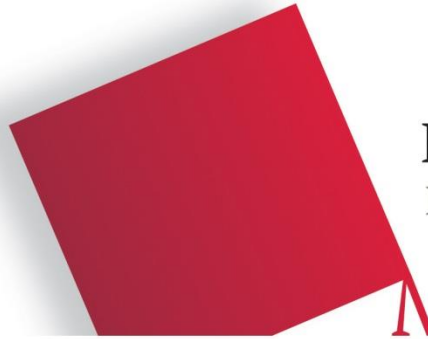


HOCHSCHULE
FÜR ANGEWANDTE
WISSENSCHAFTEN
MÜNCHEN

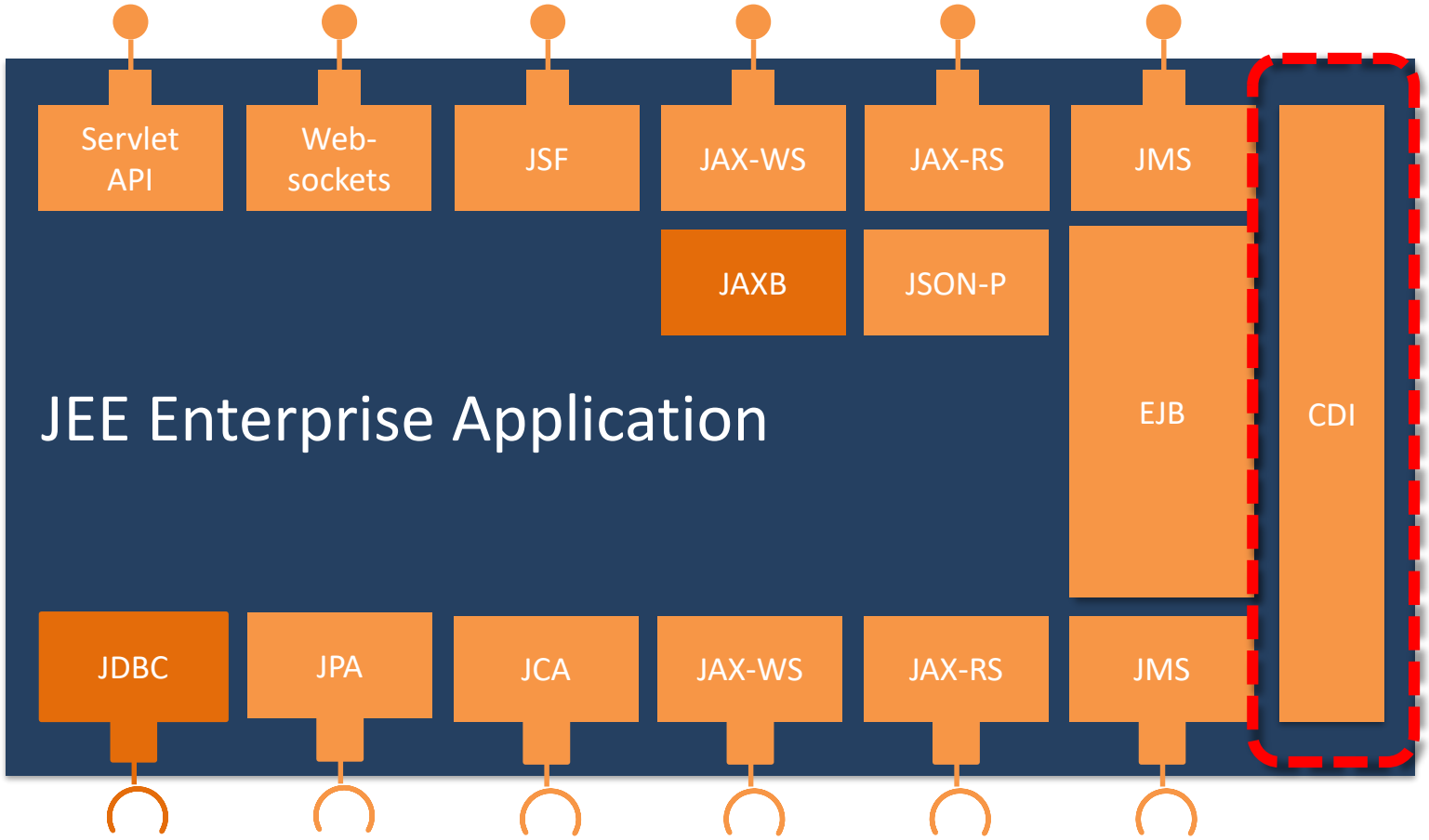
Dependency Injection, AOP und Kontexte mit CDI

FWP Aktuelle Technologien zur Entwicklung verteilter Java-Anwendungen

Michael Theis, Lehrbeauftragter

MOTIVATION

Kontext: JEE Technologien



Grundidee von CDI

- Lose gekoppelte POJOs als Programmiermodell
- Kontextuelle Komponenten mit automatischer Verwaltung ihres Lebenszyklus
- Dependency Injection
- Aspect Oriented Programming
- Ereignisbasierte Kommunikation
- Vollständige Integration in alle bestehenden JavaEE-Technologien

Dependency Injection mit CDI

DAS „DI“ IN CDI

Dependency Injection (rekap.)

