1.– 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Java Persistence in Action

Einsatz von JPA 2.x in Java-EE-Anwendungen

Dirk Weil

GEDOPLAN

Dirk Weil

- GEDOPLAN GmbH, Bielefeld
  - IT Consulting
  - IT Training

- Java EE seit 1998
- Konzeption und Realisierung
- Vorträge
- Seminare
- Veröffentlichungen

- JPA im EE-Kontext
- JPA 2.1
- Reicht der Standard?

Java Persistence in EE-Anwendungen

- Basics:
  - **`META-INF/persistence.xml`** referenziert Datasource

```
<persistence …>
  <persistence-unit name="conference">
    <jta-data-source>datasource/conference</jta-data-source>
    …
```

  - EntityManager-Bereitstellung per Injektion

```
@PersistenceContext(unitName = "conference")
EntityManager entityManager;
```

Java Persistence in EE-Anwendungen

- The CDI way:

```
public class EntityManagerProducer {
  @PersistenceContext(unitName = "conference")
  @Produces
  EntityManager entityManager;
```

```
public class PersonRepository {
  @Inject
  EntityManager entityManager;
```

```
public class ProjektRepository {
  @Inject
  EntityManager entityManager;
```

```
public class FirmaRepository {
  @Inject
  EntityManager entityManager;
```

Entity Manager Lifecycle

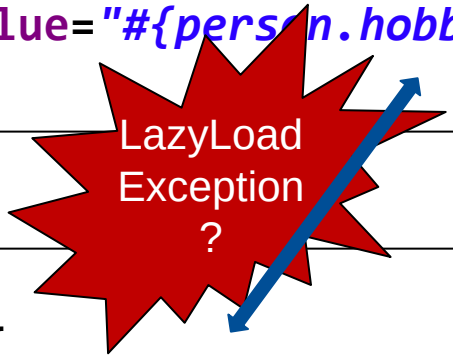
- EM-Lebensdauer ⇔ Entity-Management-Zeit
  - Entities werden detached
  - Nachladen von Lazy-Werten etc. nicht mehr möglich

alle evtl. benötigten Werte müssen geladen sein

- Remote-Anwendung (Swing, Java FX, … per Remoting, REST, …)
  - Entities im Client sind stets detached
- Web-Anwendung (JSF, …)
  - Render-Phase, Folgerequests?

Entity Manager Lifecycle

```
<h:dataTable value="#{personModel.personen}" var="person">
  …
    <h:… value="#{person.hobbies}" />
  …
```

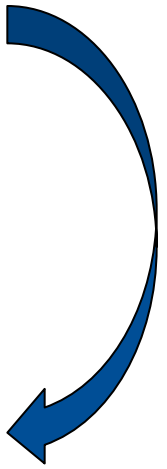



```
@Entity
public class Person {
  @ElementCollection(fetch = FetchType.LAZY)
  private List<String> hobbies;
```

```
@Model
public class PersonModel {
  public List<Person> getPersonen() {
    … personRepository.findAll() …
```

Entity Manager Lifecycle

- EM-Optionen:
  - Transaction-scoped
  - (Extended)*
  - Application-managed
- Patterns
  - Eager Loading
  - Open Entity Manager in View
  - Conversational Entity Manager



verschiedene Kombinationen möglich

*nur Stateful EJBs – nicht weiter betrachtet

Transaction-scoped Entity Manager

- Standard für container-managed Entity Manager

```
@Transactional
public class PersonRepository {
  @Inject
  EntityManager entityManager;

  public void persist(Person entity) { … }
  public List<Person> findAll() { … }
```

```
public class EntityManagerProducer {
  @PersistenceContext(unitName = "conference")
  @Produces
  EntityManager entityManager;
```

