

3.– 6. September 2012
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Freundliche Eindringlinge

Moderne Web-Applikationen mit CDI

Norman Erck

Holisticon AG

Norman Erck

- **Holisticon AG - Management- und IT-Beratung**
 - Architektur
 - Agil/Projektmanagement
 - BPM/SOA
- **Berater**
 - Architektur
 - eBusiness
 - Web-Technologien
 - Portale



Inhalt

- Der Weg zu CDI
- Motivation
- Theorie
 - Neuerung
 - Bean-Verständnis
 - Konzepte
- Praxis
- Demo „Friendly Invaders“
- Ausblick Weld 2.0
- Zusammenfassung



Der Weg zu CDI - Geschichte

- Fowler et al prägen den Begriff POJO (2000)
- Johnson et al prägt J2EE & IoC (2002)
 - „J2EE Development without EJB“
 - Inversion Of Control (IoC)
- Erste IoC Container erscheinen (Mitte 2002)
- Martin Fowler bezeichnet Dependency Injection als Form von Inversion of Control (2004)
 - Konstruktor Injection
 - Interface Injection
 - Getter Injection

Der Weg zu CDI – Warum?

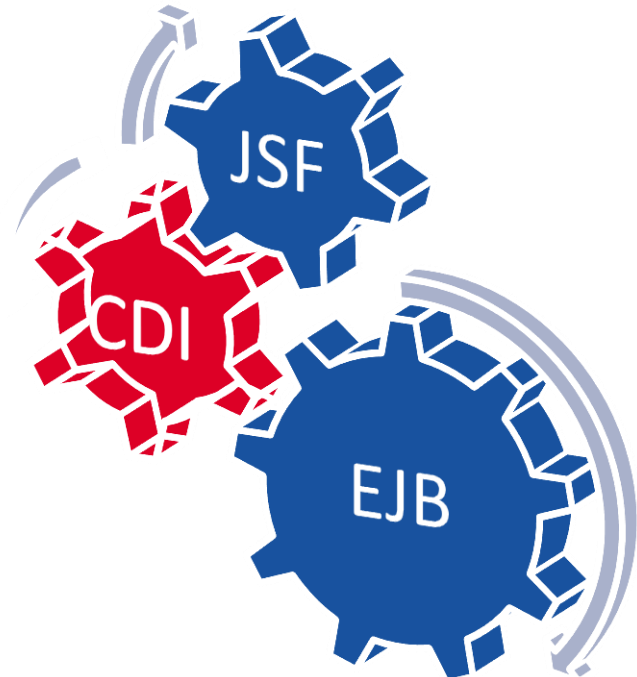
- DI ermöglicht POJOs
- POJO & KISS
 - Einfachheit
 - Separation of Concerns
- Schöner Code
 - Zwang zu schönem Code
 - Wartbarer Code
- Unit Testing erleichtern

Der Weg zu CDI - JSRs

- JSR 250 Common Annotations (2006-05)
 - Einführung allgemeiner Annotationen für Java EE & SE
 - Erlauben deklarative Programmierung
 - Autorisation (@RunAs etc.) & Injection (@Resource, @PostConstruct, @PreDestroy etc.)
 - Java 6 SE
- JSR 330 Dependency Injection (DI) for Java (2009-10)
 - Allgemeines Verständnis der injection Semantik durch portable DI Annotationen
 - Kleinster JSR mit @Inject, @Named, @Qualifier, @Scope, @Singleton
 - Spring 3 & Guice 3
- JSR 299 Contexts and Dependency Injection for the Java EE platform (2009-11)

Motivation – JSR 299

- Noch kein umfassendes DI Framework im Java EE Standard
- Standardisierung und Vereinfachung des Zusammenspiels von JSF und EJB
 - Präsentationsschicht
 - Geschäftslogik



Neuerungen – Was bietet CDI?

- DI-Mechanismus (Dependency Injection)
- Deklarative Verknüpfung von Beans
- Scopes
- Interzeptorenkonzept
- Dekoratoren
- Bean-Verwaltung zur Laufzeit

Bean-Verständnis - Beans

■ JSF, EJB, Spring & CDI

- Eine Bean ist ein Objekt, dessen Lebenszyklus vom Container verwaltet wird.

