Enterprise Web Development (Java Enterprise Edition 6)

wissen.leben
WWU Münster

Dozent: Arne Scheffer

JSF Custom Components

- mit Java eigene Tags schreiben
  - etwas flexibler als Composite Component Tags
- am besten vorhandene UI-Klassen beerben
  - z.B. *UIOutput*
- Render-Methoden erweitern
  - *encodeBegin*
  - *encodeChildren*
  - *encodeEnd*
- es gibt zu diesen Klassen Möglichkeiten eigene Renderer-Klassen zu definieren
- zentrale Annotation *@FacesComponent(value="...")*
  - alternativ in *faces-config.xml*

  - Beispiel: ul-Rahmen nur ausgeben, wenn -Einträge vorhanden sind
    - andernfalls nicht xhtml-konform

Unser Beispiel: ulFrame als Erweiterung von ui:repeat

```
@FacesComponent(value="ulFrame")
public class UIul extends UIRepeat {
    @Override
    public void encodeBegin(FacesContext facesContext) throws IOException {
        ResponseWriter writer = facesContext.getResponseWriter();
        if ((getChildCount()>0) && (getValue() instanceof ListDataModel) &&
            ((ListDataModel) getValue()).getRowCount()>0
           )
        { writer.startElement("ul", null); }
        else
        { writer.startElement("span", null); }
        writer.writeAttribute("id", getId(), null);
        writer.writeAttribute("class", this.getAttributes().get("listClass"),null);
        writer.writeAttribute("style", this.getAttributes().get("style"),null);
        super.encodeBegin(facesContext);
    }
...
```

Unser Beispiel: ulFrame als Erweiterung von ui:repeat

- encodeEnd() analog zu encodeBegin():

```
…
    @Override
    public void encodeEnd(FacesContext facesContext) throws IOException {
        super.encodeEnd(facesContext);
        if ((getChildCount()>0) &&
           (getValue() instanceof ListDataModel) &&
            ((ListDataModel) getValue()).getRowCount()>0
           )
        { facesContext.getResponseWriter().endElement("ul"); }
        else
        { facesContext.getResponseWriter().endElement("span"); }
    }
}
```

JSF Custom Components: Registrierung

- Binden der Tags an einen Namespace, um diese nutzen zu können
  - dafür Definition einer eigenen Taglib
    - in *web.xml* (liegt in *WEB-INF*):

