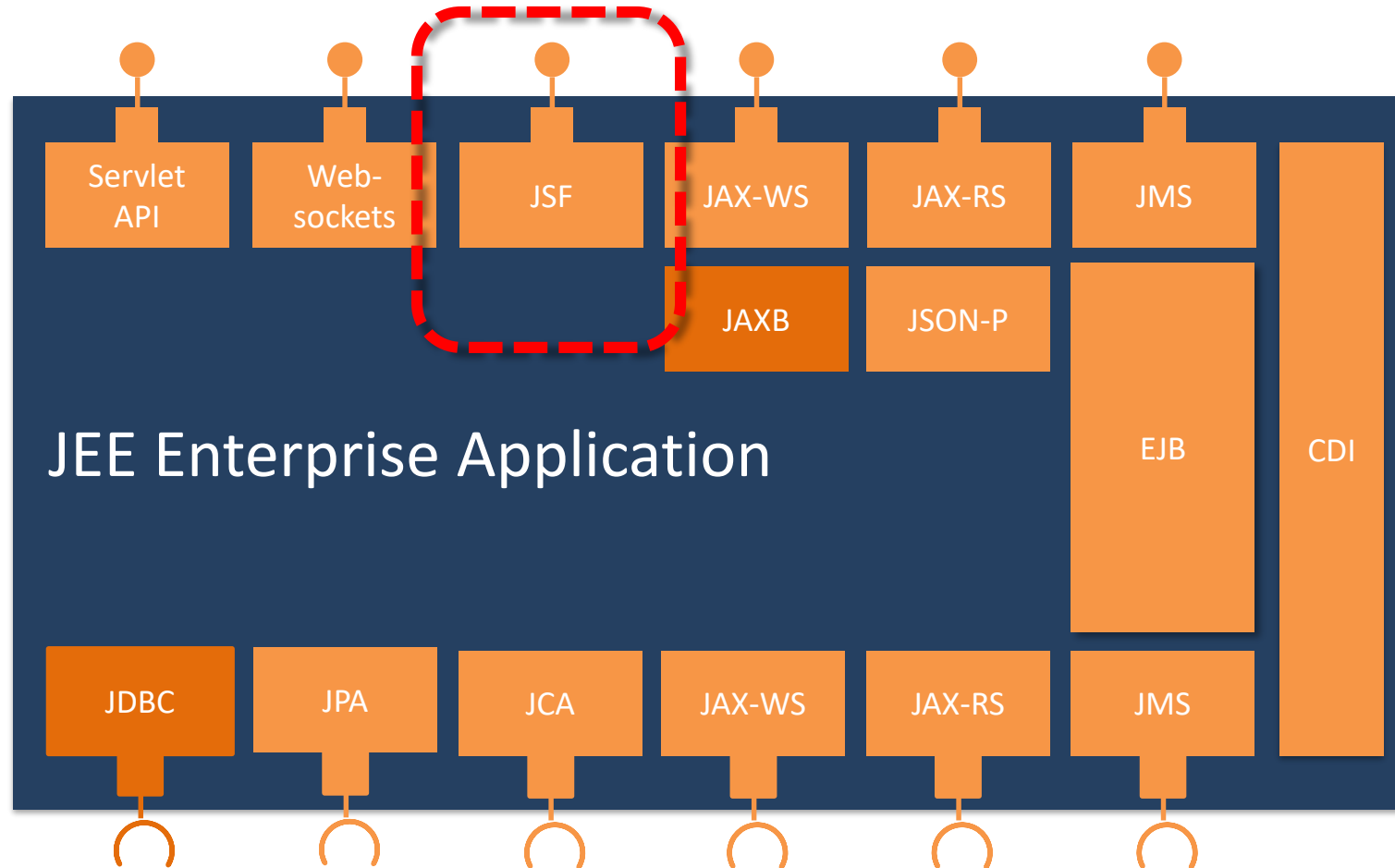


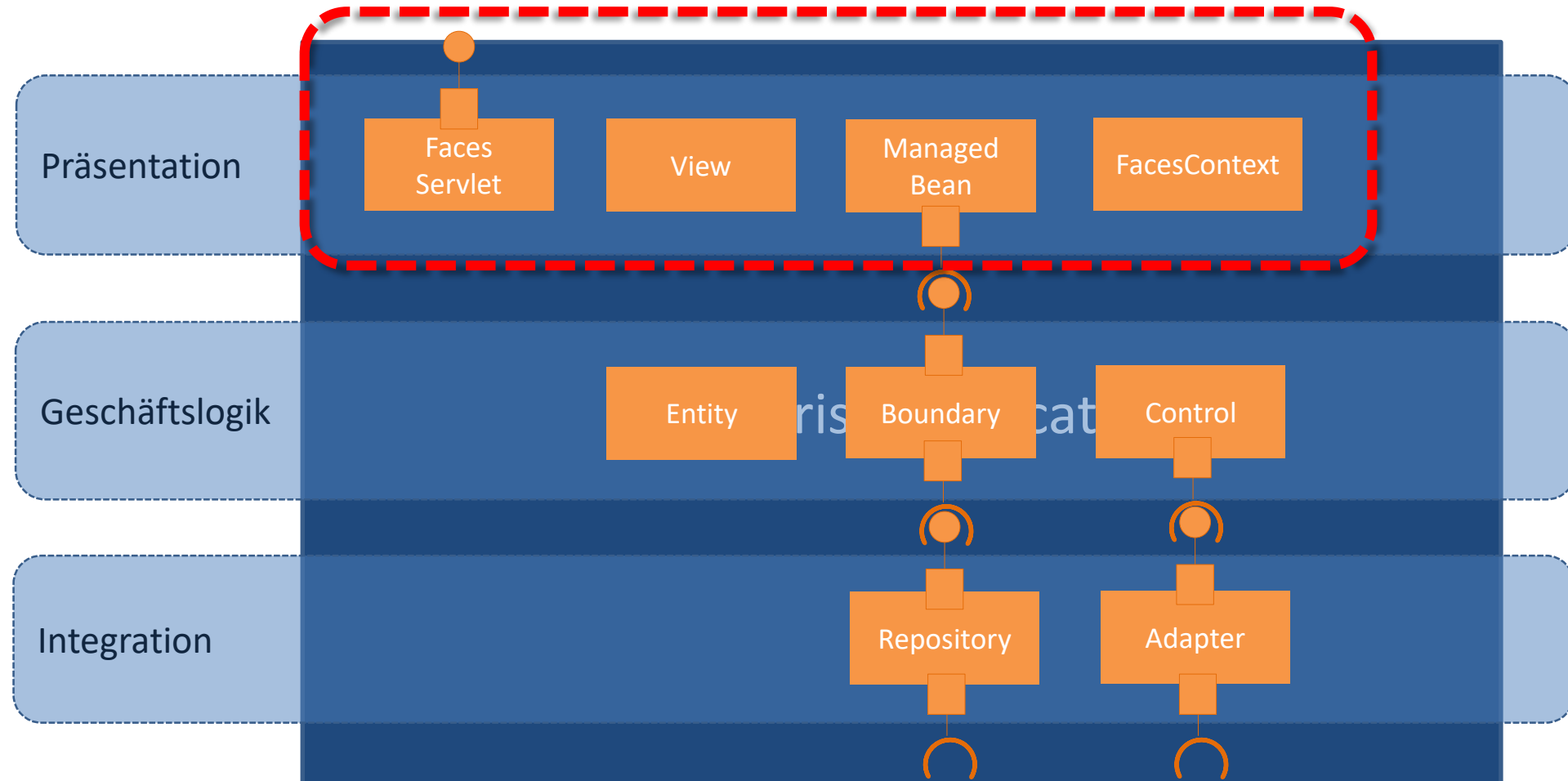
# Webbasierte Benutzerschnittstellen mit JSF

FWP Aktuelle Technologien zur Entwicklung  
verteilter Java-Anwendungen

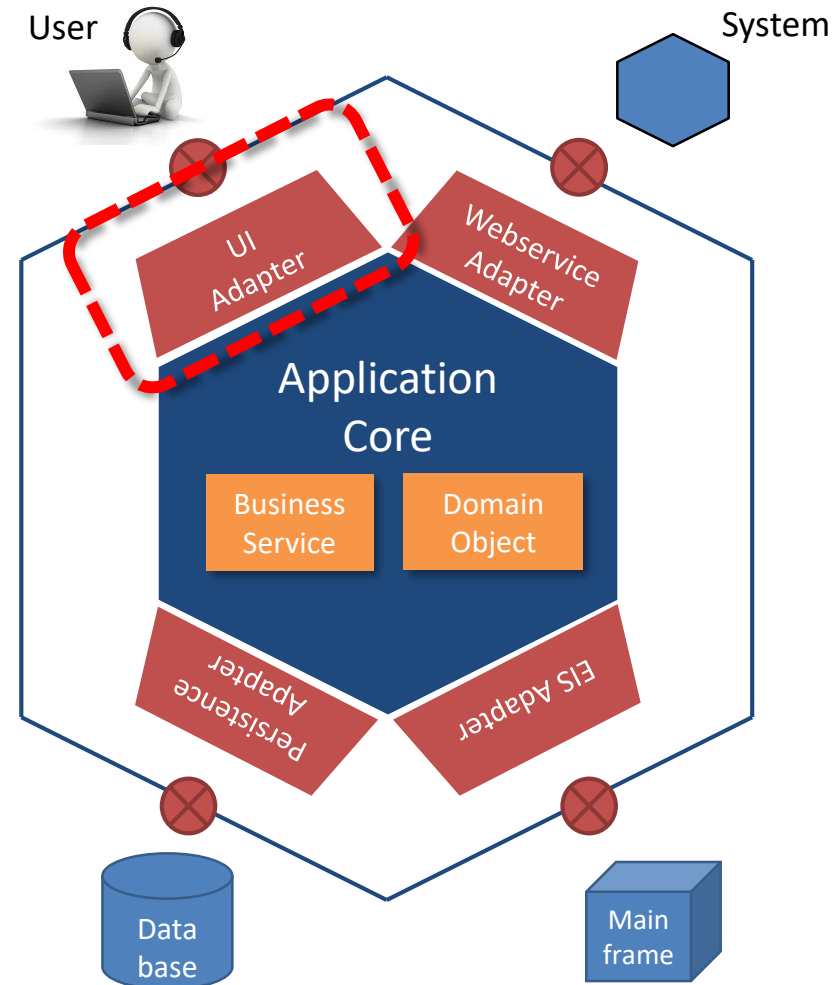
# Kontext: JEE Technologien



# Kontext: Einheitliches Schichtenmodell



# Kontext: Hexagonale Architektur



# **WEB-BASIERTE BENUTZERSCHNITTSTELLEN MIT JAVA SERVER FACES**

# Was ist Java Server Faces (JSF)?

- Standard Java-Framework zum Bauen von Benutzeroberflächen für Webapplikationen
- Design unterstützt einfache Entwicklung:
  - ⊙ Komponentenorientierter Entwicklungsansatz unabhängig von konkreten Endgeräten
  - ⊙ Vereinfachtes Management von Applikationsdaten
  - ⊙ Automatische Verwaltung von Zuständen
  - ⊙ Bietet das Beste aus der Welt der Webapplikations-entwicklung über eine einheitliche, verständliche API