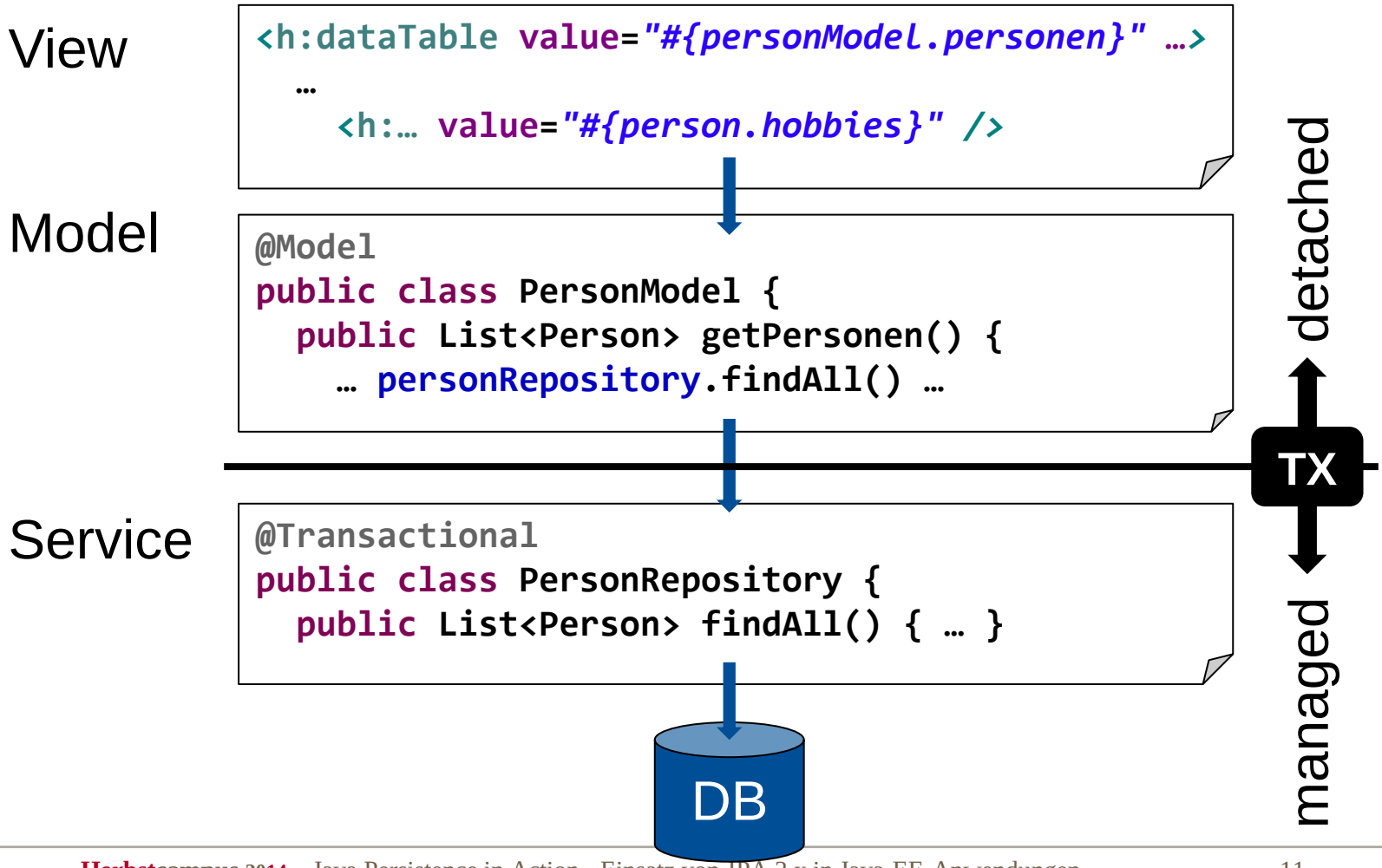
Transaction-scoped Entity Manager

- Start: Erste Nutzung des EM in einer TX
- Ende: TX-Ende ➔ alle Entities werden detached

- TX-Steuerung
  - per CDI Interceptor `@Transactional`
  - per EJB
  - mittels UserTransaction

  - häufig für (Teil-)Geschäftsprozesse
    (nur für CRUD wäre ein Antipattern)

Transaction-scoped Entity Manager



Transaction-scoped Entity Manager

- Eager Loading nötig
  - deklarativ

```
@ElementCollection(fetch = FetchType.EAGER)
private List<String> hobbies;
```

