5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Ausgewachsen

JPA – Ein Standard wird erwachsen

Arne Limburg

open knowledge **GmbH**

Agenda

1. Historie
2. JPA Core
3. Advanced Concepts
4. Integration mit Java EE

Java Persistence – Ausgangslage

- JDBC
- EJB 1.0 (März 98) – Bean-managed Persistence
- EJB 1.1 (Dez. 99) – Container-managed Persistence
  - Keine Beziehungen (bzw. container-abhängig)
- Parallel: JDO

Java Persistence – Ein weiter Weg

- EJB 2.0 (Aug. 01) – Container-managed Persistence
  - Beziehungen standardisiert
  - EJBQL
- EJB 2.1 (Nov. 03)
  - Erweiterung EJBQL (Aggregate Functions, order by, …)
- Open Source auf der Überholspur

→Hibernate als Alternative in Java SE (Erste Versionen Anfang 02)

JPA

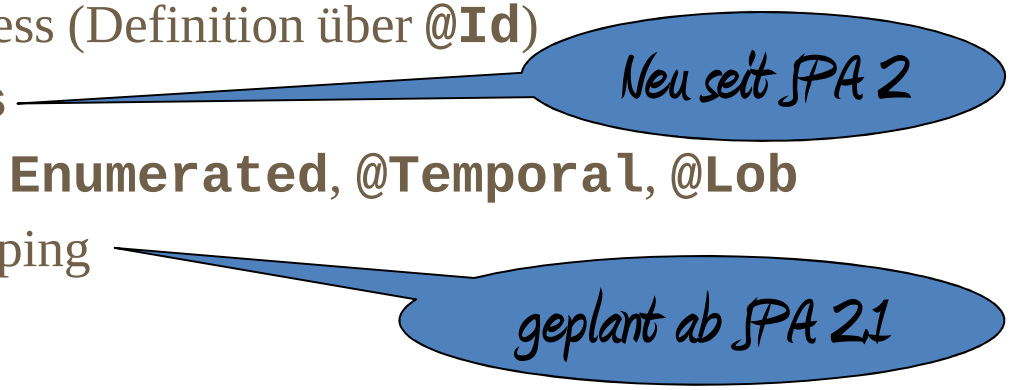
- Ein Neubeginn mit EJB 3.0
  - CMP eine Sackgasse
  - OR-Mapping von POJOs
  - Vorbilder: Hibernate, Toplink, „JDO“
- Herauslösung aus der EJB-Spec
  - Möglichkeit der Nutzung in Java SE von vornherein
  - EJB 3.0 Persistence (= JPA 1.0)
  - JPA 2 Teil von Java EE 6 (komplett unabhängig von EJB 3.1)

