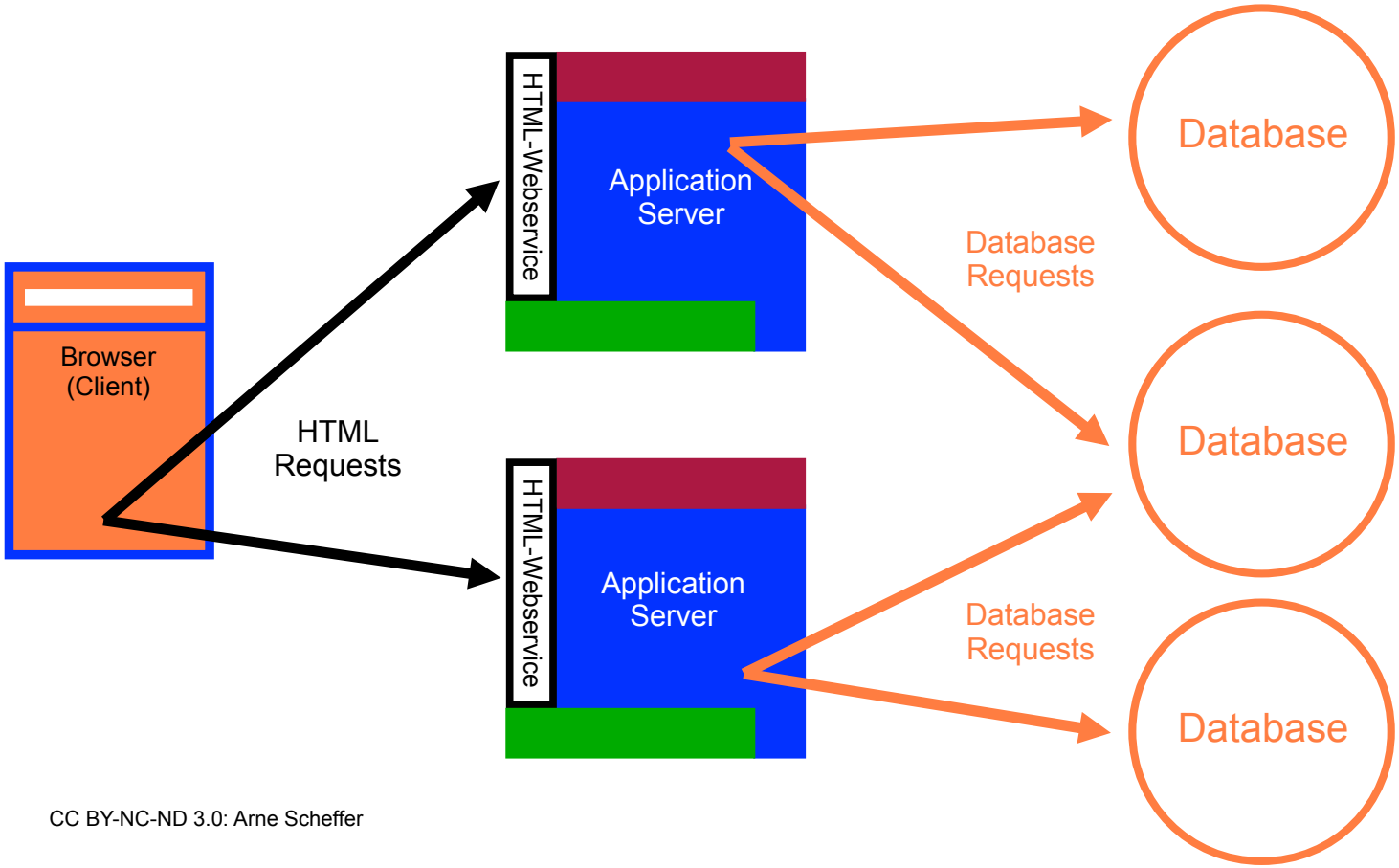

# Enterprise Web Development (Java Enterprise Edition 6)

wissen.leben
WWU Münster



Dozent: Arne Scheffer

# Wiederholung: Application Server / Webserver



# Wiederholung: Model-View-Controller-Konzept

- strikte Gliederung der Code-Bestandteile einer (Web-)Anwendung in
  - ***Sicht*** (View)
  - ***Businesslogik*** (Controller)
  - ***Datenmodell***