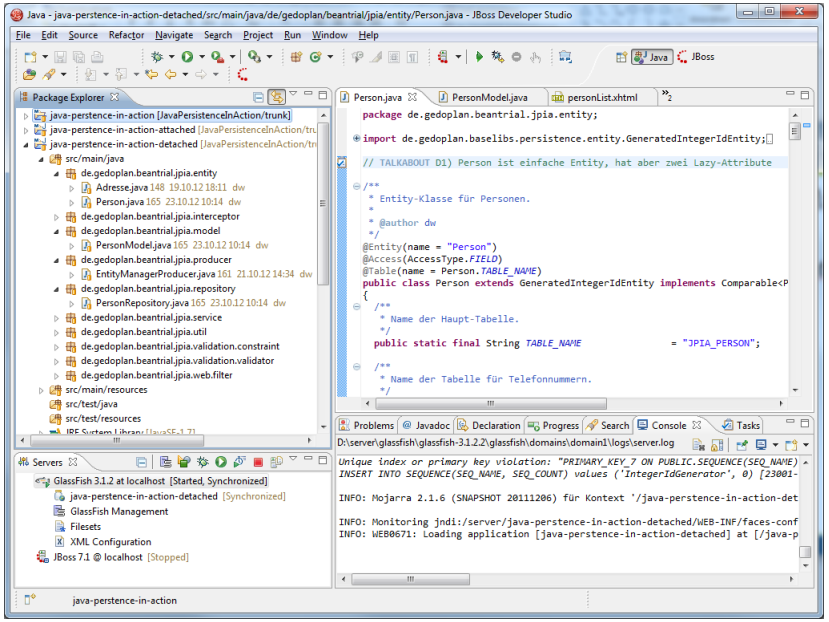
  - oder mittels Load Graph

```
@Entity
@NamedEntityGraph(name = "Person.TelAndHobbies",
  attributeNodes = { @NamedAttributeNode("telefonNummern"),
                     @NamedAttributeNode("hobbies") })
public class Person
```

```
Map<String, Object> prop = new HashMap<>();
prop.put("javax.persistence.loadgraph",
         entityManager.getEntityGraph("Person.TelAndHobbies"));
Person person = entityManager.find(Person.class, id, prop);
```

Transaction-scoped Entity Manager

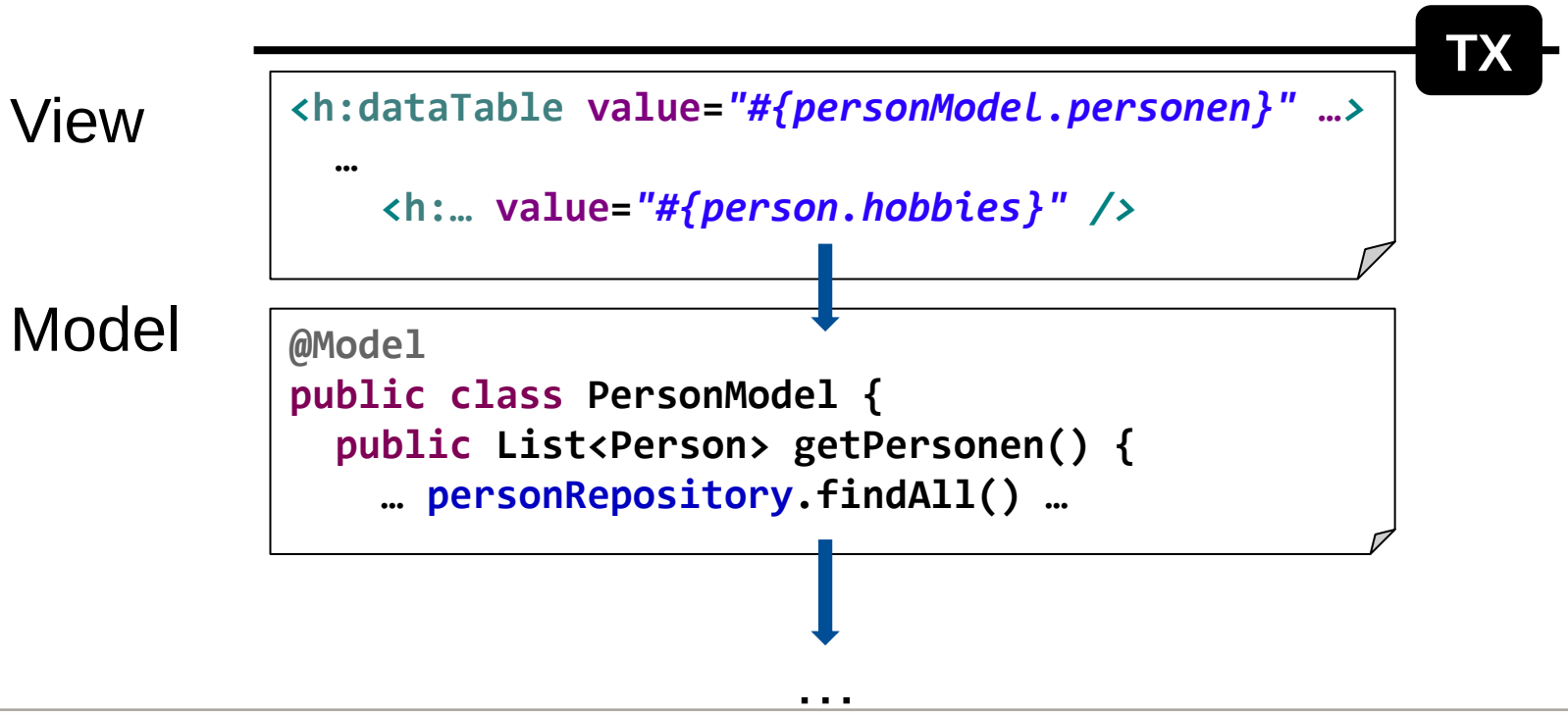
- Code Browsing:
  Personenverwaltung mit detached Entities