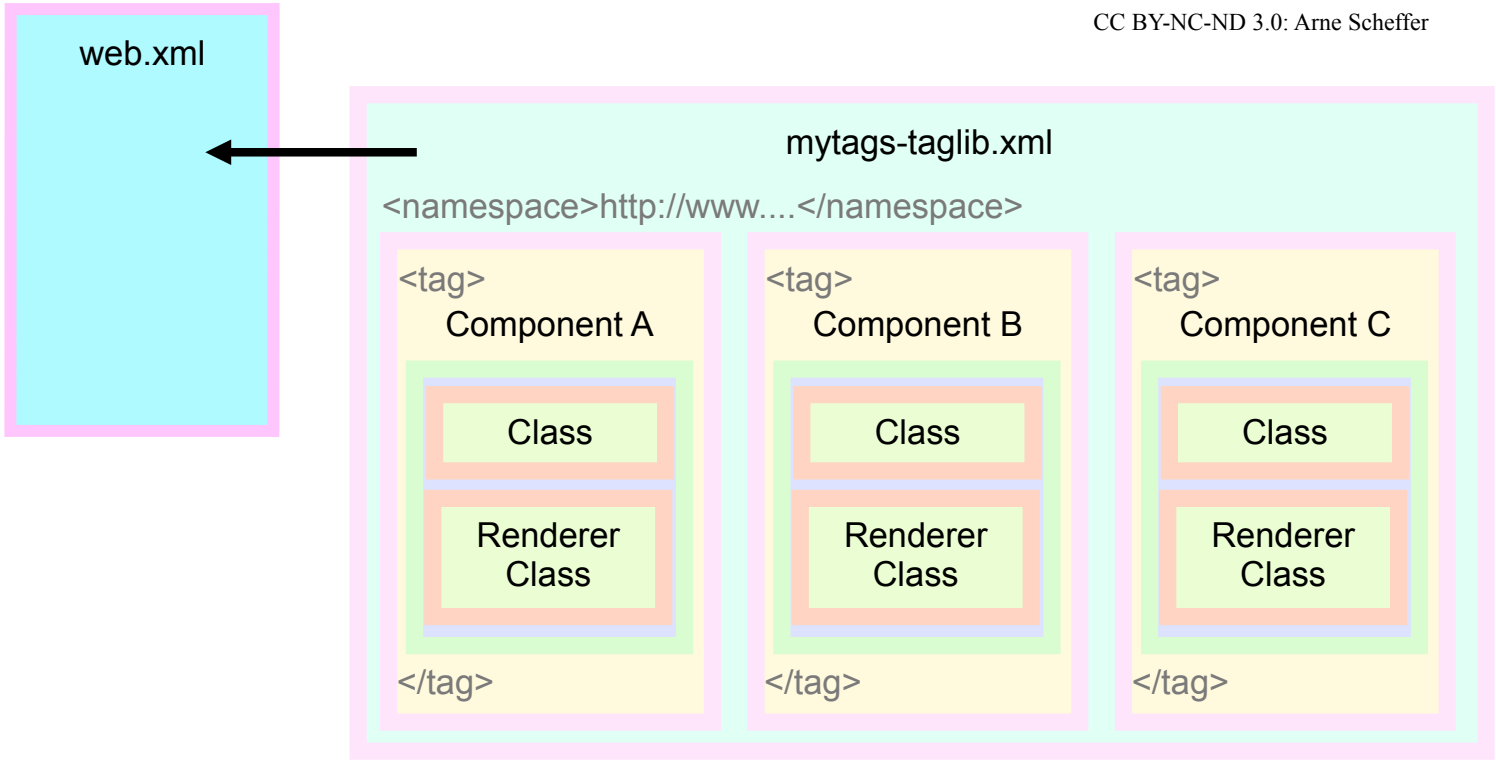
```
<web-app ...
   <context-param>...</context-param>
   <context-param>
      <param-name>javax.faces.FACELETS_LIBRARIES</param-name>
      <param-value>
       /WEB-INF/mytags-taglib.xml
      </param-value>
   </context-param>
   <servlet>...
```

JSF Custom Components: Custom Tag Library

- Datei *mytags-taglib.xml* in *WEB-INF*:

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
  "-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
  "http://java.sun.com/dtd/facelet-taglib_1_0.dtd">

<facelet-taglib>
    <namespace>http://www.uni-muenster.de/jsf/util</namespace>
    <tag>
        <tag-name>ulFrame</tag-name>
        <component>
            <component-type>ulFrame</component-type>
            <renderer-type>ulFrame.renderer</renderer-type>
        </component>
    </tag>
</facelet-taglib>
```

Die eigene Tag-Library



Verwendung von *ulFrame*: in *linkliste.xhtml*

```
...
   xmlns:ownfaceletlib="http://www.uni-muenster.de/jsf/util">
<cc:interface>...</cc:interface>
<cc:implementation>
  <span id="#{cc.attrs.id}"><h:form id="form">
      <ownfaceletlib:ulFrame id="ulFrame" style="list-style: none" listClass="#{cc.attrs.listClass}"
                                          value="#{cc.attrs.liste}"        var="listeneintrag">
       <li><ez:vieweditdestroypanel rendered="#{cc.attrs.edit}"/>
        <h:outputText value=": " rendered="#{cc.attrs.edit}"/>
        <h:outputLink value="#{listeneintrag.linkUrl}">
  <f:ajax disabled="#{!cc.attrs.ajaxEdit}" event="mouseover" listener="#{linklisteController.prepareEdit}"
     render=":editForm:linkName :editForm:linkUrl"/>
  <f:ajax disabled="#{!cc.attrs.ajaxView}" event="mouseover" listener="#{linklisteController.prepareView}"
     render=":viewForm:linkName :viewForm:linkUrl :viewForm:linkId"/>
         <h:outputText value="#{listeneintrag.linkName}" /></h:outputLink>
       </li>
      </ownfaceletlib:ulFrame></h:form></span></cc:implementation>
```

Verwendung von *ulFrame*: in einer CC *ul.xhtml*

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:cc="http://java.sun.com/jsf/composite"
      xmlns:ownfaceletlib="http://www.uni-muenster.de/jsf/util">
<!-- INTERFACE -->
<cc:interface>
      <cc:attribute name="liste"/>  <cc:attribute name="listClass"/> <cc:attribute name="style"/>
</cc:interface>
<cc:implementation>
   <ownfaceletlib:ulFrame listClass="#{cc.attrs.listClass}" id="#{cc.attrs.id}"
                                    style="#{cc.attrs.style}"  value="#{cc.attrs.liste}" var="entry" >
    <li><cc:insertChildren /></li>
   </ownfaceletlib:ulFrame>
 </cc:implementation>
</html>
```