■ Managed Beans

- Parameterlosen Konstruktor | mit `@Inject` annotiert
- Lebenszyklus & Beziehungen verwaltet

■ Session Beans

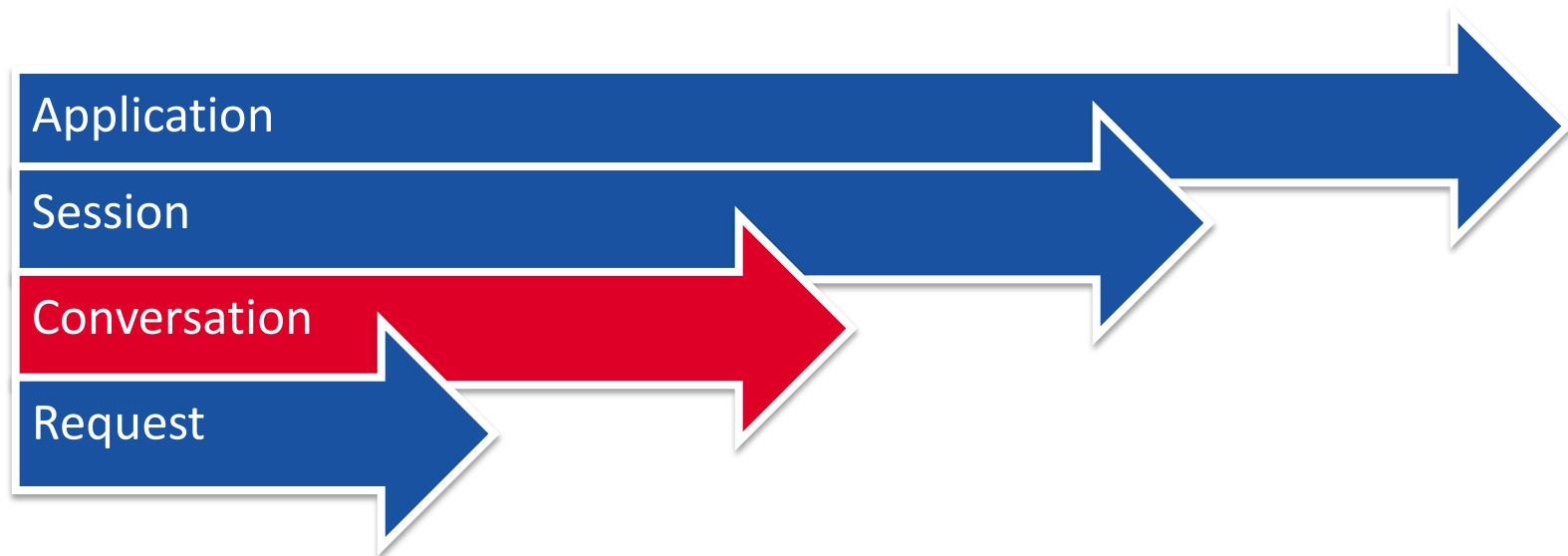
- Managed Beans + Lebenszyklus, Statusmanagement- & Nebenläufigkeitsmodell
- Stateless Session Bean und Singleton Session Bean „scopeless“
- Stateful Session Beans beliebiger Scope

Bean-Verständnis - Nicht Beans

- Message-Driven Beans
 - Interzeptoren
 - Servlets
 - JAX WS Services
 - JSP Tag Handler
 - Tag Library Event Listener
-
- Können Beans injiziert bekommen