(github.com/dirkweil/javaPersistenceInAction)

Transaction-scoped Entity Manager

- Alternative: Transaction per Request
  - mittels Servlet Filter
  - mittels JSF Phase Listener



Transaction-scoped Entity Manager

- Einfaches Verfahren
- Eager Loading
  - ist aufwändig
  - hebelt Lazy-Optimierungen aus
- TX per Request
  - durchbricht 'normale' Architektur
  - nicht für Multi-Request-Fälle geeignet

Application-managed Entity Manager

- Nutzung von **`EntityManagerFactory`**
- Lifecycle wird von Anwendung bestimmt

```
public class EntityManagerProducer {
  @PersistenceUnit(unitName = "conference")
  EntityManagerFactory entityManagerFactory;

  @Produces
  public EntityManager createEntityManager() {
    return this.entityManagerFactory.createEntityManager();
  }

  public void closeEntityManager(@Disposes @Any EntityManager em) {
    if (em.isOpen())
      em.close();
  }
```

Application-managed Entity Manager

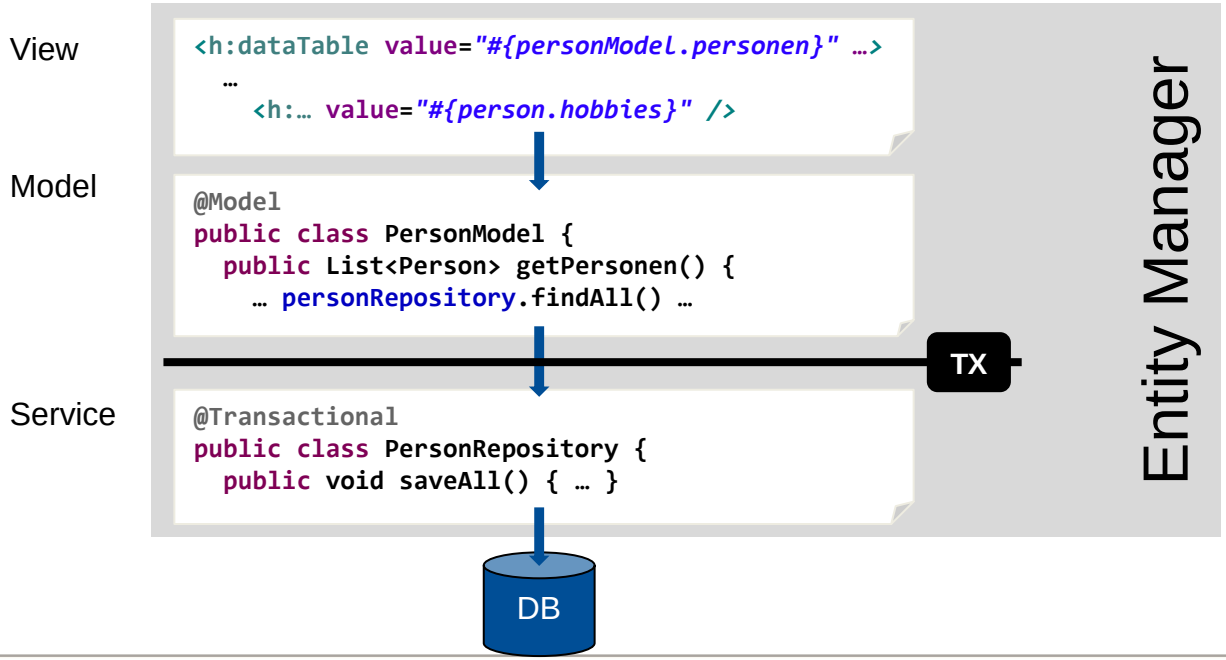
- **`persist`**, **`merge`**, **`remove`** auch ohne TX erlaubt
- Änderungen werden erst durch Commit abgelegt
- Entity Manager nicht automatisch TX-gebunden

```
@Transactional
public void saveAll() {
  this.entityManager.joinTransaction();
}
```