Agenda

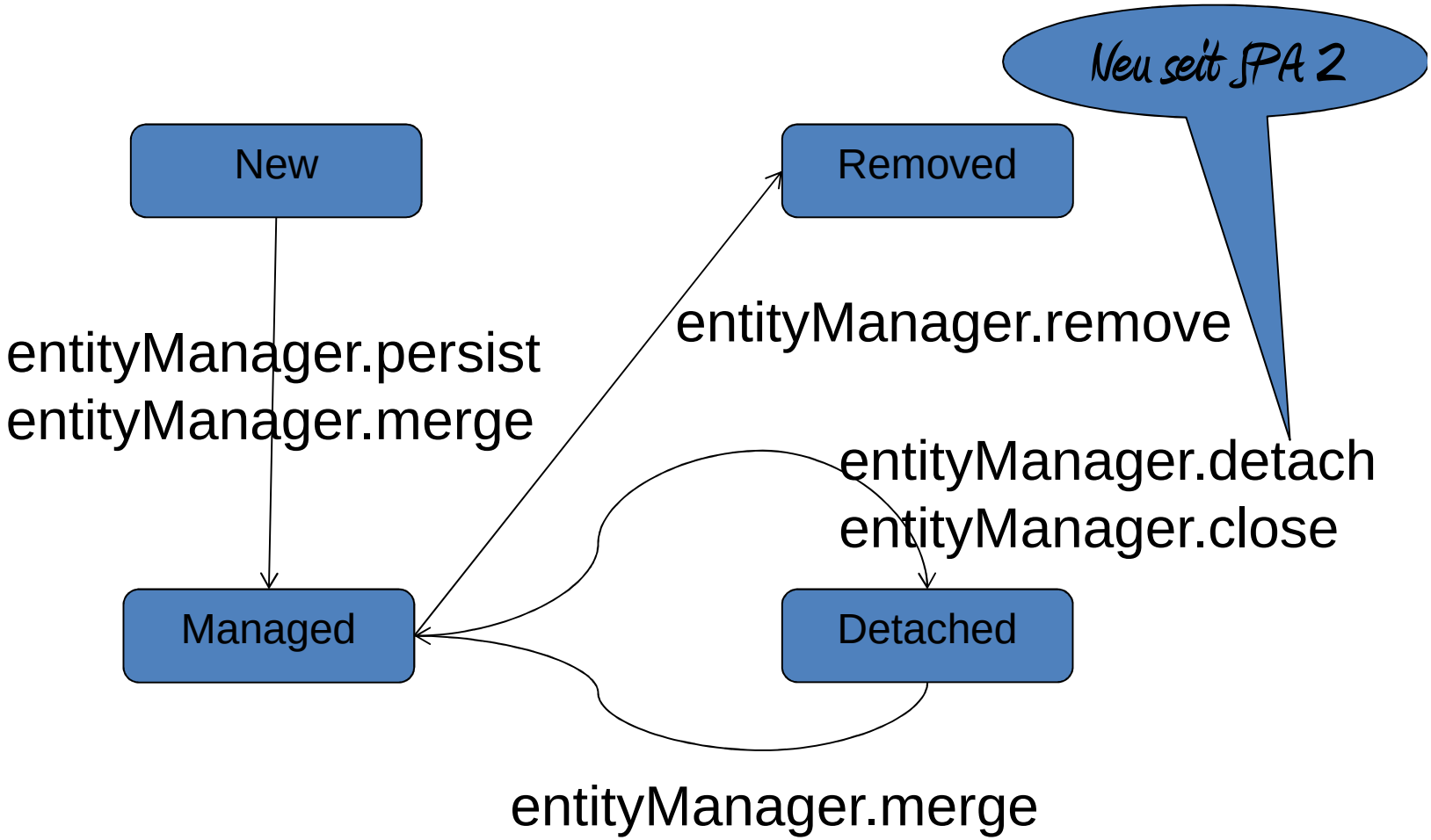
1. Historie
2. JPA Core
3. Advanced Concepts
4. Integration mit Java EE

OR-Mapping mit JPA

- Definition einer JPA-Entität
  - Parameterloser Konstruktor (nicht zwingend **`public`**)
  - Annotations (**`@Entity`**, **`@Id`**)
  - Alternativ über XML
- Persistente Attribute
  - Alle Attribute sind persistent (Ausnahme über **`@Transient`**)
  - Feld- vs. Property-Access (Definition über **`@Id`**)
  - Ändern über **`@Access`**
  - Einflussmöglichkeiten: **`Enumerated`**, **`@Temporal`**, **`@Lob`**
  - Eigenes Datentyp-Mapping



Entity-Lebenszyklus



Association-Mapping mit JPA

Assoziationen

- Typen
  - Bag, Set (kein SortedSet!), List, Map
- Varianten
  - @OneToMany, @ManyToOne, @ManyToMany, @OneToOne
  - Unidirektionale DB-Beziehungen

    → Uni- oder bidirektionale Objektbeziehungen

Association-Mapping mit JPA

Bidirektionale Assoziationen

- Eine Seite ist immer die „owning side“
  - Ohne Zwischentabelle immer die -to-one-Seite (evtl. mit **`@Join…`**)
  - Die andere Seite hat **`mappedBy="…"`**
  - Nur Änderungen an der „owning side“ werden gespeichert
- Konsistenz im Code sicherstellen