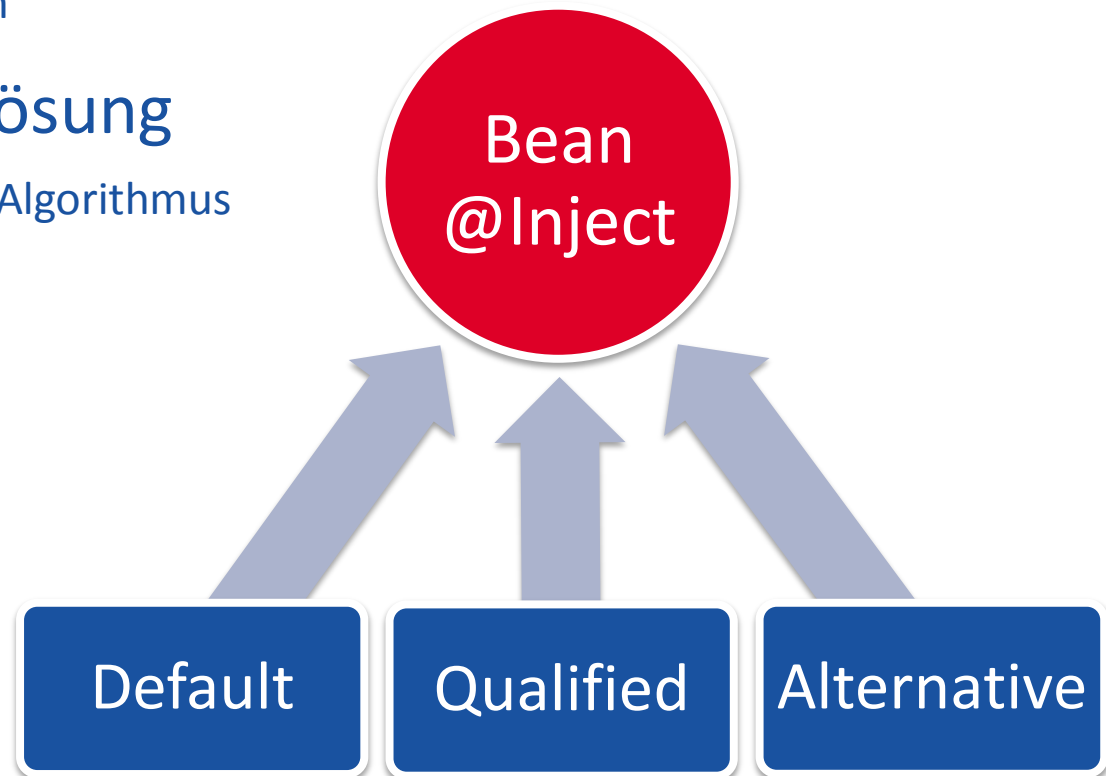
Konzepte

- Container, Kontext & Scope
 - Verwaltung Lebenszyklus
 - Beziehungen zwischen verwalteten Objekten
 - Sichtbarkeit



Konzepte

- Dependency Injection
 - Nicht neu
 - Java-EE-Applikationen
- Typsichere Auflösung
 - Typesafe-Resolution-Algorithmus
 - Qualifier
 - Alternativen



Konzepte

- Lose Kopplung
 - Client & Server
 - Lebenszyklus
 - Produzenten & Konsumenten
- Bean-Namen
 - EL-Namen
 - *Keine* JSF Managed Beans
- Interceptor-Binding & Dekoratoren
- SPI
 - Portable Integration von Erweiterungen

Praxis

- Weld
 - Referenzimplementierung JBoss
- Deployment-Deskriptor
 - WAR: `WEB-INF/beans.xml`
 - JAR: `META-INF/beans.xml`
 - Kann leer sein



Praxis

- „Friendly Invaders“
- Werkzeugkiste
 - DI
 - Qualifier
 - Alternativen
 - EL-Namen
 - Scopes
 - Stereotypen
 - Events
 - Interceptor-Binding
 - Dekoratoren
 - Produzenten

Dependency Injection

```
@Stateless
```

```
public class BlogServiceBean implements BlogService {
```

```
...
```

```
}
```

```
public class InvasionController {
```

```
    @Inject
```

```
    private BlogService blogService;
```

```
...
```

```
}
```


Qualifier

@Qualifier

@Target({*TYPE*, *METHOD*, *PARAMETER*, *FIELD*})

@Retention(*RUNTIME*)

public @interface Holisticon {}

@Holisticon

@Stateless

public class HolisticonBlogServiceBean
 implements BlogService {

 ...
}

Qualifier

```
public class InvasionController {  
    @Inject @Holisticon  
    private BlogService blogService;  
    ...  
}
```

■ Mehrere Implementierungen?

- @Default
- Sonst: Start verweigert

WELD-001409 Ambiguous dependencies for type [BlogService] with qualifiers [@Default] at injection point ...