Request-scoped Entity Manager

- ≈ Open Entity Manager in View

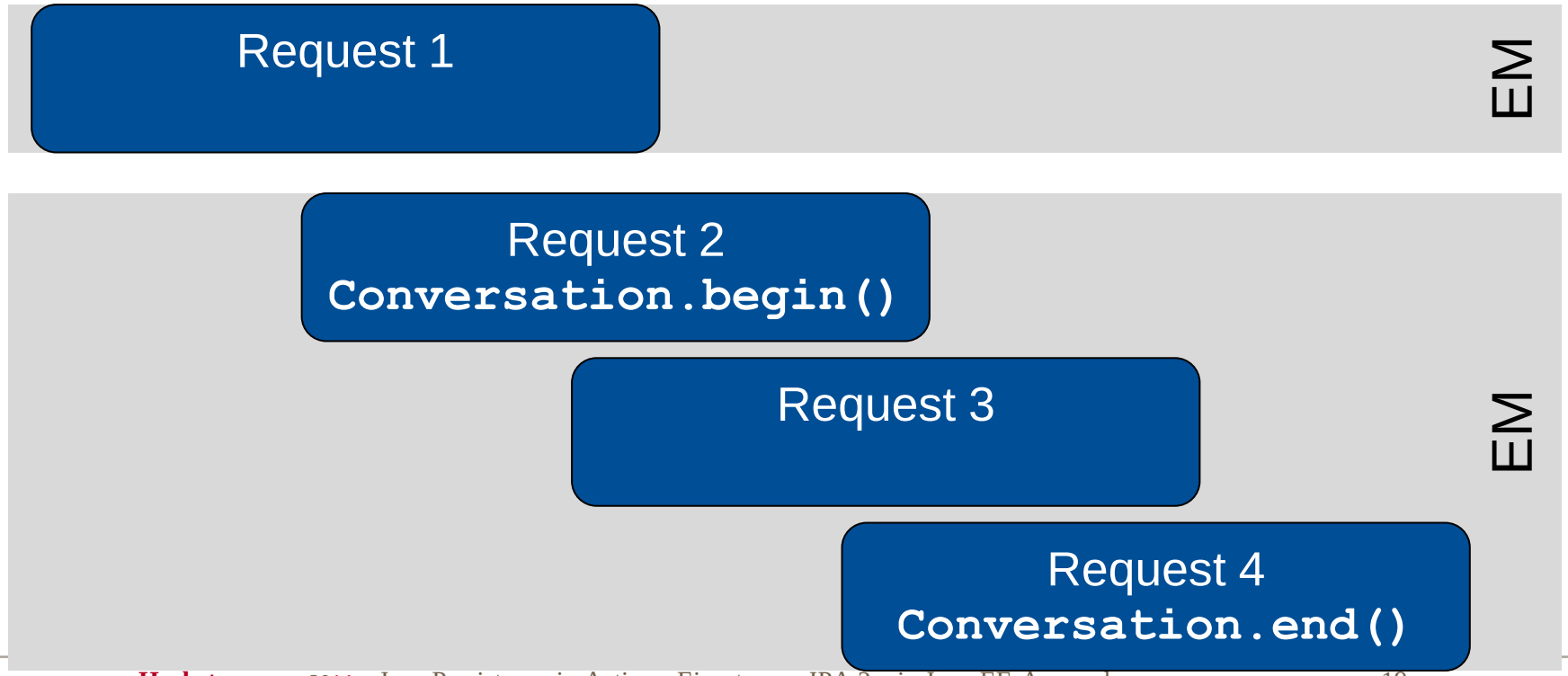
```
@Produces
@RequestScoped
public EntityManager createEntityManager() {
```