Association-Mapping mit JPA

Cascading

- EntityManager-Operationen

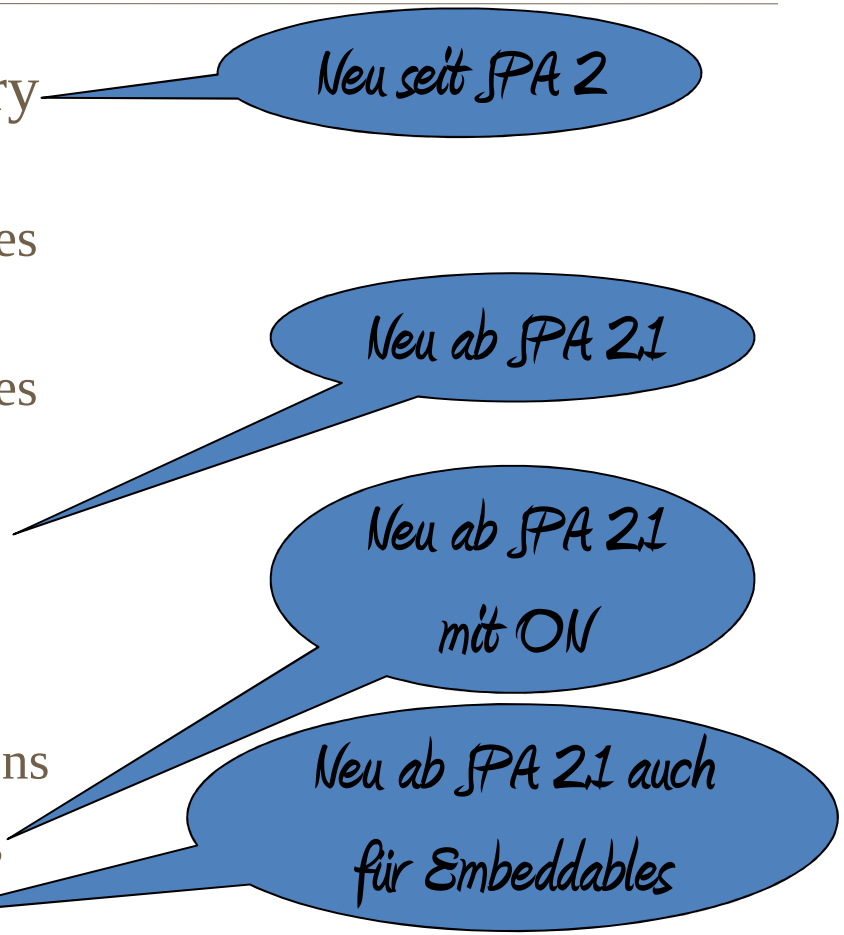
  persist, refresh, remove, merge, detach

- Änderungen im Objekt-Modell
  - Hinzufügen von neuen Objekten
  - Entfernen von Objekten (orphan-removal)
    - Unterstützt von **`@OneToOne`** und **`@OneToMany`**
    - Löschung des Objektes bei Kappung der Beziehung

    → Achtung! Nur anwenden bei Parent-Child-Beziehungen

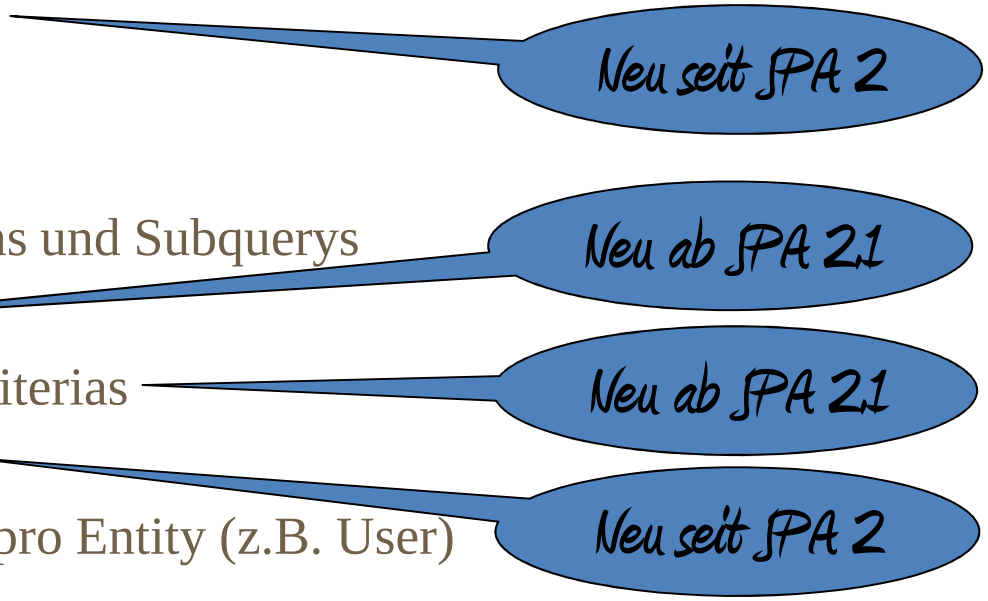
JPA Queries

- Query bzw. TypedQuery
- JPQL
  - „inline“ vs. Named Queries
- SQL
  - „inline“ vs. Named Queries
  - ResultSetMapping
  - StoredProcedure-Queries
- JPQL – Joining
  - Outer Join vs. Inner Join
  - Navigation über Collections
  - Join mit Zusatzbedingung
- Bulk Update/Delete