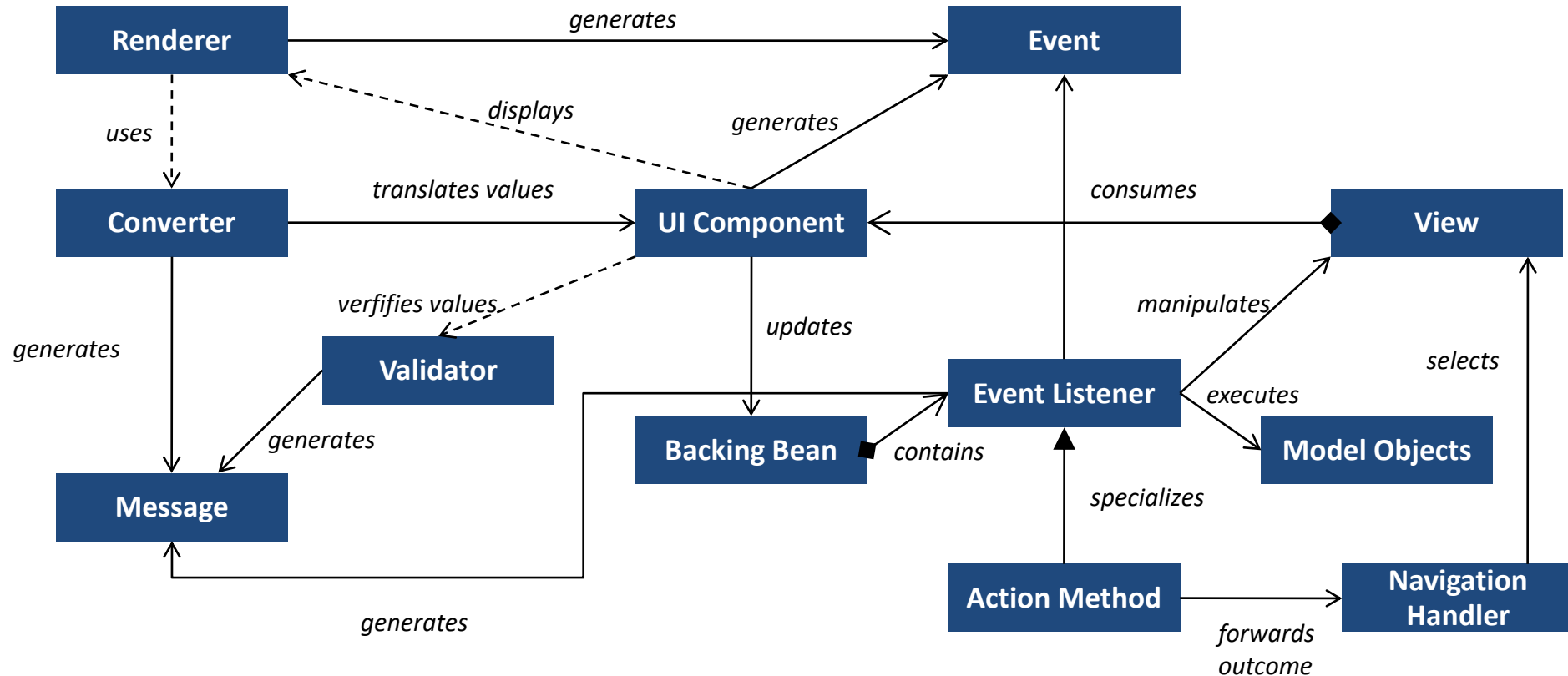
# Design-Ziele von JSF (I)

- Standard-UI-Komponenten-Framework, von Entwicklungstools wirksam einsetzbar
- Einfache, leicht-gewichtige Basisklassen für UI-Komponenten, Komponentenzustände und Eingabeereignisse
- Gemeinsamen UI-Komponenten, die als Basis für neue Komponenten verwendet werden können

# Design-Ziele von JSF (II)

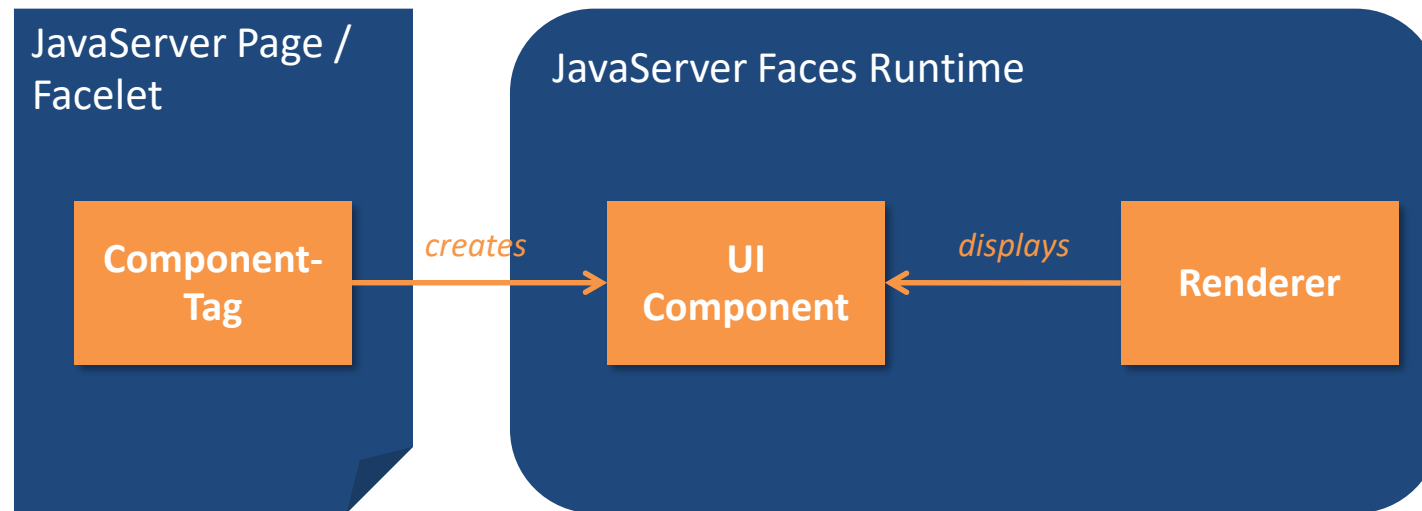
- APIs für die Eingabevalidierung inklusive der Unterstützung für clientseitige Validierung
- Model für die Internationalisierung und Lokalisierung der Benutzerschnittstelle
- Automatische Generierung der zum Endgerät passenden Ausgabe
- Automatische Generierung der Ausgabe bietet Erweiterungspunkte für barrierefreien Zugriff