Conversation-scoped Entity Manager

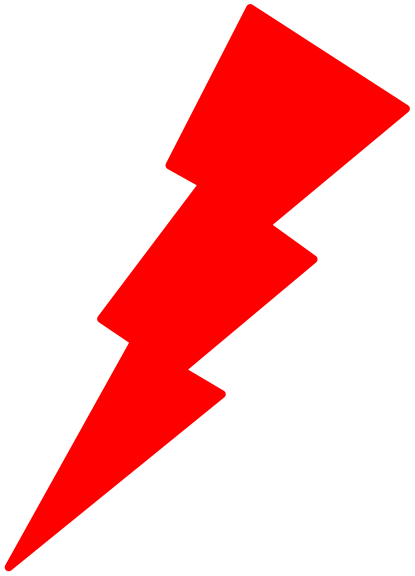
- Conversational Entity Manager

```
@Produces
@ConversationScoped
public EntityManager createEntityManager() {
```



Conversation-scoped Entity Manager

- ABER:
  - Conversation Scope ist passivating
  - Objekte in ihm müssen serialisierbar sein
  - **`EntityManager`** nicht **`Serializable`**
- Abhilfe: **`Serializable`**-Wrapper



Conversation-scoped Entity Manager

- **`Serializable`**-Wrapper:

```
@Produces
@ConversationScoped
public EntityManager createEntityManager() {
  EntityManager em = entityManagerFactory.createEntityManager();

  if (!(em instanceof Serializable)) {
    ClassLoader loader = em.getClass().getClassLoader();
    Class<?>[] ifs = { EntityManager.class, Serializable.class };
    EMInvocationHandler handler = new EMInvocationHandler(em);
    em = (EntityManager) Proxy.newProxyInstance(loader, if, handler);
  }

  return em;
}
```

Conversation-scoped Entity Manager

- **`Serializable`**-Wrapper:

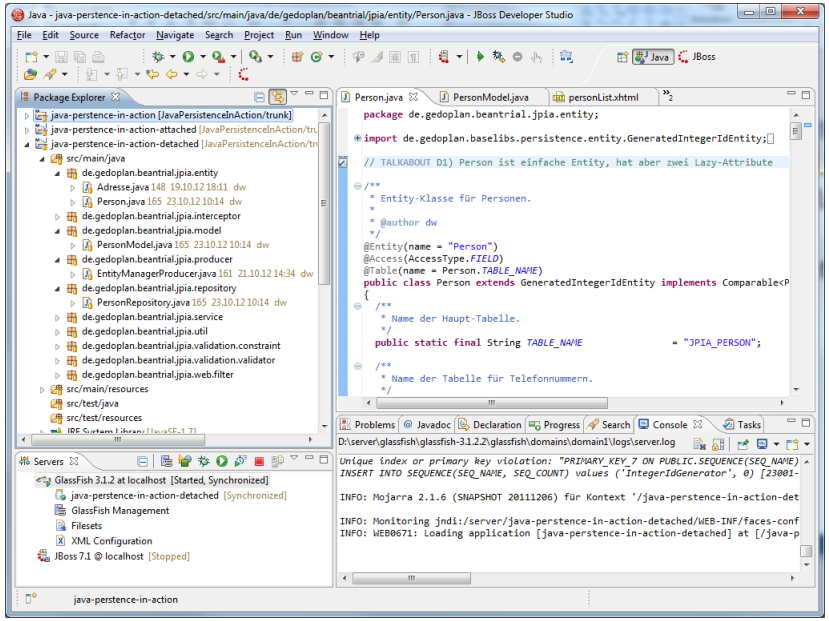
```
class EMInvocationHandler implements InvocationHandler, Serializable {
  private transient EntityManager em;

  public EntityManagerInvocationHandler(EntityManager em) {
      this.em = em;
  }

  public Object invoke(Object proxy, Method method, Object[] args)
    throws Throwable {
    if (this.entityManager == null)
      throw new IllegalStateException("EntityManager is null");

    return method.invoke(this.em, args);
  }
}
```