JPA Criteria Queries

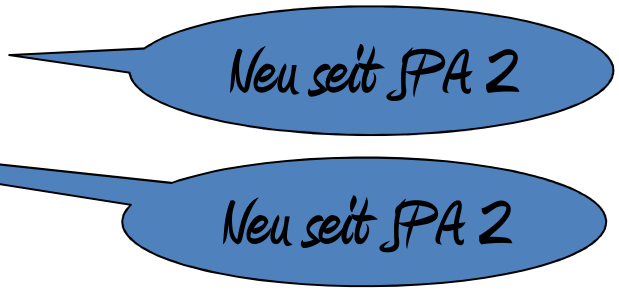
- JPA Criteria-Queries
  - Objektorientiert
  - Dynamisch
  - Unterstützung von Joins und Subquerys
  - Downcasting
  - Update- und Delete-Criterias
- Typesafe Metadata
  - Eine Metadata-Klasse pro Entity (z.B. User)
  - Namenskonvention User_
  - Deklaration aller Klassenattribute als statische Member vom Type Attribute
  - Automatische Generierung durch IDE



OR-Mapping mit JPA

Embedded Objects

- Parameterloser Konstruktor (nicht zwingend **`public`**)
- Annotations
  - Markierung mit **`@Embeddable`**
  - Referenziertes Attribut mit **`@Embedded`** bzw. **`@EmbeddedId`**
- Referenzierung von weiteren Objekten
- Collection of Elements
  - Markierung mit **`@ElementCollection`**
  - Collection von
    - Embeddables
    - Einfachen Datentypen

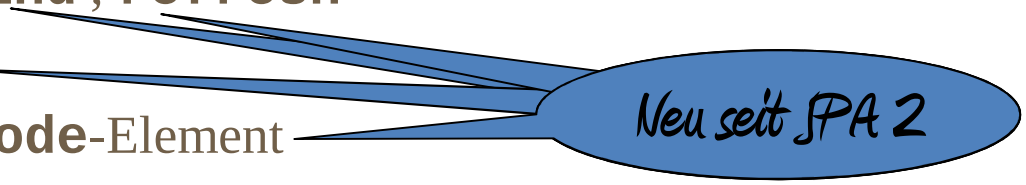


Agenda

1. Historie
2. JPA Core
3. Advanced Concepts
4. Integration mit Java EE

Locking mit JPA

- Optimistic vs. Pessimistic
  `OPTIMISTIC, OPTIMISTIC_FORCE_INCREMENT, PESSIMISTIC_READ, PESSIMISTIC_WRITE PESSIMISTIC_FORCE_INCREMENT`
- Programmatisches Locken
  - EntityManager **`lock`**, **`find`**, **`refresh`**
  - Query **`setLockMode`**
  - **`@NamedQuery lockMode`**-Element
- Parameter
  - **`javax.persistence.lock.scope`** (**`NORMAL`** vs. **`EXTENDED`**)
  - **`javax.persistence.lock.timeout`**



Lazy Loading

Problemstellung:

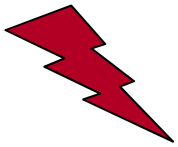
Wann werden die Daten aus der Datenbank geladen?

Beispiel:

```
User user = entityManager.find(User.class, id);

String roleName = user.getRole().getName();

//Wann wird die Rolle aus der Datenbank geladen?
```

Lazy Loading

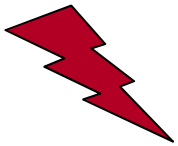
Möglichkeit 1: Wenn ein User geladen wird, werden alle angehängten Objekte mitgeladen!



Problem: Wenn alle Objekte zusammenhängen, wird bei jedem Select der gesamte Datenbank-Inhalt geladen!

Lazy Loading

Möglichkeit 2: Anhängende Objekte werden nie mitgeladen!



Problem:

```
User user = entityManager.find(User.class, id);

String roleName = user.getRole().getName();
//führt zu LazyInitializationException
```

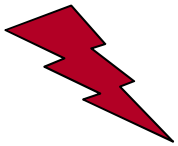
Lazy Loading

Möglichkeit 3: Anhängende Objekte werden nachgeladen, wenn sie gebraucht werden!

