jsf-academy.com

Andy Bosch | www.jsf-academy.com

JavaServer Faces und CDI

Das neue Dreamteam

jsf-academy.com

Agenda

- Wenige Worte zu CDI allgemein
- HalloWelt mit CDI + JSF
- Qualifiers
- Stereotypes
- Conversations
- Fazit

Ein paar Worte zu mir

- Name: Andy Bosch
- Freiberuflicher Trainer, Coach, Entwickler, …

- Fokus auf JSF, Portlets und „angrenzenden Technologien“
- Mitglied der Expert Groups des JSR-301, JSR-329 und JSR-344 (JSF 2.2)

- Herausgeber von www.jsf-academy.com

jsf-academy.com

Agenda

- **Wenige Worte zu CDI allgemein**
- HalloWelt mit CDI + JSF
- Qualifiers
- Stereotypes
- Conversations
- Fazit

Was ist CDI? – Technisch gesprochen

- CDI ist ein Komponentenmodell, das es erlaubt, verschiedene Arten von Komponenten mittels loser Kopplung miteinander zu verbinden.
  Die Kopplung von Komponenten erfolgt typsicher.

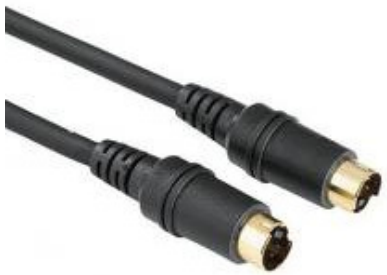
- CDI ist ein JavaEE-Standard

- CDI ist ein Name für einen Standard, der ursprünglich mal unter dem Namen Web Beans (basierend auf Ideen von JBoss Seam) gestartet wurde

- CDI ist eine Sammlung von Services, die beim Zusammenkitten von Enterprise Beans und User Interface Beans hilfreich ist.

Was ist CDI? – eher marketingmässig

- CDI ist die Next-Generation Dependency Injection in Java EE
- CDI bringt die besten Ideen von Seam 2, Spring und Google Guice zusammen
- CDI macht JavaEE flexibler, testbarer und erweiterbarer

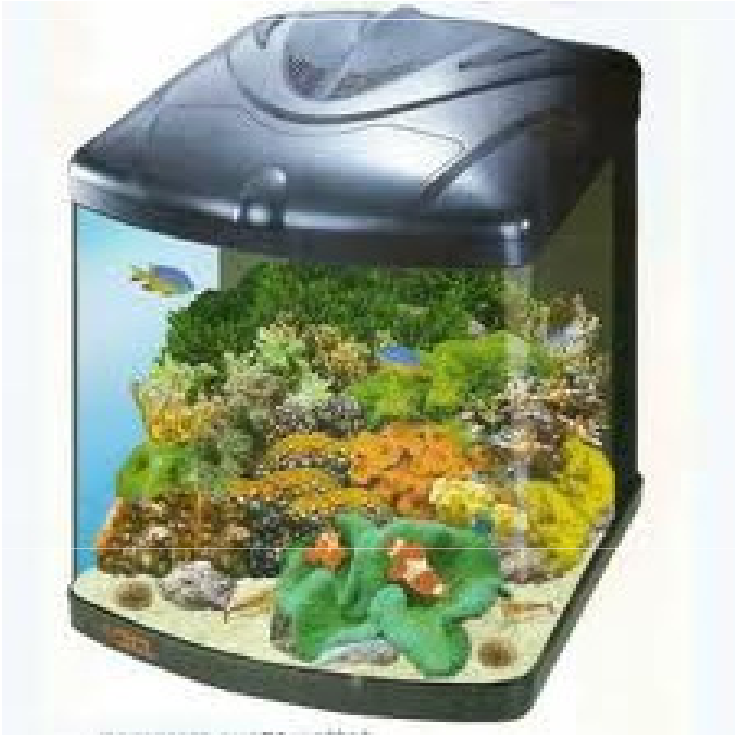


Spezifikation und Implementierung

- Basis ist der JSR-299, ein Standard des JCP
- Aktuell (Herbst 2011) sind drei Implementierungen geläufig:
  - JBoss Weld (Reference implementation)
  - Apache OpenWebBean (OWB)
  - CanDI (Caucho implementation included in the Resin server)

