
```

JSF Komponentenbaum

- JSF ist ein Baum
- Bäume sind zum Manipulieren da
  - Hinzufügen/Entfernen von Knoten (Komponenten)
  - Ändern von Knoteneigenschaften (Attribute von Komponenten)
- JSF Komponentenbaum ist bidirektional
  - `getChildren()`, `getFacetsAndChildren()`
  - `getParent()`
- JSF selbst nutzt Baumeigenschaft für Durchlaufen des Baums
  - `processDecodes()`, `processRestoreState()`, ...

JSF Komponentenbaum

- Informationen aus Komponentenbaum gewinnen
  - Erste editierbare Komponente im Baum
  - Erste Komponente mit Validierungsfehler (`isValid() == false`)
  - ...
- Komponentenbaum manipulieren
  - Hinzufügen von JavaScript
    - Aktuelle Eingabe-Komponente (`<h:inputText/>`, ...) hervorheben (`onblur, onfocus`)
    - ...
  - Anzeige von Fehlern
    - `<h:message/>` dynamisch in Baum hinzufügen
    - Eingabefelder mit falschen Benutzereingaben kennzeichnen, z. B rot umranden

JSF Komponentenbaum – `TreeWalker`

- `TreeWalker`
  - Durchlaufen des Baums
- `Command`
  - Ausführen eines Kommandos aus den aktuellen Knoten
  - Interface
- `PhaseListener`
  - Aufruf des `TreeWalker` mit `Command`
- `BeforeViewHandlerRenderViewPlugin`
  - Aufruf des `TreeWalker` mit `Command`
  - Plugin für `PlugableFaceletViewHandler`
    - Erweitert `FaceletViewHandler`
    - Erlaubt nachdem der Baum erstellt wurde, etwas auszuführen

JSF Komponentenbaum – TreeWalker

```
public final class TreeWalker {
  public static void walk(
    UIComponent component, Command command ) {

    if( component == null || command == null ) {
      return;
    }
    try {
      TreeWalker.walkChild( component, command );
    } catch( AbortException wae ) {
      // is OK, stop walking
    }
  }
  private static void walkChild(
    UIComponent component, Command command ) throws AbortException {

    try {
      command.execute( component );
    } catch( AbortChildWalkException wace ) {
      return;
    }
    for( Iterator it = component.getFacetsAndChildren(); it.hasNext(); ) {
      TreeWalker.walkChild( it.next(), command );
    }
  }
}
```

JSF Komponentenbaum – TreeWalker – Command

```
public void execute(
 UIComponent component
 ) throws AbortException,
          AbortChildWalkException;
```

JSF Komponentenbaum – TreeWalker – Command

```
public class AddValidatonLayoutCommand implements Command {

    private static final String ERROR_STYLE_CLASS =
                                        "validationErrorClass ";

    public void execute( UIComponent component ) { {
        try {
            EditableValueHolder editableValueHolder =
                            (EditableValueHolder) component;
             if( ! editableValueHolder.isValid() ) {
                 component.getAttributes().put(
                            "styleClass", ERROR_STYLE_CLASS );
             }
        } catch( ClassCastException cce ) {
            return;
        }
    }
}
```

JSF Komponentenbaum – TreeWalker – PhaseListener

```
public class AddValidationLayoutPhaseListener implements PhaseListener {

    private static final Command COMMAND =
                          new AddValidatonLayoutCommand();

    public void afterPhase( PhaseEvent phaseEvent ) {
        //do nothing
    }

    public void beforePhase( PhaseEvent phaseEvent ) {
        TreeWalker.walk(
            phaseEvent.getFacesContext().getViewRoot(), COMMAND );
    }

    public PhaseId getPhaseId() {
        return PhaseId.RENDER_RESPONSE;
    }

}
```

JSF Komponentenbaum – `TreeWalker` – Konfiguration

- **faces-config.xml**