Aufruf der CC *ul.xhtml*, Möglichkeiten der Inhaltsübergabe

- Aufruf z.B. mit:

```
<ez:ul id="myid" listClass="myclass"
        style="list-style-type: none"
        liste="#{linklisteController.items}" >
  <h:outputLink value="#{entry.linkUrl}">
   <h:outputText value="#{entry.linkName}"  />
  </h:outputLink>
</ez:ul>
```

- möchte man den Inhalt innerhalb eines CC-Tags z.B. für Validatoren nutzen
  - *f:validate*.... (Länge einer Variablen sicherstellen etc.)
  - kann man *f:facet* nutzen, um Inhalt zu übergeben
    - und statt *cc:insertChildren cc:renderFacet* verwenden

Anforderungen der BITV

- Barrierefreie Webseiten für den öffentlichen Dienst sind (sinnvolle) Pflicht
  - vieles liegt in den Händen des Webdesigners
  - Bedienbarkeit ohne Javascript muss möglich sein
    - Auswahl der Schaltflächen-Komponenten, z.B.:
      - Ersetzung aller *CommandLink*-Tags (Javascript-Funtionalität) durch
        - *CommandButton*
        - *Link*, *OutputLink*

          (in unserer Beispielanwendung geschehen)
      - evtl. Ergänzung um *<f:ajax>*-Funktionalität
    - Abschalten von Sichtbarkeiten ist programmatisch durchzuführen
    - Trennung von Javascript- und Webseiten-Code sinnvoll
      - Javascript-Client-Bibliotheken später austauschbar

Abschalten von Sichtbarkeiten mittels Javascript

- Aktivierungen per Javascript machen die Schaltflächen sonst unbenutzbar
- beim Laden der Seite z.B.

```
/* <![CDATA[ */
document.write('<style type="text/css">.initialnotvisible{display:none}<\/style>\n');
/* ]]> */
```

in den Kopf einbauen

- bei uns später als Datei *initialnotvisible.js* geladen aus *resources/jslibs*
- dieses ist z.B. möglich per Tag: <h:outputScript … target="head" />

ClientBehavior-API

- Idee:
  - Trennung:
    - durch Javascript im Client induziertes Verhalten ↔ Facelet-Code

- mit *<f:ajax>* ist ein prominenter Vertreter der API implementiert:

  - Formularcode wird - derart getagt -
    - unterstützt durch Javascript und Ajax ausgeführt:
      - nur Seitenfragmente werden transportiert (s. Vorlesung dazu)
  - schaltet man Javascript ab (oder entfernt das *<f:ajax>*-Tag):
    - bleibt die normale Formular-Funktionalität erhalten:
      - die gesamte Seite wird dafür neu geladen

Einfach(st)es Beispiel: *SimplestBehavior.java*

- Verwendung einer mit *@FacesBehavior* annotierten Behavior-Klasse
  - zur Aufnahme des Javascript-Codes
    - als Return-Wert der Methode *getScript*