➔Vorgehen von JPA bei geöffnetem EntityManager
➔Bei geschlossenem EntityManager führt Zugriff auf nicht geladene Entitäten zu einem Fehler

Lazy Loading

Um das Nachladen zu realisieren, verwenden die JPA-Implementierungen entweder Proxies oder Byte-Code-Enhancement

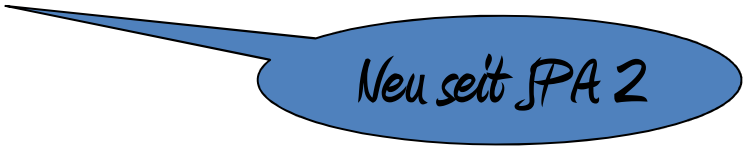


Problem: Bei Ableitungshierarchien gibt es Situationen, in denen der korrekte Typ nicht ermittelt werden kann

Lazy Loading

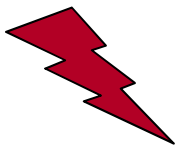
Beeinflussungsmöglichkeiten des Nachladens

- Mapping (über fetch-Attribute)
  - @ManyToOne (default EAGER)
  - @OneToOne (default EAGER)
  - @OneToMany (default LAZY)
  - @ManyToMany (default LAZY)
- Query
  - Angabe von **`join fetch`** in JPQL
  - Angabe von **`root.fetch(…)`** in Criteria
- Manuell (Test über **`PersistenceUnitUtil`**)



Second-Level Cache

- First-Level Cache (EntityManager Cache)
- Second-Level Cache
  - Provider-spezifisch
  - **`javax.persistence.sharedCache.mode`**
    `ALL, ENABLE_SELECTIVE, DISABLE_SELECTIVE, NONE`
  - Selektives Caching über **`@Cachable`** und
    `javax.persistence.cache.retrieveMode` (**`USE`**, **`BYPASS`**)
  - Beeinflussung des Inhalt über Cache-Interface oder
    `javax.persistence.cache.storeMode`
    (**`USE`**, **`BYPASS`** , **`REFRESH`**)
- Query-Cache (nicht standardisiert)



Problem: n+1 **`SELECT`**s

Agenda

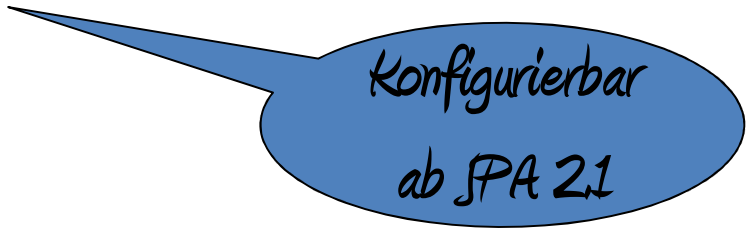
1. Historie
2. JPA Core
3. Advanced Concepts
4. Integration mit Java EE

Herausforderungen

- Scope des Persistence Contexts
  - → Lazy Loading von Entitäten
- Transaktionsverwaltung
  - Schreibende EntityManager-Interaktion nur innerhalb einer Transaktion
  - Lesende Interaktion nicht zwingend
- Persistence-Context-Propagation

Dimensionen

- EntityManager
  - Container- vs. Application-Managed
  - Transactional vs. Extended
- Transaction
  - JTA vs. Resource Local
  - Container- vs. Application-Managed
- Datasource
  - JNDI vs. local
  - JTA vs. Non-JTA