# JSF basiert auf Komponenten



Quelle: JavaServer Faces in Action, Kito Mann

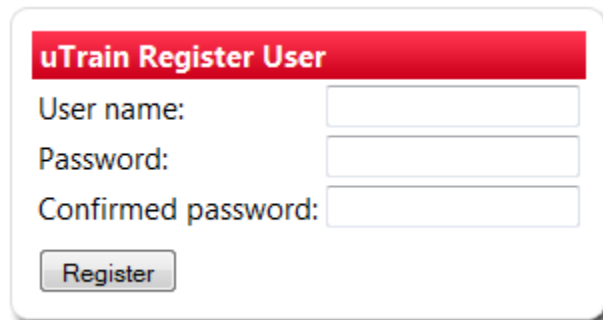
# Dreifaltigkeit als Grundprinzip



# Komponentenbaum erstellt Webseiten

```
<h:form id="registerUserForm">
  <h:inputText
    id="userName" ... />
  <h:inputSecret
    id="password" ... />
  <h:inputSecret
    id="confirmedPassword" .../>
  <h:commandButton
    id="registerButton" ... />
</h:form>
```

*Facelets or JSP view mit  
Komponententags*

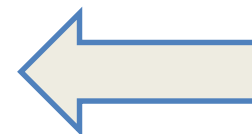


*Im Browser angezeigter View*

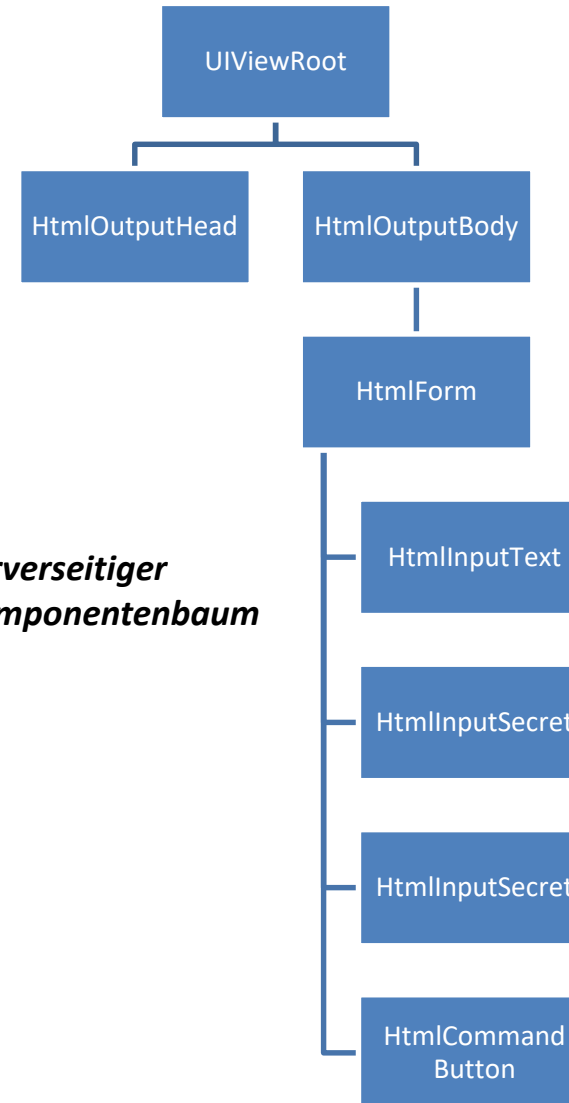


(1)  
*Erstellen des  
Komponentenbaumes*

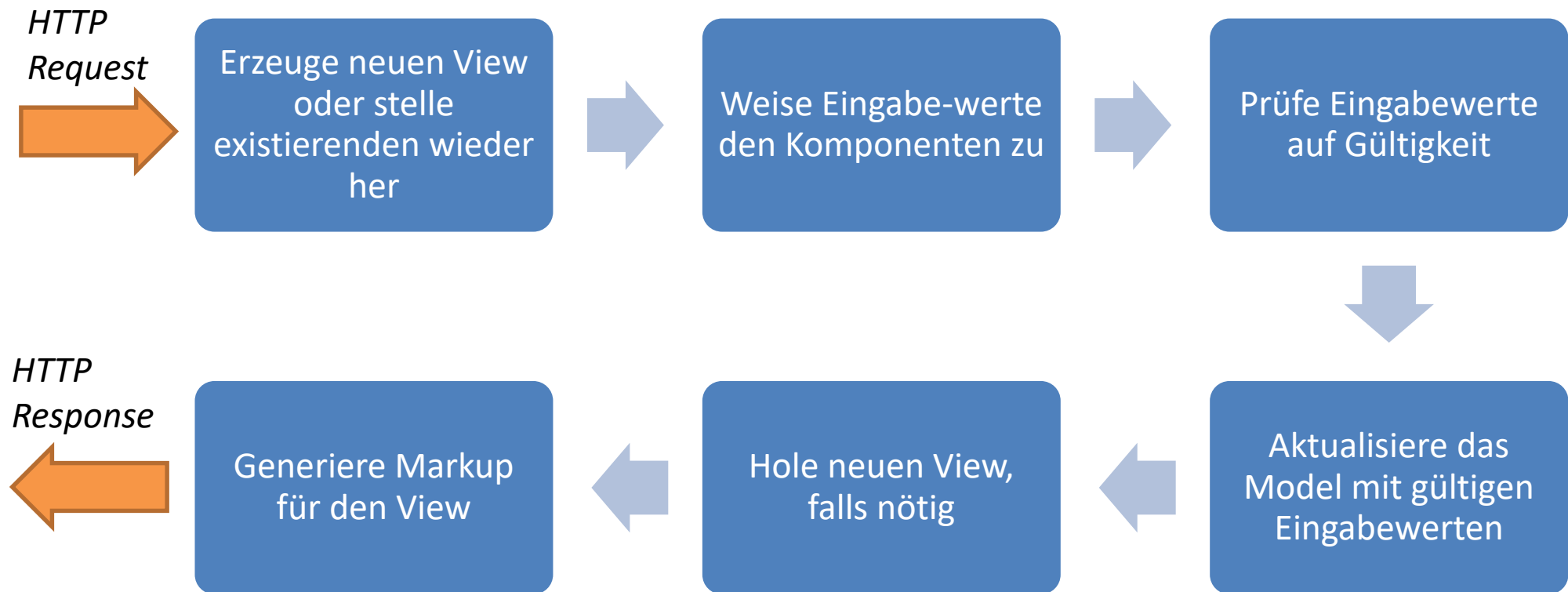
*Serverseitiger  
Komponentenbaum*



(2)  
*Generieren  
des HTML  
Codes*



# Phasen der Requestverarbeitung



# Faces Servlet

- Verarbeitet alle an JSF gerichteten HTTP-Requests
- Folgende URL-Muster identifizieren an JSF gerichtete HTTP-Requests:
  - ◉ `/faces/*` oder `*.jsf`
- Initialisiert beim Start der Web-Applikation die JSF-Laufzeitumgebung
  - ◉ Unter Berücksichtigung der JSF-spezifischen Kontext-Parameter
  - ◉ Unter Berücksichtigung der JSF-Konfigurationsdatei `/WEB-INF/faces-config.xml`

# Faces Context

- Abstrakte Klasse, über welche mit der JSF-Laufzeitumgebung kommuniziert werden kann:
  - ⊙ Zugriff auf Request/Response über `ExternalContext`
  - ⊙ Zugriff auf all konfigurierten JSF-Artefakte
  - ⊙ Zugriff auf den Komponentenbaum (`UIViewRoot`)
  - ⊙ Direktes Schreiben der HTTP Response-Nachricht
  - ⊙ Error-Handling (`FacesMessage`)
- Instanzen werden über die statische Factory-methode `FacesContext.getCurrentInstance()` ermittelt

# Managed Beans

- Managed Bean = POJO + @Named
- Kapseln Präsentationslogik:
  - ⊙ Validiere alle Benutzereingaben
  - ⊙ Bilde Benutzeraktionen auf Aufrufe von Geschäftskomponenten ab
  - ⊙ Ermittle und stelle Geschäftsdaten zur Visualisierung bereit