- package FaceletBehaviors;
  … // Imports

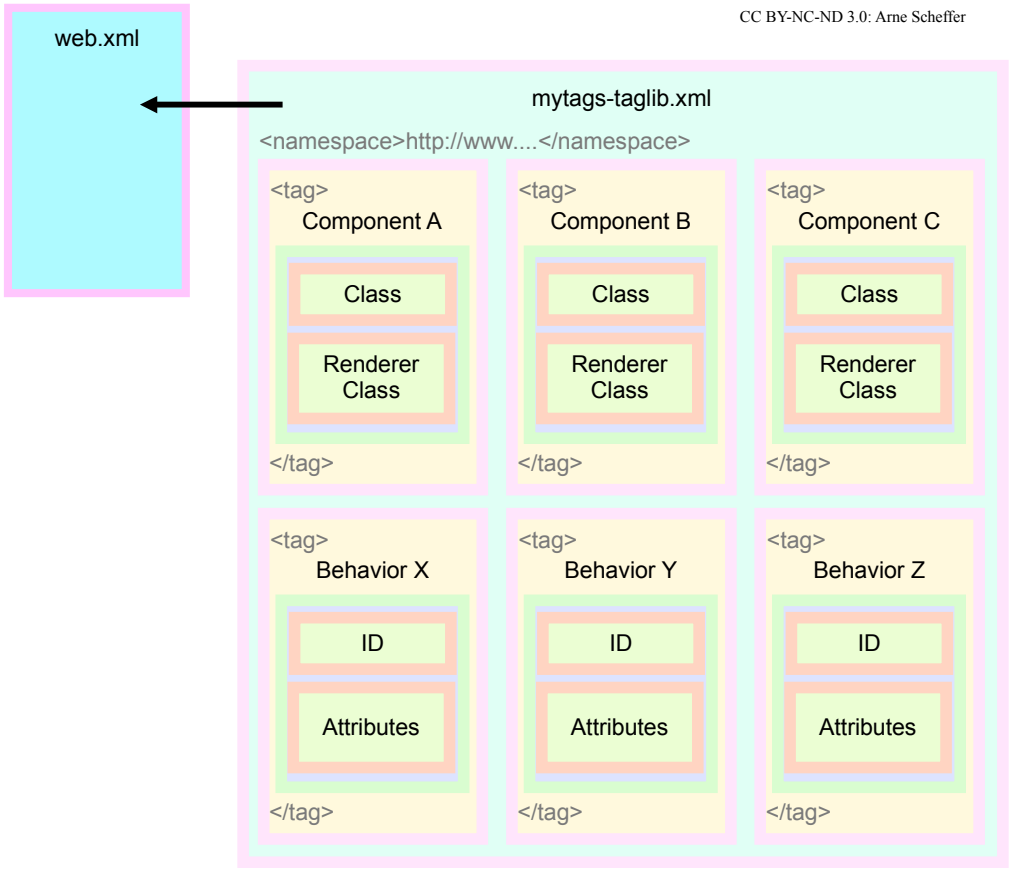
```
@FacesBehavior("FaceletBehaviors.simplestBehavior")
public class SimplestBehavior extends ClientBehaviorBase {

  @Override
  public String getScript(ClientBehaviorContext behaviorContext) {
   return "alert('Simplest Behavior!'); return false";
  }

}
```

Einfach(st)es Beispiel: Ergänzung unserer JSF-Tag-Library

- bei uns die Datei *mytags-taglib.xml* in *WEB-INF* (registriert in *web.xml*)

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC  "-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
  "http://java.sun.com/dtd/facelet-taglib_1_0.dtd">

<facelet-taglib>
    <namespace>http://www.uni-muenster.de/jsf/util</namespace>
   <!-- .... evtl. weitere Tags -->
   <tag>
       <tag-name>simplestBehavior</tag-name>
       <behavior>
          <behavior-id>FaceletBehaviors.simplestBehavior</behavior-id>
       </behavior>
    </tag>
</facelet-taglib>
```

Die eigene Tag-Library mit Behavior-Tags



Einfach(st)es Beispiel: Composite Component & Aufruf

- CC *SimplestComponent.xhtml*:

```
...
 <!-- INTERFACE -->
 <cc:interface>
    <cc:clientBehavior name="mouseover" event="mouseover" targets="opLink" />
 </cc:interface>
 <!-- IMPLEMENTATION -->
 <cc:implementation>
  <h:outputLink id="opLink" value="http://www.wwu.de">zur WWU</h:outputLink>
 </cc:implementation>
…
```

- Aufruf:

```
<html ... xmlns:ownfaceletlib="http://www.uni-muenster.de/jsf/util" ...>
...   <ez:simplestComponent >
        <ownfaceletlib:simplestBehavior event="mouseover" />
      </ez:simplestComponent>  ...
```

Beispiel 2: Einbau in die Linklistenanwendung

- Ziel: Buttons neben dem Link sollen nur bei *mouseover* angezeigt werden
- dafür sind zudem
  - die Buttons zu Beginn auszublenden
    - siehe gezeigtes Javascript
  - die Buttons bei *mouseout* auszublenden
- Trick: um keinen Bruch zu erzeugen beim Zeiger-Übergang Link → Buttons
  - werden die Links mit *mousover/-out*- Steuerung versehen
  - werden die Buttons mit *mouseover/-out*-Steuerung versehen
  - bleibt das erste Leerzeichen von den Button immer „sichtbar“

  - ansonsten
    - verschwinden die Buttons, wenn der Zeiger die Linkfächen verlässt
    - erscheinen nicht wieder, da mouseover nur für sichtbare Flächen funktioniert

Beispiel 2: Die Behavior-Basisklasse

```
package FaceletBehaviors;
// … Imports
@ResourceDependencies({
 @ResourceDependency(name="jquery-1.5.min.js", library="jslibs", target="head"),
 @ResourceDependency(name="initialnotvisible.js", library="jslibs", target="head")
})
public abstract class JQueryBehavior extends ClientBehaviorBase {
 protected String target    = null;
 protected Boolean disabled = false;
 protected Boolean chained  = false;

 // … triviale Getters und Setters für target, disabled und chained: durch IDE erzeugen lassen

  protected String getjQueryID (ClientBehaviorContext behaviorContext){
    // siehe Folgeseite.
  }
}
```

Beispiel 2: Methode *getjQueryID* der Behavior-Basisklasse

```
protected String getjQueryID (ClientBehaviorContext behaviorContext){
 FacesContext faces_context       = behaviorContext.getFacesContext();
 UIComponent parentComponent = behaviorContext.getComponent();
 String targetId                          = parentComponent.getClientId();