Container-Managed

Der EntityManager wird vom Java EE Container geöffnet und geschlossen

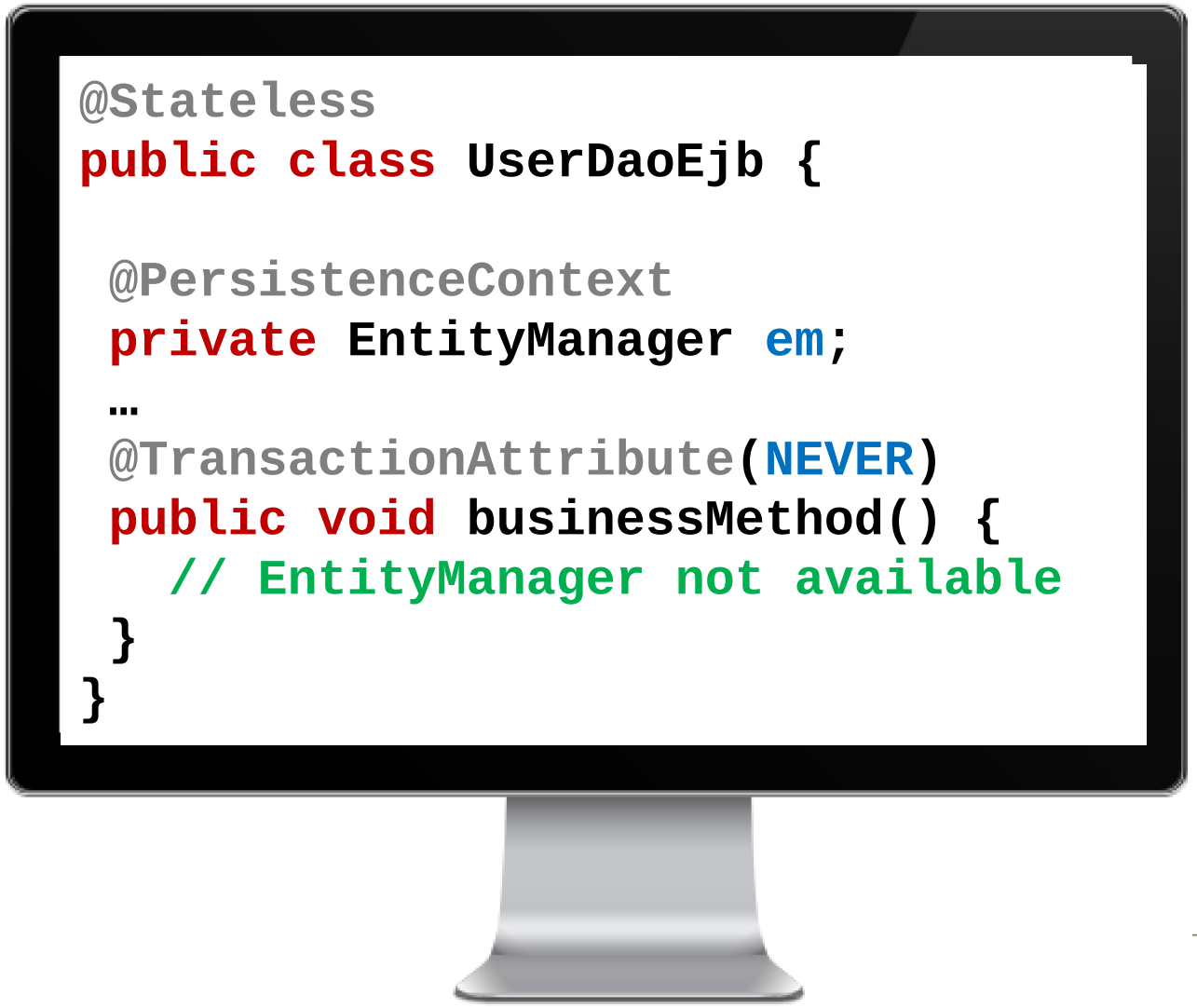
- Vorteil
  - Persistence-Context-Propagation
  - Transaktionsverwaltung vom Container
- Offen
  - Scope des Persistence Contexts

Container-Managed

Variante 1

- Session Bean (Stateful oder Stateless)
  - Transaktionsgrenze: Business-Methode (Transaction-Propagation)
  - Beeinflussung durch `@TransactionAttribute`
- Scope:
  - Transactional
  - Mit Persistence-Context-Propagation

Container-Managed



```
@Stateless
public class UserDaoEjb {

 @PersistenceContext
 private EntityManager em;
 …
 @TransactionAttribute(NEVER)
 public void businessMethod() {
   // EntityManager not available
 }
}
```