Conversation-scoped Entity Manager

- **`Serializable`-Wrapper:**
  - Prinzip Hoffnung:
    Wahrscheinlich keine Passivierung;
    wenn doch, ordentliche Fehlermeldung.
  - Ungeeignet für Replikation

Conversation-scoped Entity Manager

- Code Browsing:
  Personenverwaltung mit attached Entities



(github.com/dirkweil/javaPersistenceInAction)

SQL-Zugriffe im Beispiel

Aktion	TX-scoped EM	Conv-scoped EM
Anzeige aller Personen		
	`SELECT ID, ... FROM PERSON`	`dito`
Auswahl einer Person zum Editieren		
	`SELECT ID, ... FROM PERSON` `WHERE (ID = ?)`	
	`SELECT TEL, ... FROM PERSON_TEL` `WHERE (PERSON_TEL.Person_ID = ?)`	`dito` (Annahme: es werden zunächst nur Tel.-Nr. benötigt)
	`SELECT HOBBY, ... FROM PERSON_HOBBY` `WHERE (PERSON_HOBBY.Person_ID = ?)`	

SQL-Zugriffe im Beispiel

Aktion	TX-scoped EM	Conv-scoped EM
Ändern des Vornamens und Speichern der Person		
	`SELECT ID, ... FROM PERSON WHERE (ID = ?)`	
	`SELECT TEL, ... FROM PERSON_TEL WHERE (PERSON_TEL.Person_ID = ?)`	
	`SELECT HOBBY, ... FROM PERSON_HOBBY WHERE (PERSON_HOBBY.Person_ID = ?)`	
	`UPDATE PERSON SET VORNAME = ? WHERE (ID = ?)`	`dito` (Annahme: nur Vorname verändert)

TX- oder Conversation-scoped?

- Transaction-scoped EM
  - ✓einfaches Vorgehen
  - ✓speichert nur das, was explizit übergeben wird
- Conversation-scoped EM
  - ✓vermeidet Lazy-Load-Effekte im Web-Anwendungen
  - ✓erzeugt weniger DB-Zugriffe
  - ▪ speichert immer alle Änderungen
  - ✗benötigt mehr Speicher
  - ✗funktioniert nicht im Cluster oder für Remote-Clients

JPA 2.1: Converter

- **`@Convert`**
  - Explicit type / value mapping
  - replaces „User Types“ etc.
  - generalizes **`@Enumerated`**, **`@Temporal`**

```
@Entity
public class Country
{
  @Convert(converter = YesNoConverter.class)
  private boolean expired;
```

JPA 2.1: Converter

- `@Converter`
  - declares converter
  - can be auto-applied (`autoApply = true`)

```
@Converter
public class YesNoConverter implements AttributeConverter<Boolean, String> {
  public String convertToDatabaseColumn(Boolean fieldValue) {
    if (fieldValue == null) return null;
    return fieldValue ? "Y" : "N";
  }

  public Boolean convertToEntityAttribute(String columnValue) {
    if (columnValue == null) return null;
    return columnValue.equals("Y");
```

JPA 2.1: JPQL & Criteria Query Enhancements

- **ON**: Join filter

```
select p.name, count(b)
from Publisher p
left join p.books b
  on b.bookType = BookType.PAPERBACK
group by p.name
```

- **TREAT**: Downcast (includes filter)

```
select s from StorageLocation s
where treat(s.product as Book).bookType = BookType.HARDCOVER
```

- **FUNCTION**: Call DB function

```
select c from Customer c
where function('hasGoodCredit', c.balance, c.creditLimit)
```