Alternativen

@Alternative

```
public class BlogServiceMock implements BlogService,  
    Serializable{
```

```
    ...  
}
```

- Aktivierung in beans.xml

```
<alternatives>  
  <class>  
    de.holisticon.blog.fiendlyInvaders.service.  
    mock.BlogServiceMock  
  </class>  
</alternatives>
```

EL-Namen

- Bean CDI Kontext hinzufügen

@Named

```
public class InvasionController{
```

```
    @Inject @Holisticon
```

```
    private BlogService blogService;
```

```
    ...
```

```
}
```

```
<ui:repeat var="blogArticle"  
    value="#{invasionController.blogArticles}">  
    #{blogArticle.title}  
</ui:repeat>
```

Scopes

@Named

@SessionScoped

```
public class InvasionController implements Serializable {
```

```
    private static final long serialVersionUID = 1L;
```

```
    @Inject @Holisticon
```

```
    private BlogService blogService;
```

```
    ...
```

```
}
```

■ Serialisierbar

- SessionScoped & ConversationScoped
- Sonst: Start verweigert

WELD-000072 Managed bean declaring a passivating scope must be passivation capable.

Scopes

@Stateful

@ConversationScoped

public class CheckoutServiceBean **implements** ... {

@Inject

private Conversation **conversation**;

private KudosCart **kudosCart**;

public KudosCart createKudosCart() {

kudosCart = **new** KudosCart();

 // if no conversation is "open"

if (**conversation.isTransient()**)

conversation.begin();

return kudosCart;

}

public void checkout() {

kudosCart.getCart().clear();

conversation.end();

}

}

Stereotypen

- Annotationen gebündelt

@SessionScoped

@Named

@Stereotype

@Retention(*RUNTIME*)

@Target(*TYPE*)

public @interface NSS {}

@NSS

public class InvasionController **implements** Serializable{

...
}

Events

@Qualifier

@Target({*FIELD*, *PARAMETER*})

@Retention(*RUNTIME*)

public @interface Invaded {}

■ Producer klassifizieren

- Filtern ermöglichen

@Inject @Invaded

private Event<BlogArticle> blogArticleEvent;

public String giveKudos(BlogArticle blogArticle, int amount{

blogArticleEvent.fire(blogArticle);

...

}

Events

■ Observer Lauschen

- InvasionTweeter
- InvasionMailer
- ...

@RequestScoped

```
public class InvasionTweeter{
```

```
    public void afterBlogIntrude(
```

```
        @Observes @Invaded BlogArticle blogArticle) {
```

```
        ...
```

```
    }
```

```
}
```

Events

- Observers + Producers = Observer Pattern?
 - Jein
 - Producer von Observer & Observer von Producern
 - Verzögertes Aufrufen
 - Filtern/Auswählen von Events (z.B. @Invaded vs. @Any)

Interceptor-Binding

```
@InterceptorBinding  
@Retention (RUNTIME)  
@Target ({METHOD, TYPE})  
public @interface LoggerBinding {}
```

```
@LoggerBinding @Interceptor  
public class LoggingInterceptor implements Serializable {  
  
    private static final long serialVersionUID = 1L;  
    @Inject  
    private Logger log;  
  
    @AroundInvoke  
    public Object onMethodCall(InvocationContext context)  
        throws Exception {  
  
        ...  
    }  
}
```

Interceptor-Binding

```
@Stateful
@ConversationScoped
public class CheckoutServiceBean implements ... {
    ...
    @LoggerBinding
    public void checkout() {
        kudosCart.getCart().clear();
        conversation.end();
    }
}
```

■ Aktivierung in beans.xml

```
<interceptors>
  <class>
    de.holisticon.blog.friendlyInvaders.
    interceptor.LoggingInterceptor
  </class>
</interceptors>
```

Dekoratoren

@Decorator

```
public abstract class CheckoutServiceDecorator  
    implements CheckoutService, Serializable {
```

```
...
```

```
@Inject @Delegate
```

```
private CheckoutService checkoutService;
```

```
public void giveKudos(  
    BlogArticle blogArticle, float amount) {
```

```
    if (blogArticle.getAuthor().getName().
```

```
        endsWith("Erck")){
```

```
        logger.info("No Kudos in the form of money.");
```

```
    }else{
```

```
        checkoutService.giveKudos(blogArticle, amount);
```

```
    }
```

```
}
```

```
}
```

Dekoratoren

■ Aktivierung in beans.xml

```
<decorators>  
  <class>  
    de.holisticon.blog.friendlyInvaders.  
    decorator.CheckoutServiceDecorator  
  </class>  
</decorators>
```

■ Interzeptoren vs. Dekoratoren

- Jeden Java-Typ
- Technische Aspekte
- Konkreten Java-Typ
- Kennen Semantik
- Geschäftslogik