  - Vorgehen in dieser Vorlesung: Top Down

  - per Automatismus wird eine MVC-Anwendung erzeugt
    - Basis ist eine einfache Aufgabenstellung
    - Ergebnis ist eine einfache CRUD-Webanwendung

  - dann wenden wir uns den drei Bereichen im Detail zu
    - begonnen mit der Sicht

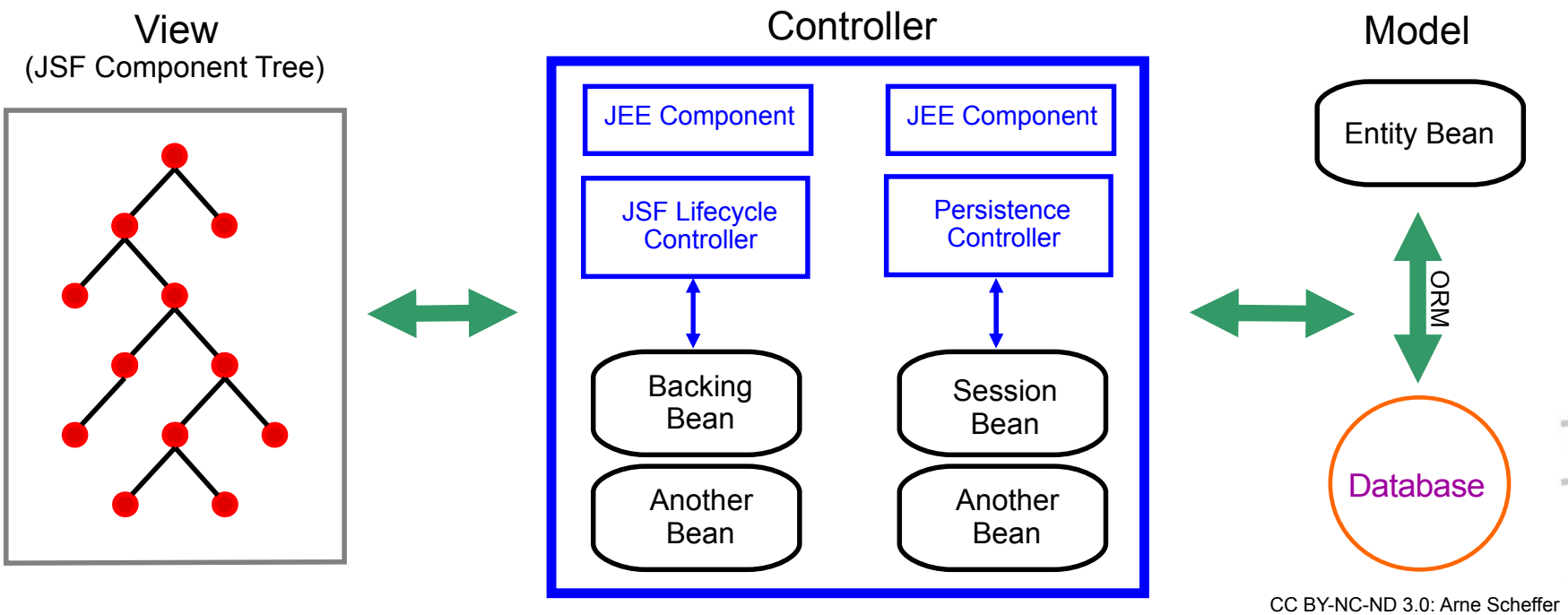
# Wiederholung: Java Server Faces (vereinfacht)

- verwendetes Web-Framework für die Steuerung der Sichten

- generiert HTML aus einem Komponentenbaum
  - Komponentenbaum enthält
    - HTML
      - Webseiten
      - Webseiten-Teile
    - weitere Tags (Auswahlboxen, Eingabefelder, Buttons etc.)
      - mit Variablen parametrisierbar