  - ⊙ Halte Daten über mehrere Requests hinweg
  - ⊙ Entscheide welcher View als nächster anzuzeigen ist
  - ⊙ Behandle alle Fehler, die dazwischen aufgetreten sind

# Lebensdauer einer Managed Bean

- Managed Bean Instanzen werden beim ersten Zugriff erzeugt und dann wiederverwendet
- Gültigkeitsbereich bestimmt deren Lebensdauer:
  - ◉ **RequestScoped**: für die Dauer des aktuellen Requests
  - ◉ **SessionScoped**: für die Dauer der aktuellen Session
  - ◉ **ApplicationScoped**: solange die Applikation läuft
  - ◉ **ViewScoped**: solange der aktuelle View angezeigt wird
  - ◉ **ConversationScoped**: für die Dauer der aktuellen Konversation
  - ◉ **FlowScoped**: solange der aktuelle Flow angezeigt wird

# Managed Beans und Views

- Kommunikation zwischen Managed Beans und Views erfolgt über Ausdrücke in den Views
- Als Sprache für die Ausdrücke dient die **Unified Expression Language**
- Über **Value Expressions** lassen sich Daten aus Managed Beans lesen und schreiben
- Über **Method Expressions** lassen sich Methoden aus den Managed Beans aufrufen

# Unified Expression Language

- Erlaubt Autoren, von Web-Seiten über einfache Ausdrücke auf Daten in Java-Objekten zuzugreifen
- Allgemeine Syntax: `#{<Ausdruck>}`
- Einfache Ausdrücke können Felder oder Methoden in Java-Objekten/Java-Objekt-Bäumen referenzieren:  
`#{userEditor.userName}`
- Komplexe Ausdrücke reichen von Abfragen von logischen Bedingungen bis zu Aufrufen von Methoden mit beliebiger Signatur

# Value Expressions

- Zustandsbehaftete JSF Komponenten besitzen ein Attribut **value**, mit dem Daten in Managed Beans referenziert werden können
  - ⊙ Referenzierung erfolgt über eine Value Expression
  - ⊙ Werte werden beim Rendern eines Views aus dem referenzierten Feld gelesen
  - ⊙ Werte werden beim Submit eines Formulars in referenzierte Feld geschrieben

```
<h:inputText id="userName,"  
  value="#{userEditor.user.userName}" ... />
```