- Abhängigkeiten zu benötigten Komponenten werden mit Annotationen deklariert
- Container (CDI/EJB) löst die Abhängigkeiten zwischen Komponenten auf
- Ermöglicht Konzentration auf das Wesentliche
  - Kein Lookup-Code, keine Factories, kein hand-geschriebener Code (➔ DRY)

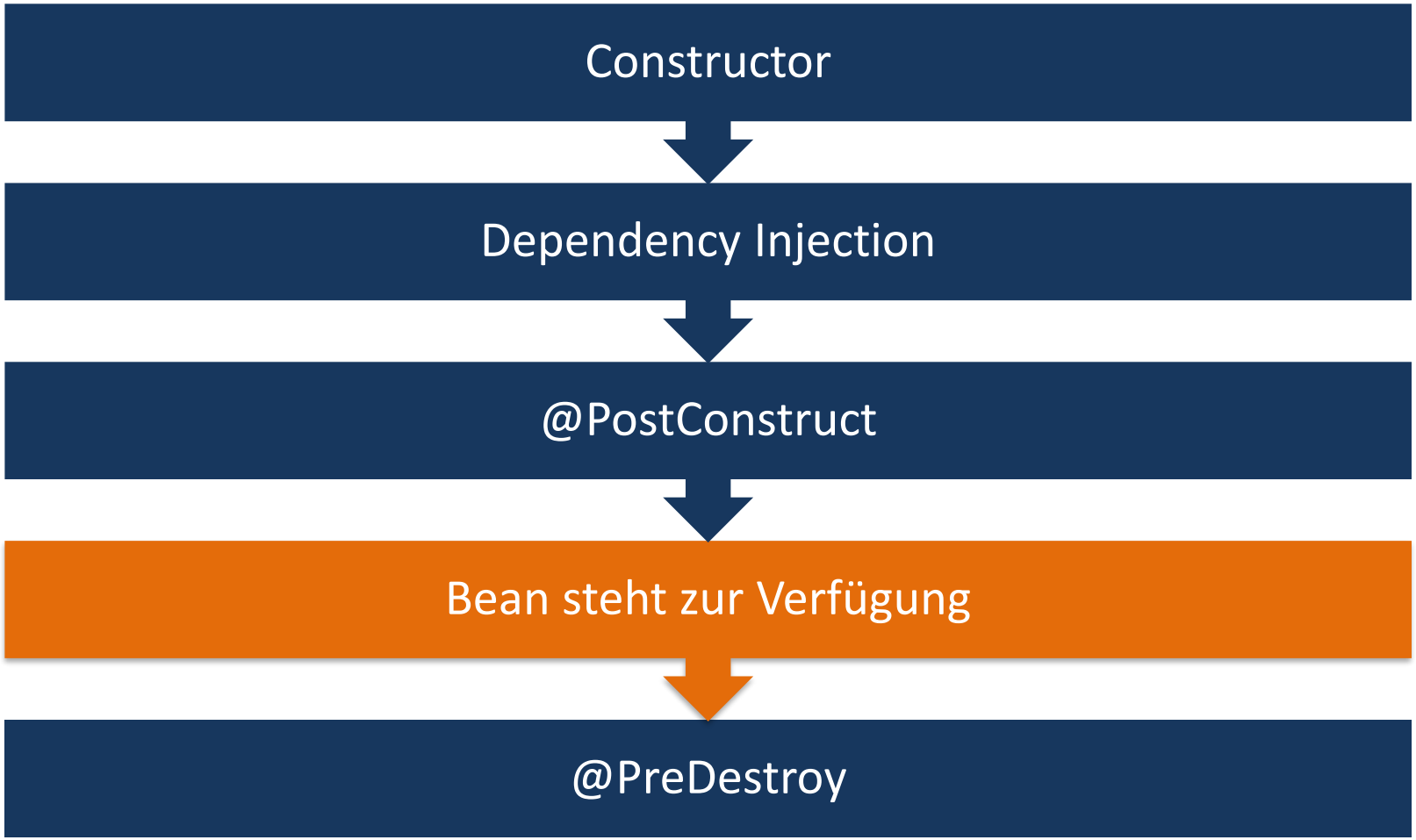
Beans mit CDI

- CDI-Managed Bean = POJO-Klasse + @Named
  - oder POJO + @Stereotype-basierte Annotation
- Bindung an einen Kontext über Scopes
- Optionale Qualifizierung über Qualifiers
- Über @Inject injizierbar in alle Komponenten-typen (u.a. EJB)
- Können über @Inject Abhängigkeiten zu allen Komponententypen besitzen (u.a. EJB)

Anforderungen an Beans

- Konkrete nicht-finale Klasse
  - oder abstrakte Basisklasse mit @Decorated
  - innere statische Klassen möglich
- Passender Konstruktor
  - parameterloser Default-Konstruktor
  - spezialisierter Konstruktor mit @Inject