 if (target != null)
 {
  UIComponent targetComponent = parentComponent.findComponent(target);
  if (targetComponent != null)
          targetId = targetComponent.getClientId(faces_context);
   else  throw new FacesException("Kann target nicht finden \""+ target + "\"");
 }
     /* Der Doppelpunkt ist grundsätzlich, da JSON-Trenner, zu escapen
      * normalerweise 4 Schrägstriche = 1*escapen für Java, 1*escapen für Javascript */
 return targetId.replace(":", "\\\\:");
}
```

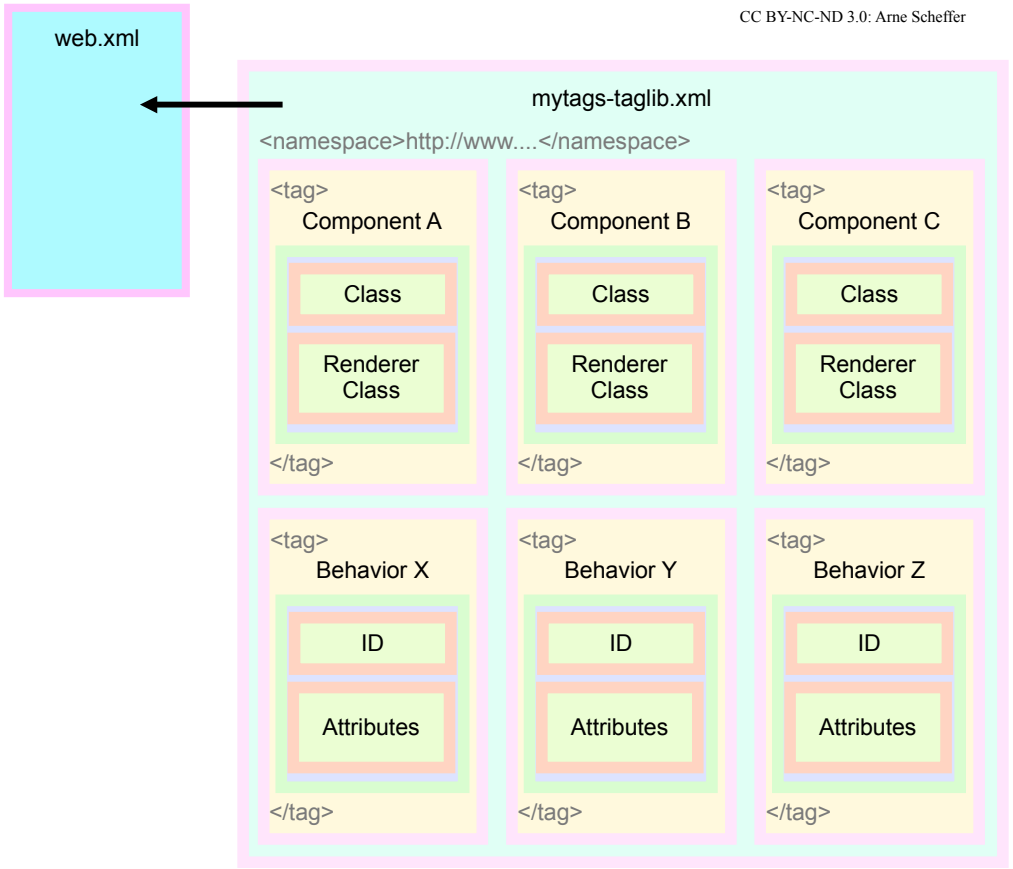
Beispiel 2: *ShowBehavior* beim Einblenden (z.B. bei *mouseover*)

```
package FaceletBehaviors;
// … Imports
@FacesBehavior("FaceletBehaviors.show")
public class ShowBehavior extends JQueryBehavior {

  @Override
  public String getScript(ClientBehaviorContext behaviorContext){
    if (getTarget().startsWith("."))
    return "$('"  + getTarget() + "').show(); return false";     // Klasse anzeigen
   else
    return "$('#" + getjQueryID(behaviorContext) + "').show(); return false";
  }
}
```

Beispiel 2: Hide*Behavior* beim Ausblenden (z.B. bei *mouseout*)

```
package FaceletBehaviors;
// … Imports
@FacesBehavior("FaceletBehaviors.hide")
public class HideBehavior extends JQueryBehavior {

  @Override
  public String getScript(ClientBehaviorContext behaviorContext){
    if (getTarget().startsWith("."))
    return "$('"  + getTarget() + "').hide(); return false";     // Klasse anzeigen
   else
    return "$('#" + getjQueryID(behaviorContext) + "').hide(); return false";
  }
}
```

Registrierung wie gewohnt...



Beispiel 2: Registrierung; *tag*-Einträge in *mytags-taglib.xml*

```
 <tag>
    <tag-name>hide</tag-name>
    <behavior><behavior-id>FaceletBehaviors.hide</behavior-id></behavior>
    <attribute>
       <name>target</name>
       <description>Ziel der Javascriptmodifikation</description>
       <type>java.lang.String</type>
       <required>true</required>
    </attribute>
</tag><tag>
    <tag-name>show</tag-name>
    <behavior><behavior-id>FaceletBehaviors.show</behavior-id></behavior>
    <attribute>
       <name>target</name>
       <description>Ziel der Javascriptmodifikation</description>
       <type>java.lang.String</type>
       <required>true</required>
    </attribute>
</tag>
```

Taglib-Eintrag für *HtmlOutputTextJS* mit Renderer:

- für jedes Tag-Element sind gewisse Javascript-Events aktiviert
  - ABER: *HtmlOutputText* kann kein *Mouseover/-out*
- *HtmlOutputTextJS* wird eine Erweiterung von *HtmlOutputText*
  - mit der Befähigung zu weiteren Javascript DOM-Events
- Taglib-Eintrag:

```
<tag>
 <tag-name>outputTextJS</tag-name>
  <component>
    <component-type>HtmlOutputTextJS</component-type>
    <renderer-type>FaceletComponents.OutputTextJSRenderer</renderer-type>
  </component>
</tag>
```

Einschub: HTMLOutputText zu *mouseover* etc. befähigen

```
@FacesComponent(value="HtmlOutputTextJS")
public class HtmlOutputTextJS extends HtmlOutputText implements ClientBehaviorHolder{

   private static final Collection<String> EVENT_NAMES =    Collections.unmodifiableCollection
      (Arrays.asList("blur","click","action","dblclick","focus","keydown","keypress",
                     "keyup","mousedown","mousemove","mouseout","mouseover","mouseup"));

   public HtmlOutputTextJS(){ super();
                              setRendererType("FaceletComponents.OutputTextJSRenderer"); }
   @Override
   public Collection<String> getEventNames() { return EVENT_NAMES; }
   @Override
   public String getDefaultEventName() { return "mouseover"; }
   @Override
   public String getFamily(){ return "HtmlOutputTextJS"; }
}
```

Einschub: *HTMLOutputTextJS* → der Renderer

- *HtmlOutputText* benutzt zum Rendern des HTML-Codes
  - eine externe Renderer-Klasse: *TextRenderer*
- damit die zusätzlichen Events auch beim Rendern berücksichtigt werden
  - brauchen wir einen erweiterten Renderer: *OutputTextJSRenderer*
- aufgrund schlechter Programmierung von *TextRenderer*
  - müssen wir eine komplette Methode kopieren: *getEndTextToRender*
- zudem verwenden wir einen kleinen Trick zum „Unterschieben“ zusätzlicher Events
  - wir übergeben als Event-Attribute (*OUTPUT_ATTRIBUTES*)
    - die von OutputBody statt derer von OutputText
- Details irrelevant, bei Interesse siehe Appendix A

Anpassungen in der CC *vieweditdestroypanel.xhtml*

```
...     xmlns:ownfaceletlib="http://www.uni-muenster.de/jsf/util">

  <cc:interface>
     <cc:clientBehavior name="mouseover" event="mouseover" targets="blk1 pV blk2 pE blk3 pD"/>
     <cc:clientBehavior name="mouseout"   event="mouseout"   targets="blk1 pV blk2 pE blk3 pD"/>
  </cc:interface>
  <cc:implementation>
   <ownfaceletlib:outputTextJS  id="blk1" value="&amp;nbsp;&amp;nbsp;&amp;nbsp;"/>
   <span class="initialnotvisible" style="position:absolute; z-index:100000; margin-top:-0.8em" id ="#{cc.clientId}">
    <h:commandButton id="pV" action="#{linkController.prepareView}" value="#{bundle.ListLinkViewLink}"/>
    <ownfaceletlib:outputTextJS  id="blk2" value=" "/>
    <h:commandButton id="pE" action="#{linkController.prepareEdit}" value="#{bundle.ListLinkEditLink}"/>
    <ownfaceletlib:outputTextJS  id="blk3" value=" "/>
    <h:commandButton id="pD" action="#{linkController.destroy}" value="#{bundle.ListLinkDestroyLink}"/>
   </span>
  </cc:implementation>
</html>
```

Finale: *linkliste.xhtml* : 1. Teil (falls editierbar)

```
<span id="#{cc.attrs.id}">
<h:form id="form" rendered ="#{cc.attrs.edit}">
 <ownfaceletlib:ulFrame id="ulFrame" style="list-style: none" listClass="#{cc.attrs.listClass}"
                                                value="#{cc.attrs.liste}"        var="listeneintrag">
  <li>
    <h:outputLink value="#{listeneintrag.linkUrl}">
      <ownfaceletlib:show target="vieweditdestroy" event="mouseover"/>
      <ownfaceletlib:hide   target="vieweditdestroy" event="mouseout"/>
    <h:outputText value="#{listeneintrag.linkName}" /></h:outputLink>
    <ez:vieweditdestroypanel  id="vieweditdestroy">
      <ownfaceletlib:show target="vieweditdestroy" event="mouseover"/>
      <ownfaceletlib:hide   target="vieweditdestroy" event="mouseout" />
    </ez:vieweditdestroypanel>
  </li>
 </ownfaceletlib:ulFrame>
 <h:outputText value="If you want to edit existing links, move the mouse over them." />
</h:form>
```

Finale: *linkliste.xhtml* : 2. Teil (falls nicht editierbar)