```
<application>
   <view-handler>
   de.mathema.web.jsf.util.viewhandler.PlugableFaceletViewHandler
   </view-handler>
</application>
```

- **web.xml**

```
<context-param>
  <param-name>
      de....PlugableFaceletViewHandler.PARAM_BEFORE_VIEW_HANDLER_RENDER_VIEW_PLUGIN
  </param-name>
  <param-value>
de.mathema.web.jsf.treewalker.command.layout.BoldPlugin
  </param-value>
</context-param>
```

Validierung

- JSF bietet Validierung
  - required
  - Standardvalidatoren <f:valdiateLength/>, <f:validateDoubleRange/>, ...
  - Erweiterbar um eigene Validatoren und Validierungsmethoden
  - Converter

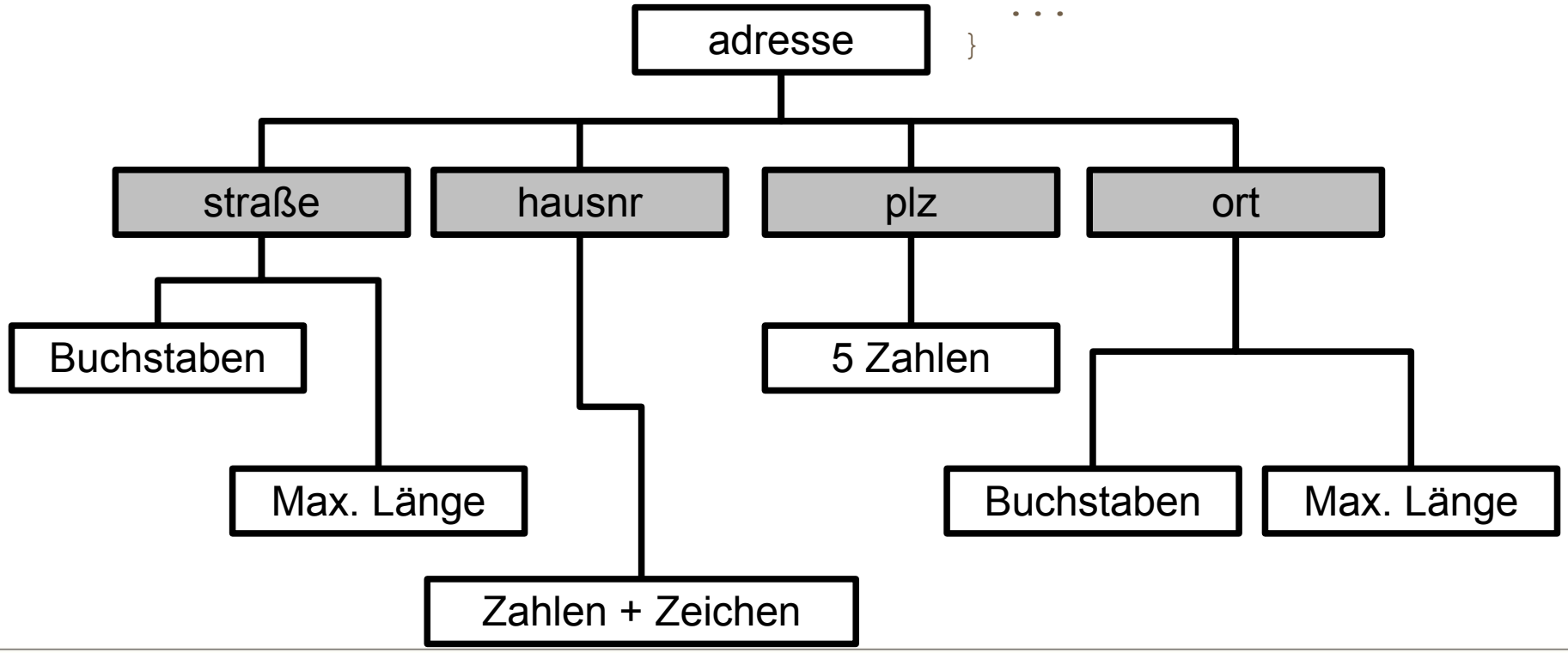
- Unterstützte Validierung beschränkt sich auf Typ und Bereichsvalidierung
- Crossvalidierung möglich aber aufwendig und unschön
  - z.B Adresse bestehend aus Straße, Hausnummer, PLZ und Ort

- ==> Validierung in Action oder nach Phase 4 Update Model Values
  Widerspricht Reinheitsgebot des Model

Validierung – Mal anders

- Validierung als Baum
  - in Wirklichkeit azyklischer Graph