# Method Expressions (Actions)

- Kommandobasierte JSF Komponenten besitzen ein Attribut **action**, mit dem Methoden in Managed Beans referenziert werden können
  - ⦿ Referenzierung erfolgt über eine Method Expression
  - ⦿ Action-Methoden sind parameterlos und liefern einen String-Wert zurück, der den nächsten View bestimmt
  - ⦿ Aufruf der Methode erfolgt beim Submit eines Formulars

```
<h:commandButton id="registerButton"  
  action="#{userEditor.registerUser}" .../>
```

# Method Expressions (Events)

- Ereignisbasierte JSF Komponenten besitzen Attribute, mit dem Listener-Methoden in Managed Beans referenziert werden können
  - ⊙ Referenzierung erfolgt über eine Method Expression
  - ⊙ Art des Ereignisses bestimmt die Signatur der Methode
  - ⊙ Aufruf der Methode erfolgt beim (ggf. partiellen, über AJAX ausgelösten) Submit eines Formulars

# **GESTALTEN VON VIEWS MIT FACELETS**

# Facelets als neue Standard-Viewtechnologie

*„Facelets is a powerful but lightweight page declaration language that is used to build JavaServer Faces view using XHTML style templates to build component trees“*

The JavaEE 6 Tutorial, 2. Ausgabe

- Server-seitiges Templating welches die Komposition von Views aus wiederverwendbaren Teilen ermöglicht
- Entwickelt unter der vollständigen Berücksichtigung von JSF
- Views werden in Standard-XHTML geschrieben
- Bieten die Möglichkeit zur einfachen Erstellung eigener Tag-Libraries
- Bestehende JSF and JSTL Tag-Libraries können weiterhin genutzt werden
- Unterstützen die Unified Expression Language
- Erzwingen die strikte Trennung nach MVC-Paradigma, da Java-Code nicht mehr in Views verwendet werden kann

# Templating

- Ermöglicht die Modularisierung von Facelets in wiederverwendbare Teile basierend auf objekt-orientierten Konzepten:
  - ⊙ **Komposition**: Existierende Facelets können zu neuen Facelets zusammengesetzt werden
  - ⊙ **Vererbung**: Template-Clients können existierende Templates erweitern
  - ⊙ **Überschreiben**: Template-Clients können Teile von Templates überschreiben

# Templates

- Facelet-Templates sind einfache Facelet-Dokumente
  - ⊙ Definieren überschreibbare Templateteile (Platzhalter) mit `ui:insert`
  - ⊙ Inkludieren andere Facelet-Dokumente als Templateteile mit `ui:include`
- Dienen als Vorlagen für die Template-Clients
  - ⊙ Änderungen am Template wirken sich auf alle Clients aus

# Template Clients

- Template Clients sind einfache Facet-Dokumente
  - ⊙ Deklarieren über `ui:composition`, dass sie auf einem Template beruhen
  - ⊙ Geben über das Attribut `template` an, auf welches Template sie sich beziehen
  - ⊙ Überschreiben mit `ui:define` gezielt gleichnamige Platzhalter aus dem Template

# KOMPONENTENMODELL VON JSF

# JSF UI Komponenten

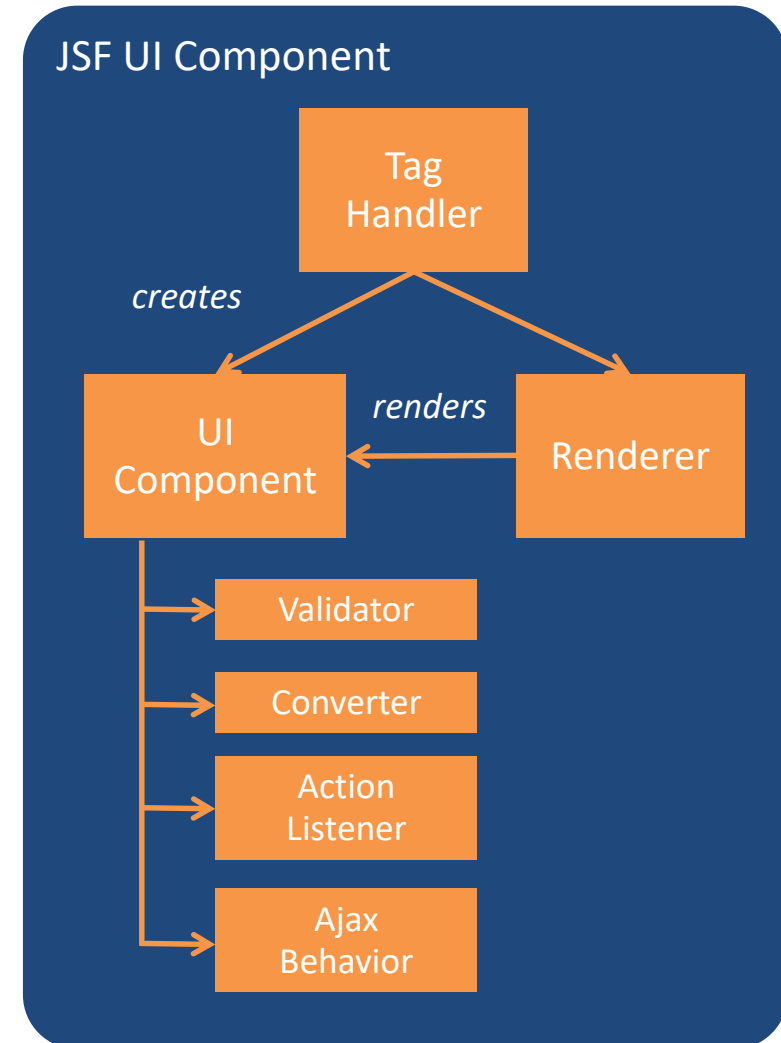
- JSF kommt mit einem erweiterbaren Komponentenmodell für Benutzerschnittstellen
- Basierend auf diesem Komponentenmodell bietet JSF mehr als 30 vorgefertigte UI-Komponenten
- Diese Standard-HTML-UI-Komponenten sind gebunden an den Namensraum <http://xmlns.jcp.org/jsf/html> und den Präfix **h**

A **user interface component** (UI component) is a specific type of component that displays user interface content which the user can modify over time. This content ranges from simple input fields or buttons to more complex items, such as trees or datagrids.