- hat einen fest definierten Ablaufzyklus zur Parameterverarbeitung

# MVC-Konzept im JEE-Framework






- die Business-Logik („Controller“) steuert frameworkseitig u.a.
  - den JSF-Lifecycle
  - das Persistieren der Daten

# Eingangsbeispiel - Ein Zettel flattert herein...

- die Aufgabe vom Chef:
  - Für ein Webangebot soll
    - auf einigen rechten Spalten eine allgemeine Linkliste angezeigt werden
      - die von der Webredaktion zu editieren ist
        - die dafür eine Hilfskraft hat
  - Die Hilfskräfte haben dort keine HTML-Kenntnisse.
  - Ein Mockup-Muster zum Gucken (und Nörgeln)
    - bis zum Ende der Woche
      - wäre … schön

# Demonstration der Erstellung einer Webanwendung

- Linkliste mittels „Rapid CRUD“
  1. Basis: SQL-Tabellenstruktur

```
DROP TABLE link;
CREATE TABLE link(
 link_id INT NOT NULL,
 link_name VARCHAR(30) NOT NULL,
 link_url VARCHAR(150) NOT NULL,
 primary key (link_id)       );
INSERT INTO link VALUES (1,'Permail','https://permail.uni-muenster.de');
```

  2. Einlesen in die Datenbank
  3. Automatische Generierung einer Entity mittels Netbeans
  4. Automatische Generierung von JSF-Seiten mittels Netbeans
  5. Customizing zur Linkliste

# Vorgehen in Netbeans

- neues Projekt generieren : *Java Web - Web Application*
  - Personal Glassfish V3.x, JEE Web Profile, enable CDI
  - Frameworks: JSF
- SQL-Datei unter im Projekt unter Files ablegen und ausführen
  - verbinden mit: jdbc/sample
- neues File generieren: *Persistence - Entity Classes from Database*
  - Datasource: jdbc/sample
  - Tabelle auswählen, hinzufügen
  - Package benennen (z.B. „entities“), Persistence-Unit erzeugen:
    - Eclipse-Link(JPA2.0), jdbc/sample, use JTA, Strategy: None
- neues File generieren : *Web - JSF-Pages from Entity-Classes*
  - Entity hinzufügen
  - Packages benennen (z.B. „jpasessionbeans“, „jsfclasses“)
- Projekt starten (Clean&Build, Run)

# Customizing

- Inhalte und Daten
  - zunächst automatisiert - Fokus auf JSF-Framework
    - SQL-File so angelegt/ändern, dass das Wesentliche passt