Container-Managed

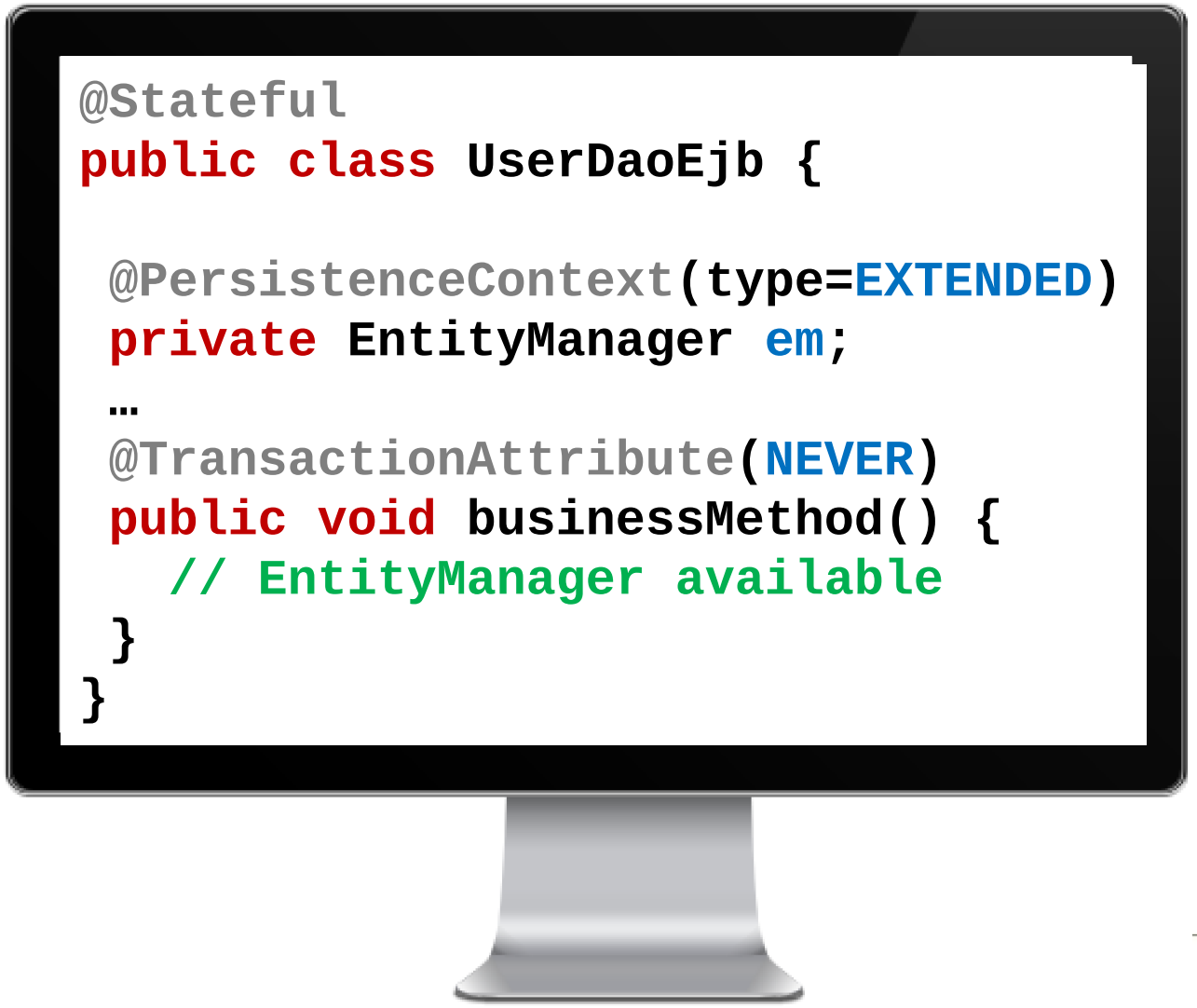
Variante 2

- Stateful Session Bean
- EXTENDED EntityManager
  - Transaktionsgrenze

    → wie bei Variante 1
  - Lesen ohne Transaktion

    → Lazy-Loading möglich
  - Scope Session

Container-Managed



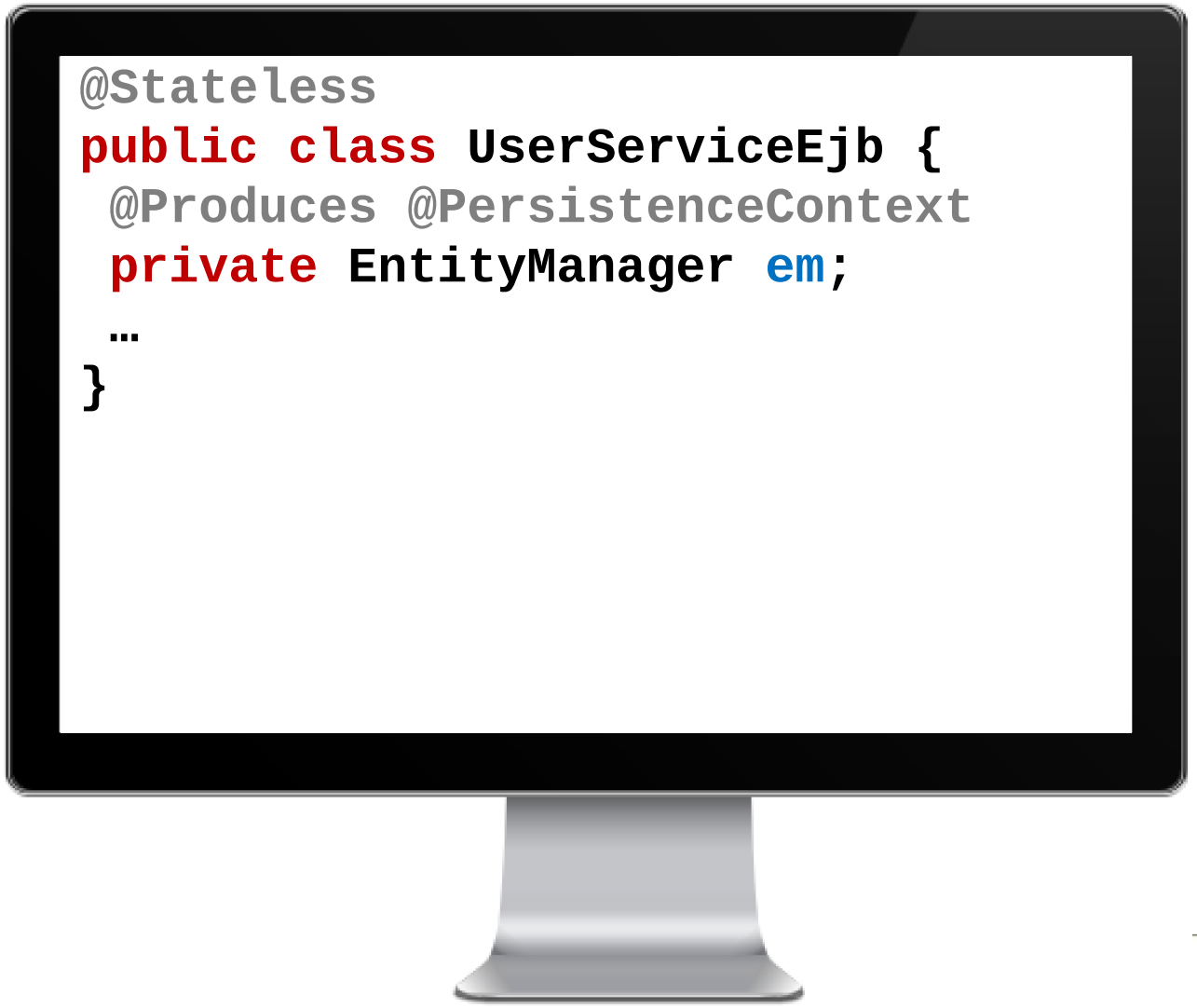
```
@Stateful
public class UserDaoEjb {

 @PersistenceContext(type=EXTENDED)
 private EntityManager em;
 …
 @TransactionAttribute(NEVER)
 public void businessMethod() {
   // EntityManager available
 }
}
```

Container-Managed

Zusatz-Feature Java EE 6

- Service-Schicht als EJBs
- DAO-Schicht als CDI Beans
- EJB stellt EntityManager zur Verfügung `@Produces`
- Injection des EntityManagers in CDI-Beans `@Inject`

Container-Managed



Container-Managed

Fazit

- Injection in Session Beans möglich
- Transaktionsgrenze
  - Methodenaufruf der Session Bean (inklusive Transaction-Propagation)
- Scopes
  - Transaction (alle Session Beans)
  - Session (Stateful Session Beans)

Application-Managed

- Erzeugung via
  `entityManagerFactory.createEntityManager(…)`
- Transaktionsbehandlung (Manuell)
  - JTA **`entityManager.joinTransaction()`**
  - RESOURCE_LOCAL
    `entityManager.getTransaction().begin()`
    `entityManager.getTransaction().commit()`
- Scope EXTENDED (Manuelle Verwaltung)

Application-Managed

- Probleme
  - Keine Persistence-Context-Propagation
  - Fehleranfälligkeit (Manuelle Transaktionen)
- Lösung
  - Verwaltung des EntityManagers über CDI
    - Transaktionsgrenze via Interceptor
    - Scope via CDI Scopes

➢Voller Flexibilität und Persistence-Context-Propagation

Application-Managed

5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Fragen und Antworten!

Arne Limburg

open knowledge **GmbH**

5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Arne Limburg

open knowledge **GmbH**

Firma

open knowledge **GmbH**
Bismarckstrasse 13
26122 Oldenburg

Fon: +49 (0)441 4082-0
Fax: +49 (0)441 4082-111

arne.limburg@openknowledge.de

offenkundiggut