```
<ownfaceletlib:ulFrame id="ulFrame" style="list-style: none" listClass="#{cc.attrs.listClass}"
                       value="#{cc.attrs.liste}" var="listeneintrag" rendered="#{!cc.attrs.edit}">
  <li>
    <h:outputLink value="#{listeneintrag.linkUrl}">
      <f:ajax disabled="#{!cc.attrs.ajaxEdit}" event="mouseover"
              listener="#{linkController.prepareEdit}"
               render=":editForm:linkName :editForm:linkUrl"/>
      <f:ajax disabled="#{!cc.attrs.ajaxView}" event="mouseover"
              listener="#{linkController.prepareView}"
               render=":viewForm:linkName :viewForm:linkUrl"/>
      <h:outputText value="#{listeneintrag.linkName}" />
    </h:outputLink>
  </li>
</ownfaceletlib:ulFrame>
</span>
```

Die Vorlesung ist zu Ende...

- Ich hoffe es hat Ihnen Spaß gemacht!

Appendix A: *HTMLOutputTextJS* → der Renderer

```
package FaceletComponents;
// … Imports
@FacesRenderer(rendererType="FaceletComponents.OutputTextJSRenderer",
                        componentFamily="HtmlOutputTextJS")
public class OutputTextJSRenderer extends TextRenderer {
  //      private static final Attribute[] INPUT_ATTRIBUTES =
  //              AttributeManager.getAttributes(AttributeManager.Key.INPUTTEXT);
        private static final Attribute[] OUTPUT_ATTRIBUTES =
              AttributeManager.getAttributes(AttributeManager.Key.OUTPUTBODY);
        public OutputTextJSRenderer(){  super(); System.out.println("Renderer created."); }

        @Override
        protected void getEndTextToRender(FacesContext context,
                                  UIComponent component,
                                  String currentValue)    throws IOException {
         // siehe Folgeseite
       }}
```

Appendix A: *getEndTextToRender* von *OutputTextJSRenderer*

```
ResponseWriter writer = context.getResponseWriter();
assert(writer != null);
boolean shouldWriteIdAttribute = false;
boolean isOutput = false;
String style = (String) component.getAttributes().get("style");
String styleClass = (String) component.getAttributes().get("styleClass");
String dir = (String) component.getAttributes().get("dir");
String lang = (String) component.getAttributes().get("lang");
String title = (String) component.getAttributes().get("title");
if (isOutput = (component instanceof UIOutput)) { // ist es immer
if (styleClass != null || style != null || dir != null || lang != null|| title != null
      || (shouldWriteIdAttribute = shouldWriteIdAttribute(component))) {
    writer.startElement("span", component);
    writeIdAttributeIfNecessary(context, writer, component);
    if (null != styleClass) {writer.writeAttribute("class", styleClass, "styleClass"); }
    // style is rendered as a passthru attribute
    RenderKitUtils.renderPassThruAttributes(context, writer, component, OUTPUT_ATTRIBUTES);
}  //    … 2.Teil siehe Folgeseite
```

Appendix A: *getEndTextToRender* von *OutputTextJSRenderer*

```
// ....    2. Teil:
            if (currentValue != null) {
                Object val = component.getAttributes().get("escape");
                if ((val != null) && Boolean.valueOf(val.toString())) {
                    writer.writeText(currentValue, component, "value");
                } else {
                    writer.write(currentValue);
                }
            }
        } // end if (isOutput = (component instanceof UIOutput))
        if (isOutput && (styleClass != null
                || style != null
                || dir != null
                || lang != null
                || title != null
                || (shouldWriteIdAttribute))) {
            writer.endElement("span");
        } }
```