Laufzeitumgebungen für CDI

- Java SE
- Servlet Container
  - Tomcat
  - Jetty
- Application Server
  - JBoss
  - Glassfish
  - …



Hauptmerkmale von CDI

- **Contexts**
  CDI Komponenten können stateful sein und weisen einen dedizierten Lifecycle auf

- **Dependency Injection**
  CDI Komponenten können in andere CDI Komponenten typsicher injiziert werden

CDI: Everything is a bean

- Ein Bean wird „CDI Bean“ genannt, wenn der Lifecycle dieser Instanz durch den Container gemäß der CDI Spezifikation verwaltet wird
- Ein CDI Bean ist ein kontextbehaftetes Objekt, das einen dedizierten Zustand aufweist.
- Konkret: Das Anlegen und Löschen von Objekten passiert nicht manuell, sondern über „CDI Magic“.

Wie sehe ich etwas? Am besten mit JSF.

- Um CDI in Action zu betrachten, kann ein Webprojekt oder auch ein Java-Main verwendet werden
- Am einfachsten ist es, ein JSF Projekt aufzusetzen, das dann auf CDI Beans zugreift
- JSF und CDI sind Bestandteile von Java EE und damit in jedem EE 6 kompatiblen Application Server vorhanden
- JSF und CDI kann auch in einem ServletContainer betrieben werden. Dann sind dazu weitere Installationsschritte notwendig.

jsf-academy.com

Agenda

- Wenige Worte zu CDI allgemein
- **HalloWelt mit CDI + JSF**
- Qualifiers
- Stereotypes
- Conversations
- Fazit

Sag mal „HalloWelt“ (1)

- JSF Seite mit Ausgabe eines Bean-Properties
- Beans ist mit CDI annotiert anstelle der JSF-Variante

```
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named
@RequestScoped
public class SimpleWelcomeBean  {

   private String text = "Welcome to CDI";

...
```

Exkurs: @ManagedBean

- JSF 2.0 kam mit einer neuen Annotation: **@ManagedBean**
- Es wird aktuell diskutiert, diese mit dem nächsten JSF-Release schon wieder auf **Deprecated** zu setzen.
- Grund ist, dass bei Erscheinen von JSF 2.0 der JavaEE Standard noch nicht fertig war (also auch nicht CDI), daher wurde eine Alternativlösung benötigt. @ManagedBean ist nur temporär notwendig gewesen.

Sag mal „HalloWelt“ (2)

- Auf der JSF-Seite wir direkt die Property ausgegeben
- Konvention: Beanname = Klassenname + erster Buchstabe klein

```
...
<h:body>

  From CDI:
    <h:outputText value="#{welcomeBean.text}" />

</h:body>
</html>
```

EL Namen ändern

- Die Konvention des Beannamens in der EL kann überschrieben werden.

```
@Named(value="welcomeBean")
```

```
@Named("welcomeBean")
```

HelloWorld mit Injection

```
@Named("welcomeBean")
@RequestScoped
public class WelcomeBean {

@Inject
private PersonBean personBean;
```

```
@SessionScoped
public class PersonBean implements Serializable {


private String name = "Donald Duck";
```

Agenda

- Wenige Worte zu CDI allgemein
- HalloWelt mit CDI + JSF
- **Qualifiers**
- Stereotypes
- Conversations
- Fazit

Qualifiers

- Qualifiers werden verwendet, wenn mehre Komponenten des gleichen Typs verwendet werden.
- Mit Qualifiers können mehrere Implementierungen einer bestimmten Bean erzeugt werden.
- Technisch ist ein Qualifier eine Annotation
- Ohne explizite Angabe des Qualifiers verwendet CDI automatisch den @Default Qualifier

Beispiel: Produkteliste

- Producthandler mit DAO und Elementen

```
@Named
@SessionScoped
public class ProductHandler implements Serializable {

  @Inject
  private ProductDAO productDAO;

  private List<ProductEntity> elements;

...
```

DAO Interface

```
public interface ProductDAO {

   public List<ProductEntity> loadAllProducts();

}
```

- Beachte: Das Interface ist **nicht** annotiert.