Produzenten

```
public class HcoLoggerFactory {  
    @Produces @HcoLogger  
    public Logger produceLogger(  
        InjectionPoint injectionPoint){  
        return LoggerFactory.getLogger(  
            injectionPoint.getMember().getDeclaringClass().  
                getName());  
    }  
}
```

- **Default Scope: @Dependent**
 - Scope der Zielklasse → Mehrere Instanzen möglich

Produzenten

@Decorator

```
public abstract class CheckoutServiceDecorator  
    implements CheckoutService, Serializable {
```

```
...
```

```
@Inject @HcoLogger
```

```
private Logger logger;
```

```
...
```

```
}
```

■ Einsatzgebiete

- Variierender Typ zur Laufzeit
- Individuelle Initialisierung

Beispielapplikation



Friendly Invaders: Give Kudos

http://localhost:8080/friendlyInvaders/invasion.jsf

Reader Google

Friendly Invaders. Give Kudos.

Retrospektive – Prozessmanagement im Branchenfokus

Am 14.04.2011 fand in Frankfurt am Main – naja, eigentlich war es das direkt nebenan liegende Offenbach – die Veranstaltung „Prozessmanagement im Branchenfokus“ statt. Zusammen mit Sönke Volquartz von unserem Kunden HanseMerkur habe ich die Erfolgsgeschichte zur „Automatisierten Leistungsabrechnung“ vorgestellt. Dass wir mit den angeschnittenen Themen ins Schwarze getroffen haben, war schon nach den ersten [...]

Give Roman Schlömmers Kudos.

Transaktionen in BPM/SOA

In BPM/SOA-Projekten stellt sich immer wieder recht schnell die Frage, wie transaktionales Verhalten in einem Prozess zu realisieren ist. Eine Menge von Aktivitäten eines Prozesses soll in einem transaktionalen Kontext ausgeführt werden – doch wie ist dies zu bewerkstelligen in einem Umfeld, in dem eine Prozesssteuerung eine Reihe unabhängiger Services orchestriert, die wiederum in ihren [...]

Give Jo Ehm Kudos.

Softwaremetriken: Lack of Cohesion in Methods

Lack of Cohesion in Methods, kurz: LCOM, ist eine Metrik, die sich aus den Prinzipien der Single-Responsibility und der Kohäsion (dt. Zusammenhalt) ableitet. Das Ziel in der objektorientierten Softwareentwicklung ist, ein möglichst hohes Maß an Zusammenhalt zu erreichen. Jede Softwareinheit, im objektorientierten Umfeld also jede Klasse und Methode, soll für genau eine klar umrissene Aufgabe [...]

Give Jan Weinschenker Kudos.

Ausblick Weld 2.0

- Verbesserung des Speichermanagements
 - Speicherbedarf um 50% bis 80% reduziert
- Gleichzeitige Bean Validierung und Bean Deployment
 - Multi-Core CPUs werden genutzt
 - Ladezeit verringert
- Bug Fixes

Zusammenfassung

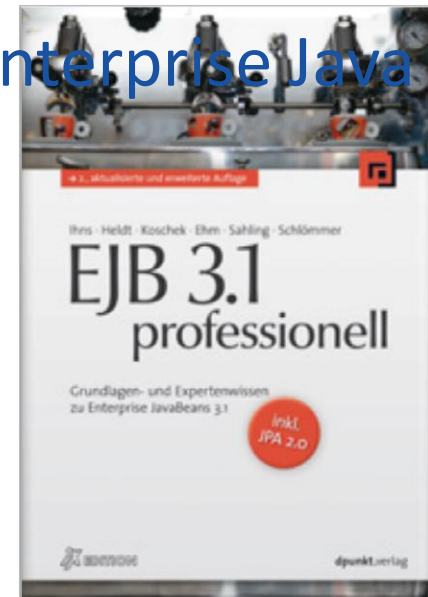
- DI Mechanismus (Dependency Injection)
 - @Inject
- Deklarative Verknüpfung von Beans
 - @Qualifier, @Alternative, @Stereotype
- Neue Scopes
 - @ConversationScope & @Dependent
- Interzeptorenkonzept
 - @InterceptorBinding
- Dekoratoren
 - @Decorator & @Delegate
- Beanverwaltung zur Laufzeit
 - @Producer

„Friendly Invaders“ & Tonspur

<http://blog.holisticon.de>

Darüber hinaus...

„Grundlagen- und Expertenwissen zu Enterprise Java Beans“



Vielen Dank!