JPA 2.1: JPQL & Criteria Query Enhancements

- Bulk Update/Delete for Criteria Query

```
CriteriaUpdate<Product> criteriaUpdate
  = criteriaBuilder.createCriteriaUpdate(Product.class);
Root<Product> p = criteriaUpdate.from(Product.class);
Path<Number> price = p.get(Product_.price);
criteriaUpdate.set(price, criteriaBuilder.prod(price, 1.03));

entityManager.createQuery(criteriaUpdate).executeUpdate();
```

- Stored Procedure Queries

```
StoredProcedureQuery query
  = entityManager.createStoredProcedureQuery("findMissingProducts");
```

JPA 2.1: CDI Injection in Entity Listener

```
public class CountryListener {
  @Inject
  private AuditService auditService;

  @PreUpdate
  public void preUpdate(Object entity) {
    this.auditService.logUpdate(entity);
  }
```

```
@Entity
@EntityListeners(CountryListener.class)
public class Country {
```

JPA 2.1: DDL Handling

- Create and/or drop db tables
  - Based on entity meta data (mapping)
  - SQL script
- Data load script

```
<persistence … >
  <persistence-unit name="test">
    …
    <properties>
      <property name="javax.persistence.schema-generation.database.action"
                value="drop-and-create" />
      <property name="javax.persistence.schema-generation.create-script-source"
                value="META-INF/create.sql" />
      <property name="javax.persistence.schema-generation.create-source"
                value="metadata-then-script" />
      <property name="javax.persistence.sql-load-script-source"
                value="META-INF/sqlLoad.sql" />
```

JPA 2.1: DDL Handling

- Write create and/or drop scripts

```
Writer createWriter = …; // File, String …

Map<String, Object> properties = new HashMap<>();
properties.put("javax.persistence.schema-generation.scripts.action",
               "create");
properties.put("javax.persistence.schema-generation.scripts.create-target",
               createWriter);

Persistence.generateSchema("test", properties);
```

JPA 2.1: Entity Graphs

- Declaration of lazy attributes to be loaded

```
@Entity
@NamedEntityGraph(name = "Publisher_books",
                  attributeNodes = @NamedAttributeNode(value = "books")))
public class Publisher
{
  …
  @OneToMany(mappedBy = "publisher", fetch = FetchType.LAZY)
  private List<Book> books;
```

- find parameter or query hint
  - **fetchgraph**: Fetch entity graph attributes only
  - **loadgraph**: Fetch eager attributes also

```
TypedQuery<Publisher> query = entityManager.createQuery(…);
query.setHint("javax.persistence.fetchgraph", "Publisher_books");
```

JPA 2.1 – Delta

- Dynamische Named Queries
  Query per API mit Namen versehen
- ConstructorResult für native Queries
  ähnlich JPQL / Criteria Query Constructor Expressions
- Unsynchronized Persistence Context
  Injektion eines Entity Managers, der nicht TX-gebunden ist

JPA 2.1 – Reicht das?

- Ja
  - für neue Anwendungen
  - in 90%+ der Fälle
- Aber:
  - Legacy Databases
  - Performance

JPA 2.1 – Reicht das?

- Legacy Databases

	EclipseLink	Hibernate
Optimistic Locking ohne Version	✓	✓
weitere Generatoren	✓	✓
Inheritance ohne Diskriminator Column	✓	✓
Polymorphe Assoziationen	✓	✓
Custom CRUD SQL	✓	✓
…		

JPA 2.1 – Reicht das?

- Performance

	EclipseLink	Hibernate
R/O Entities	✓	✓
Index	✓	✓
Eager Load Tuning	✓	✓
Batch Fetching	✓	✓
Partitioning	✓	✓
Cascading Delete in DB	✓	✓
…		

JPA 2.1 – Reicht das?

- Wunschliste ist nicht leer
  - Vieles bereits proprietär vorhanden
  - Könnte im Standard sein!

- Standard weiter entwickeln
  - Vote!
  - Engage!

More

- http://www.gedoplan-it-training.de
- http://www.gedoplan-it-consulting.de

- http://javaeeblog.wordpress.com/
- http://expertenkreisjava.blogspot.de/

- dirk.weil@gedoplan.de
- @dirkweil