Lebenzyklus eines Beans



Einschalten von CDI

- JavaEE 7 Application Server erkennt CDI-managed Beans mit Scope-Annotationen automatisch
- Optional kann der Deployment Descriptor beans.xml angegeben werden:
  - In JAR-/EJB-Modulen unter */META-INF/beans.xml*
  - In WAR-Modulen unter */WEB-INF/beans.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                  http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"
       bean-discovery-mode="all">
</beans>
```

Producer: Factories für Beans

- Producer kapseln komplexe Instanziierungslogik für Beans (➔ FACTORY-Pattern)
- Von Producern erzeugte Beans stehen für DI Injection zur Verfügung
- Producer = [Statische] Methode + `@Produces`
  (➔ [Statische] Factory-Methode)
- Factory-Methode kann optional Informationen über das Ziel der DI erhalten `(InjectionPoint)`

Modularisierung von Cross Cutting Concerns mittels Interceptors und Decorators

ASPECT-ORIENTIERTE PROGRAMMIERUNG MIT CDI

Interzeptoren *(Interceptors)*

- Kapselt `Cross Cutting Concern`
- Ein Interceptor passt auf beliebige Zieltypen
- Interceptor = POJO + `@Inceptor` + `@{BindingAnnotation}` [+`@Priority`]
- Binding Annotation = Annotation + `@InterceptorBinding`
- Bindung von Interceptor an Zielobjekt erfolgt über gleiche Binding Annotation
- Aktivierung erfolgt über `beans.xml` oder `@Priority`

Dekoratoren *(Decorators)*

- Erweitert bestehende spezifische Logik
- Decorator ist vom gleichen Typ wie Zieltyp
- Decorator = [Abstract] POJO + `@Decorator` [+ `@Priority`]
- Umhülltes Bean (Delegate) steht über DI zur Verfügung
  - Delegate = Feld + `@Inject` + `@Delegate`
- Aktivierung erfolgt über `beans.xml` oder `@Priority`

Kontexte und Konversationen mit CDI

DAS „C“ IN CDI

Scopes bestimmen Lebendauer von Beans

- Scope definiert den Kontext, an den ein Bean gebunden ist
- Bean wird instanziiert, wenn es zum ersten Mal innerhalb seines Kontextes referenziert wird
- Wird der Kontext geschlossen, werden alle gebundenen Beans freigegeben
- Beans können Scope selber bestimmen oder Scope des Nutzers erben

Vordefinierte Scopes

- CDI stellt folgende Scopes zur Verfügung
  - @RequestScoped: HTTP-Request
  - @SessionScoped: HTTP-Session
  - @ApplicationScoped: Webapplikation
  - @ConversationScoped: Konversation
  - @Dependent: Teilt Scope des Nutzers
- Integration erweiterter Scopes über SPI möglich

Domänen-Ereignisse mit CDI

FEUERN UND FANGEN VON EREIGNISSEN

Domänen-Ereignisse (Domain Events)

- Konsument registriert sich für bestimmte Ereignisse
  (➔ OBSERVER-Pattern)
- Produzent feuert Ereignis, das allen registrierten Konsumenten asynchron zugestellt wird
- Lose Kopplung mit asynchroner Kommunikation zwischen Produzent und Konsument
- Zustellen von Ereignisse kann transaktional erfolgen

Ereignistypen (Events)

- Event = POJO mit Default-Konstruktor
- Keine Annotationen erforderlich
- Eventtypen können mit `Qualifiern` feiner spezifiziert werden

Fangen von Events

- Observer-Methoden hören auf Events
- Observer-Methode =

  Method + (Parameter + `@Observes`)

```
public class MyEventObserverBean {
    ...
    public void onMyEvent(@Observes MyEvent event) {
        // Reaktion auf MyEvent
    }
```

- Zustellung über `@Observes`-Attribute steuerbar
  - `during` bestimmt Transaktionalität der Zustellung
  - `notifyObserver` bestimmt Bedingung der Zustellung

Feuern von Events

- Über `Event<EventType>` können Events gefeuert werden

```
public class MyEventNotifierBean {
    @Inject
    private Event<MyEvent> notifier;
    ...
    public void fireMyEvent(String eventArgument) {
        // Erzeugen des Events
        MyEvent event = new MyEvent();
        event.setArgument(eventArgument);
        // Feuern des Events an alle Observer
        notifier.fire(event);
    }
```

Fragen?

ANHANG

Quellen

- Beispiel-Code auf GitHub unter https://github.com/mikeT92/jeetrain
  MAVEN-Projekt **jeedemo** MAVEN-Modul **jeedemo-cdi**
- ***The Java EE Tutorial Part V Contexts and Dependency Injection for Java EE***
  https://javaee.github.io/tutorial/partcdi.html#GJBNR

Kontakt

Michael Theis
Lehrbeauftragter Hochschule München

email michael.theis@hm.edu
mobile + 49 170 5403805
web http://www.tschutschu.de/Lehrauftrag.html