(Source: "JavaServer Faces 2.0 The Complete Reference", Ed Burns & Chris Schalk, McGraw-Hill 2010, p135)

# Bauteile einer UI Komponente

- **UIComponent**: Klasse, welche die abstrakte Semantik der Komponente unabhängig von der Darstellung auf einem spezifischen Endnutzer-Gerät repräsentiert.
- **Renderer**: Eine optional Klasse, die den Markup-Code einer Komponente für ein spezifisches Endnutzer-Gerät erzeugt.
- **Tag Handler**: Klasse, die es erlaubt, eine bestimmte Komponente in eine Web-Seite zu bringen.
- **Attached Objects**: Eine optionale Sammlung von Hilfskomponenten wie *Konvertern*, *Validatoren*, *Action-Listenern* etc, welche zusätzliche Funktionalität zur Verfügung stellen.



# Gemeinsame Attribute (Auszug)

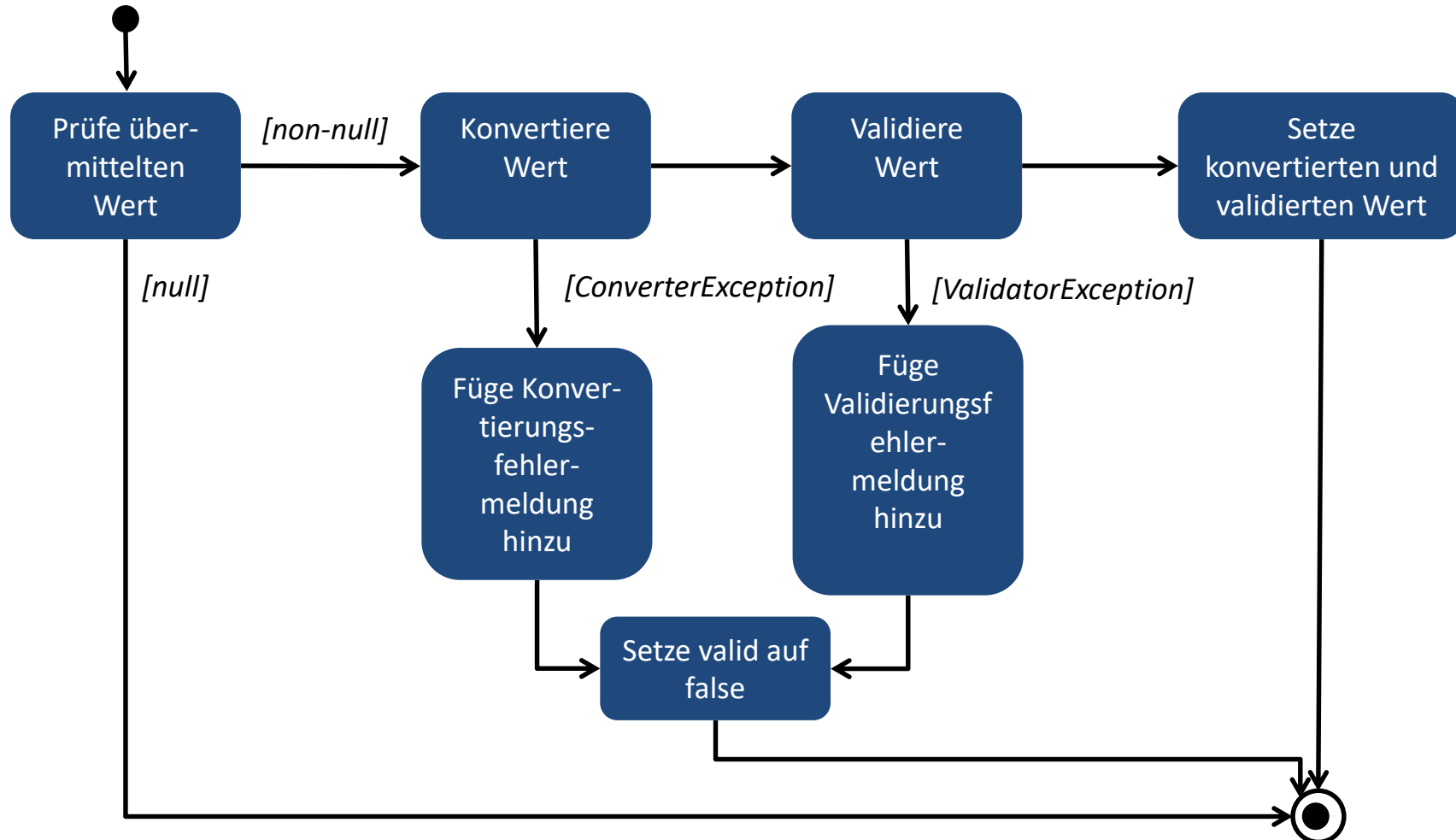
| Attribut         | Beschreibung  |
|------------------|---|
| id               | Eindeutiger Bezeichner der Komponente   |
| value            | Wert der Komponente; entweder Literal oder EL-Ausdruck  |
| immediate        | Steuert, ob ein Komponentenwert konvertiert und validiert werden muss, bevor die Aktion auf der Komponente ausgeführt werden kann |
| rendered         | Steuert, ob für Komponente Markup-Code erzeugt werden soll  |
| required         | Markiert eine Eingabekomponente als Pflichtfeld   |
| requiredMessage  | Optionale spezifische Fehlermeldung bei fehlendem Pflichtfeld   |
| validatorMessage | Optionale spezifische Fehlermeldung bei fehlgeschlagener Validierung  |