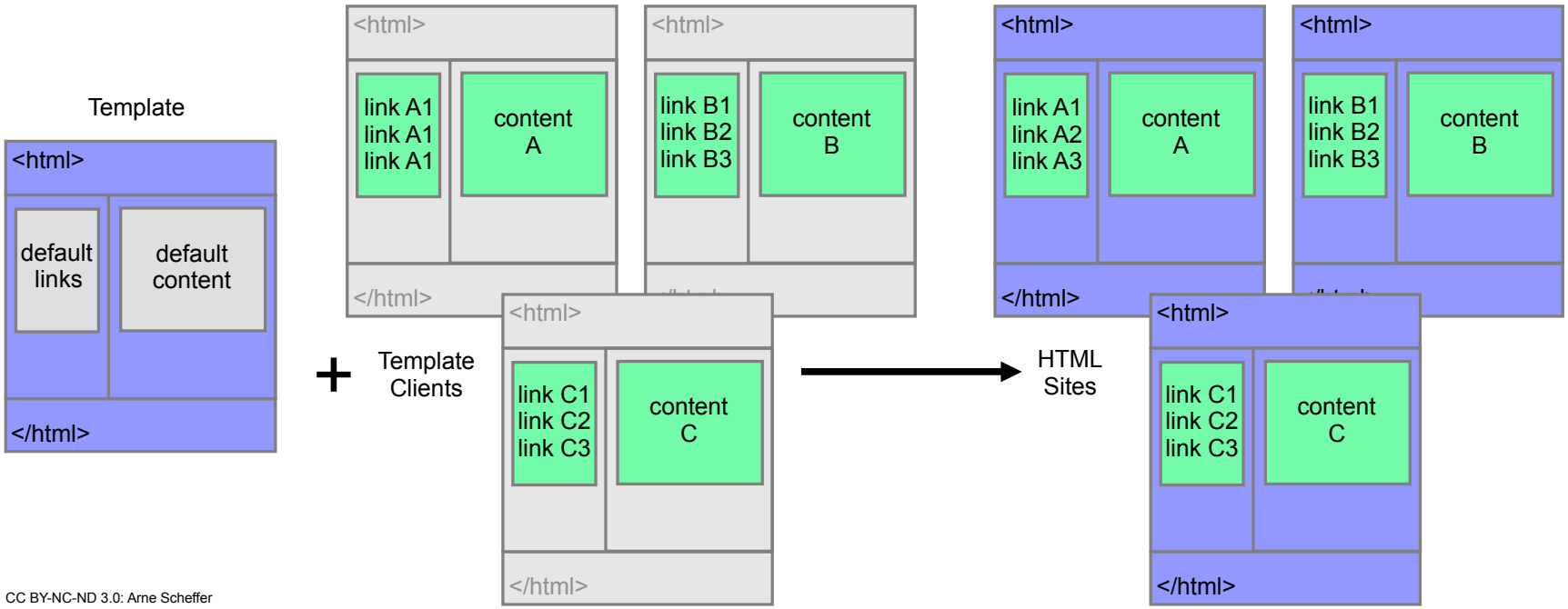
- Sicht
  - Layout einbinden
  - Strukturen an eigene Bedürfnisse anpassen
  - erzeugten Code parametrisierbar und wiederverwendbar machen
    - Copy-and-Paste vermeiden
    - DRY-Mittel in JSF
      - Templating
      - Composite Components
      - (selbstgeschriebene Tags)

# Templates in JSF2

- Facelets Template
  - HTML-Seiten-Schablone (genauer Facelets-Seiten-Schablone)
    - die von vielen weiteren Seiten als Vorlage verwendet werden kann

- Facelets Template Client
  - Facelets-Seite
    - die ein Facelets-Template als Schablone verwendet
    - einen oder mehrere Facelets-Code-Schnipsel enthält
      - die an definierten Stellen in der Schablone eingefügt werden

# Templates in JSF2 - anschaulich



# Templates in JSF2: das Template

- neues File generieren: *Java Server Faces – Facelets Template*

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" .....>
<html xmlns="http://..."  xmlns:ui="http://..."  xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
      <link href="./resources/css/default.css" rel="stylesheet" type="text/css" />
      <link href="./resources/css/cssLayout.css" rel="stylesheet" type="text/css" />
      <title>Facelets Template</title>
   </h:head>
   <h:body>
      <div id="top" class="top">
         <ui:insert name="top">Top</ui:insert>
      </div>
      <div id="content" class="center_content">
         <ui:insert name="content">Content</ui:insert>
      </div>
   </h:body></html>
```

# Templates in JSF2: ein Client

- neues File generieren: *Java Server Faces – Facelets Template Client*
  - generiertes Template auswählen (hier: newTemplate.xhtml)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ....>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="http://java.sun.com/jsf/facelets">
    <body>
        <ui:composition template="./newTemplate.xhtml">
            <ui:define name="top">
                top
            </ui:define>
            <ui:define name="content">
                content
            </ui:define>
        </ui:composition>
    </body>
</html>
```