```
public class Adresse {
    private String straße;
    private String hausnr;
    private String plz;
    private String ort;
    ...
}
```



Validierung – Mal anders

```
<h:panelGrid column="2">
    <h:outputLabel value="Adressvalidierung überspringen"
                   for="skipadressvalidierung"/>
    <h:selectBooleanCheckbox id="skipadressvalidierung"
                   value="#{model.skipAdressValidierung}"/>

    <h:outputLabel value="Straße" for="straße"/>
    <h:inputText id="straße" value="#{model.adresse.straße}">
        <v:validator id="v_straße_required"
                   validator="RequiredValidator"/>
        <v:validator id="v_straße"
                   validator="TextValidator" pattern="[a-ZßöäüÖÜÄ]*"/>
    </inputText>

    <h:outputLabel value="Hausnr" for="hausnr"/>
    <h:inputText id="hausnr" value="#{model.adresse.hausnr}">
        <v:validator id="v_hausnr_required"
                   validator="RequiredValidator"/>
        <v:validator id="v_hausnr"
                   validator="TextValidator" pattern="[0-9]{1,3}[a-Z]"/>
    </inputText>

    ...
</h:panelGrid>
```

Validierung – Mal anders

```
<v:validator
    active="true"
    messageFor="straße"
    refs="v_straße, v_hausnr, v_plz, v_ort"
    validator="AdresseValidator"

    person="#{model.person}"
    skipAdressValidierung="#{v:componentValue('skipadressvalidierung')}"
    straße="#{v:validatorComponent( 'v_straße' ) }"
    hausnr="#{v:validatorComponent( 'v_hausnr' ) }"
    plz="#{v:validatorComponent( 'v_plz' ) }"
    ort="#{v:validatorComponent( 'v_ort' ) }"
/>
```

Validierung – Mal anders

- Validatoren als Komponenten (keine JSF Validatoren)
  - Meist Kinder von `EditableValueHolder` (`<h:inputText/>`, ...)
  - Auf ersten Blick wie Validatoren
- Validatoren registrieren in `processValidators()`
- `submittedValue` zwischenspeichern
- Aufbau des Validierungsbaums in Phase 3 `afterPhase()`
- Validierung des Baums in Phase 3 `afterPhase()`
  - Von unten nach oben (von den Blättern zur Wurzel)
- Komponenten invalidieren `setValid()` und `submittedValue` zurückschreiben
- Meldungen zu `FacesContext` hinzufügen `addMessage()`

Ressourcen

- MyEclipse
  - http://www.myeclipseide.com
- JBoss Developer Studio
  - http://www.jboss.com/products/devstudio
  - http://www.jboss.org/tools/
- Facelets
  - https://facelets.dev.java.net
- Faces Trace
  - http://facestrace.sourceforge.net/
- Apache Tomahawk
  - http://myfaces.apache.org
- JBoss RichFaces
  - http://www.jboss.org/jbossrichfaces/
- ICEfaces
  - http://www.icefaces.org/
- JSF Anti-Patterns and Pitfalls
  - http://go.techtarget.com/r/4406589/2913420

Vielen Dank!

Christian Beranek
Sascha Groß