# KONVERTER UND VALIDATOREN

# Konverter und Validatoren

- HTML ist eine text-basierte Sprache wogegen Java eine Vielzahl von Datentypen unterstützt
  - ⦿ Konvertierung von String-Werten in den gewünschten Java-Typ und umgekehrt notwendig
- Datenkonvertierung erfolgt in **Konvertern**
- Datenvalidierung erfolgt in **Validatoren**
- Konverter und Validatoren werden an UI-Komponenten gehängt

# Konvertierung und Validierung



# Implizite und explizite Konvertierung

- JSF erkennt automatisch die erforderlichen Konvertierungen und hängt die entsprechenden Konverter an die UI-Komponenten
- Leider funktioniert dieser Automatismus in mindestens zwei Fällen nicht (explizite Angabe von Konvertern erforderlich)
  - ⊙ Datum/Uhrzeit basierend auf `java.util.Date`
  - ⊙ Beträge basierend auf `java.math.BigDecimal` oder `double`

# Validierung mit Bean Validation

- JSR 303 Bean Validation definiert ein einfaches Validierungsmodell
  - ⊙ Kann auf beliebige POJOs angewendet werden
  - ⊙ Einsatz nicht auf die Präsentationsschicht beschränkt
- Validierungsregeln werden durch Annotationen an Feldern definiert (@NotNull => Pflichtfeld)
- Eigene Annotationen und Validierungshandler sind leicht zu implementieren
- JSF unterstützt Bean Validation automatisch

# Bean Validation Annotationen

| Constraint                | Description   |
|---------------------------|---|
| @AssertFalse, @AssertTrue | Spezifiziert gültige Werte für boolean Felder                                     |
| @DecimalMax, @DecimalMin  | Spezifiziert gültigen Wertebereich für Dezimalfelder                              |
| @Digits                   | Spezifiziert gültige Anzahl von Stellen in BigDecimal-Feldern                     |
| @Max, @Min                | Spezifiziert gültigen Wertebereich für int-Felder                                 |
| @NotNull                  | Der Wert des Feldes darf nicht null sein<br>(= Pflichtfeld)                       |
| @Null                     | Der Wert des Feldes muss null sein  |
| @Future, @Past            | Spezifiziert den gültigen Wertebereich für Date und Calendar-Felder               |
| @Pattern                  | Spezifiziert einen regulären Ausdruck, zu dem ein String-Wert passen muss         |
| @Size                     | Spezifiziert eine zulässige Größe für einen String, eine Collection oder eine Map |

# **ERZEUGEN UND ANZEIGEN VON MELDUNGEN**

# Fehlerbehandlung

- Im Kontext von JSF kann die Behandlung von Fehlern in zwei Bereiche unterteilt werden
  - ⊙ Hinzufügen von Fehlermeldungen zum FacesContext (explizit in Managed Beans oder implizit in Konvertern und Validatoren)
  - ⊙ Anzeigen von Fehlermeldungen in Views

# Erzeugen von Meldungen

- Klasse `FacesMessage` repräsentiert Meldungen mit den folgenden Attributen
  - ⊙ severity (FATAL, ERROR, WARNING, INFO)
  - ⊙ summary
  - ⊙ detail
- `FacesMessage`-Objekte können über `FacesContext.addMessage()` zur Anzeige vorgemerkt werden
  - ⊙ Mit optionaler Angabe der betroffenen Komponente

# Anzeige von Meldungen

- Mit `h:message` kann eine einzelne Meldung zu einer bestimmten Komponente direkt neben der Komponente ausgegeben werden
- Mit `h:messages` können alle Fehlermeldungen an einer Stelle im View ausgegeben werden

# **NAVIGATION DURCH EINE JSF-APPLIKATION**

# Navigation

- Navigation funktioniert in JSF im Allgemeinen so:
  - ⊙ Konvertierungsfehler oder Validierungsfehler => Anzeige des letzten Views
  - ⊙ Eine Aktionsmethode liefert kein Ergebnis => Anzeige des letzten Views
  - ⊙ Eine Aktionsmethode liefert ein Ergebnis => der NavigationHandler versucht das Ergebnis auf die URL eines Views abzubilden
    - View lässt sich ermitteln => Anzeige des neuen Views
    - View lässt sich nicht ermitteln => Anzeige des letzten Views

# Navigationsarten

- **Implizite Navigation** (seit JSF 2.0): hard-codierte, logische Seitennamen in Views oder als Ergebnis von Aktionsmethoden
- **Regel-basierte Navigation**: Navigationsregeln in faces-config.xml

# Fragen?



# ANHANG

# Quellen

- Eric Jendrock et. al.  
**The Java EE 7 Tutorial Part III Kapitel 7-16**  
<http://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm>  
Oracle September 2014
- Angel Leonhard  
**Mastering JavaServer Faces 2.2**  
Packt Publishing 2014  
ISBN 978-1-78217-646-6
- Ed Burns, Chris Schalk  
**JavaServer Faces 2.0: The Complete Reference**  
McGraw-Hill 2010  
ISBN 978-0-07-162509-8



# Kontakt



## **Michael Theis**

Lehrbeauftragter Hochschule München

email      michael.theis@hm.edu

mobile     + 49 170 5403805

web        <http://www.tschutschu.de/Lehrauftrag.html>