# Das per RCRUD generierte Template

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ...>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ui="..." xmlns:h="...">
   <h:head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
      <title><ui:insert name="title">Default Title</ui:insert></title>
      <h:outputStylesheet name="css/jsfcrud.css"/>
   </h:head>
   <h:body>
      <h1>
         <ui:insert name="title">Default Title</ui:insert>
      </h1>
      <p>
         <ui:insert name="body">Default Body</ui:insert>
      </p>
   </h:body>
</html>
```

# Integration des eigenen Layouts

- Basis: HTML-Vorlage/Beispielseite im gewünschten Layout
  - wird normalerweise schon vom Web-Designer als Muster zugeliefert
  - man kann ein bestehendes WCMS nutzen (DRY)
- Ersetzen der HTML-Tag-Rahmens durch einen XHTML-Rahmen mit JSF-Tags
- Zugreifbarkeit der verwendeten Verweise (Grafiken, CSS) sicherstellen
  - relativ im Dateibaum verfügbar machen/verlinken (Linux)
    oder
  - absolut umschreiben
- Code-Blöcke aus dem Template der Form
  <ui:define name="...“>...</ui:define>
  - an den dafür vorgesehenen Stellen im Template platzieren
    - Template-Code an der Stelle evtl. als Default zwischen den Tags lassen

- auch der Ergebniscode kann sinnvoll vom WCMS provisioniert werden

# Demonstration

- Kopieren des Quelltextes einer Seite mit Layout

- Ausbau des RCRUD-Templates durch
  1. Drunterkopieren des Quellcodes
  2. Integration des Quellcodes in den Templatecode
  3. JSF-Bug 2215; Workaround für die Vorlesung

     In *template.xhtml* direkt unter *<h:head>* einbauen:

     ```
     <!-- JSF-Bug 2215 Workaround for early Session Generation: <h:outputText
     value="#{linkController.pagination.pageSize}" /> -->
     ```

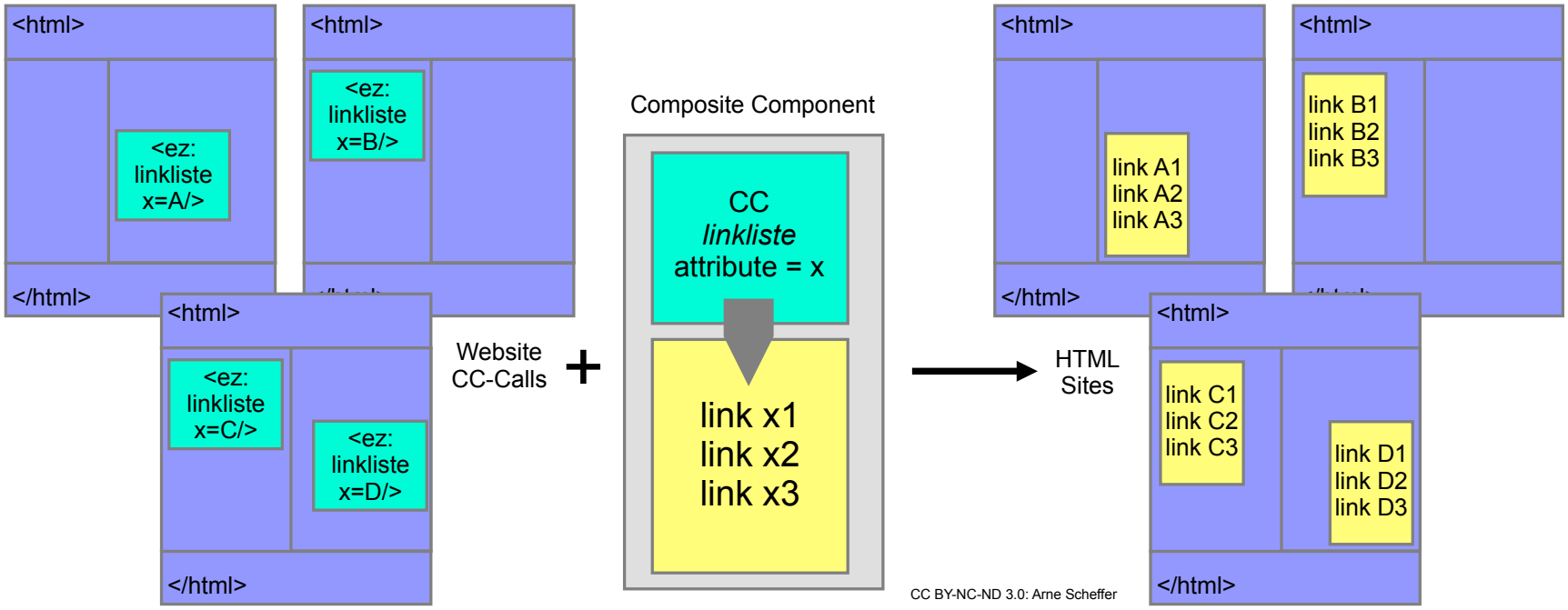
     (Bug ist gefixt in Mojarra 2.1.8)

  4. Zugreifbarkeit der Links sicherstellen (z.B. absolut adressieren)

# Composite Components in JSF2

- funktionieren analog zu Funktionen in einer Programmiersprache
- bündeln in der Sicht
  - statt einer Sequenz von Befehlen
  - eine Sequenz von HTML- und JSF-Tags
- sind parametrisierbar
- lassen so neue benutzerdefinierte Tags entstehen
- erfordern wesentlich weniger Hintergrundwissen
  - als die den JSF-Tags zugrundeliegenden Java-Klassen zu erweitern

# Composite Components in JSF2



# Erzeugen von Composite Components mit Netbeans

- Component HTML-Facelets-Code am späteren Aufrufpunkt erstellen
  - auch bestehende Fragmente können so „verwandelt“ werden
- betroffenen HTML-Code (nach Testlauf) markieren
- rechte Maustaste – Refactor – Convert to Composite Component
- fehlende Libraries in der Composite Component ergänzen
  - und ggf. in der aufrufenden Seite entfernen

# Demonstration

- Code-Fragment für eine ganz einfache Linkliste als Basis:

```
<ul style="list-style-type: none" class="linkliste">
    <ui:repeat value="#{linkController.items}" var="listeneintrag">
     <li><a href="#{listeneintrag.linkUrl}">
            <h:outputText value="#{listeneintrag.linkName}" /></a></li>
    </ui:repeat>
</ul>
```

- Umwandeln in eine Composite Component „linkliste“
- Parametrisieren mit:
  - übergebener Liste
  - CSS-Klasse
- auf der rechten Seite in unser Template einbauen

# Erzeugte Composite Component parametrisieren

- für jeden erwünschten Parameter *bla:*
  - <cc:attribute name="*bla*" /> definieren
  - im Component-HTML-Code den Parameterwert durch #{cc.attrs.*bla*} ersetzen:
    - <tag parameter="#{cc.attrs.*bla*}">
  - im Aufruf-HTML-Code Parameterwert ergänzen, z.B.:
    - <ez:linkliste *bla*="..." />

# Beispiel: Linkliste-Composite-Component

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" ...>
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:cc="http://java.sun.com/jsf/composite"
        xmlns:h="..." xmlns:ui="...">
  <cc:interface>
      <cc:attribute name="liste"/> <cc:attribute name="listClass"/>
  </cc:interface>
  <cc:implementation>
     <ul style="list-style-type: none" class="#{cc.attrs.listClass}">
      <ui:repeat value="#{cc.attrs.liste}" var="listeneintrag">
       <li><a href="#{listeneintrag.linkUrl}"><h:outputText value="#{listeneintrag.linkName}" /></a></li>
      </ui:repeat>
     </ul>
  </cc:implementation>
</html>
```

# Aufruf der Linkliste-Composite-Component vom Client

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ez="http://java.sun.com/jsf/composite/ezcomp">

    <ui:composition template="/template.xhtml">
        <ui:define name="title">...</ui:define>
        <ui:define name="body">

            <ez:linkliste liste="#{linkController.items}" listClass="linkliste" />

        </ui:define>
    </ui:composition>
</html>
```

# Ergebnisse

- Chef wäre zufrieden ?

- unser Beispiel ist der Beweis für die Trennbarkeit der Sicht
  - wir waren ob der Businesslogik VÖLLIG AHNUNGSLOS
  - wir haben sie delegiert (in diesem Fall an den Computer)
- Templating Beweis für die Delegierbarkeit der Optik an Dritte
- Nutzung der Composite Components zeigt DRY-Benefits in der Sicht
  - Wiederverwendbarkeit, Wartbarkeit
- genug für die zweite Woche
  - Dankeschön für die Aufmerksamkeit!