DAO Implementierungen

```
public class DatabaseProductDaoImpl
    implements ProductDAO, Serializable {

    ...
}
```

```
public class FilesystemProductDaoImpl
    implements ProductDAO, Serializable {

    ...
}
```

- Auch hier wieder keine Annotation.

Ergebnis bei Nicht-Eindeutigkeit

- Es wird eine Exception geworfen, wenn die CDI Runtime keine eindeutige Zuordnung vornehmen kann.
- Wir müssen der CDI Runtime die Information mitgeben, welche konkrete Implementierung wir möchten

→ Qualifiers

CDI mit Qualifiers – ein Beispiel

```
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.FIELD})
public @interface DatabaseLike {}
```

```
@DatabaseLike
public class DatabaseProductDaoImpl
                implements ProductDAO, Serializable {
```

```
@Named
@SessionScoped
public class ProductHandler implements Serializable {

   @Inject @DatabaseLike
   private ProductDAO productDAO;
...
```

jsf-academy.com

Agenda

- Wenige Worte zu CDI allgemein
- HalloWelt mit CDI + JSF
- Qualifiers
- **Stereotypes**
- Conversations
- Fazit

Stereotypes

- Stereotypen ermöglichen es, gemeinsame Metadaten in Form einer neuen Annotation zusammenzufassen.

- Ein Stereotyp kapselt eine beliebige Kombination aus
  - Einem Default Scope
  - Einer Menge von Interceptor Bindings

Ein eigener Stereotype

- In JSF sollte eine Unterscheidung von Modell Managed Beans und Controller Managed Beans stattfinden.

→ Definition einer @UIController Annotatation

```
@Stereotype
@Named
@RequestScoped
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface UIController {}
```

Verwendung von @UIController

- Sämtliche JSF Controller, also Beans, die lediglich UI-Logik ausführen, können direkt mit @UIController annotiert werden.
- Bessere Verständlichkeit des Codes
- Änderungen können zentral in der Annotation vorgenommen werden

```
@UIController
public class InvoiceUIController {
    ...
}
```

Agenda

- Wenige Worte zu CDI allgemein
- HalloWelt mit CDI + JSF
- Qualifiers
- Stereotypes
- **Conversations**
- Fazit

Conversations

- CDI bietet u.a. auch den Conversation Scope:
  Länger als ein Request, kürzer als eine Session
- @ConversationScoped
- Die Conversation wird programmatisch gestartet und beendet.
- Es können damit mehrere „Arbeitsstränge“ parallel bearbeitet werden, z.B. mehre Tabs oder Browser Fenster

Conversations - Example

```
@ConversationScoped
public class ProductDetailBean {
    ...
}
```

```
@Inject private Conversation conversation;

conversation.begin();

conversation.end();
```

Conversations im Zusammenspiel mit JSF

- Ein Modellobjekt wird in den Conversation-Scope gelegt.
- Eine neue Conversation kann z.B. bei Öffnen eines neuen Tabs erfolgen („Open in new Tab“). Hier kann man das JSF-Tag <h:link> verwenden.
- Auf der Zielseite muss die Conversation gestartet werden, z.B. mit einem SystemEvent.

Agenda

- Wenige Worte zu CDI allgemein
- HalloWelt mit CDI + JSF
- Qualifiers
- Stereotypes
- Conversations
- **Fazit**

Fazit

- CDI bietet einige sehr hilfreiche Ergänzungen zu JSF.
- CDI füllt an mehreren Stellen die Lücken, die eine reine JSF-Entwicklung mit sich bringt.
- Eine Kombination von JSF+CDI ist sehr einfach zu realisieren.
- CDI kann **weit mehr**, als was hier gezeigt wurde.

Fragen?

Gerne E-Mail an mich:
andy.bosch@jsf-academy.com

Twitter
@andybosch

Oder einfach mal ein paar JSF- und CDI-Tutorials austesten auf:
www.jsf-academy.com