Java Persistence API

Phillip Ghadir Oliver Tigges
phillip.ghadir@innoq.com oliver.tigges@gmx.de

Was wissen Sie in 45 Minuten?

- Wie man mit JPA entwickelt.
- Dass wir viele Eigenschaften von JPA ausgelassen haben.
- Dass Sie nicht auf eine Implementierung festgelegt sind?
- Wann empfiehlt sich ein Einsatz von JPA?

Agenda

- Der Standard
- Die Merkmale
- Das Programmier-Modell
- Die Möglichkeiten
- Das Fazit

JSR 220 - EJB 3 - Java Persistence API

- Spezifiziert "die" Standard-Schnittstelle für Java Persistenz Frameworks
- Basiert auf Java 5 und Annotationen
- Einheitliches Programmiermodell für
  - Java SE und
  - Java EE
- Abgrenzung zu EJB (2.x) CMP und JDO

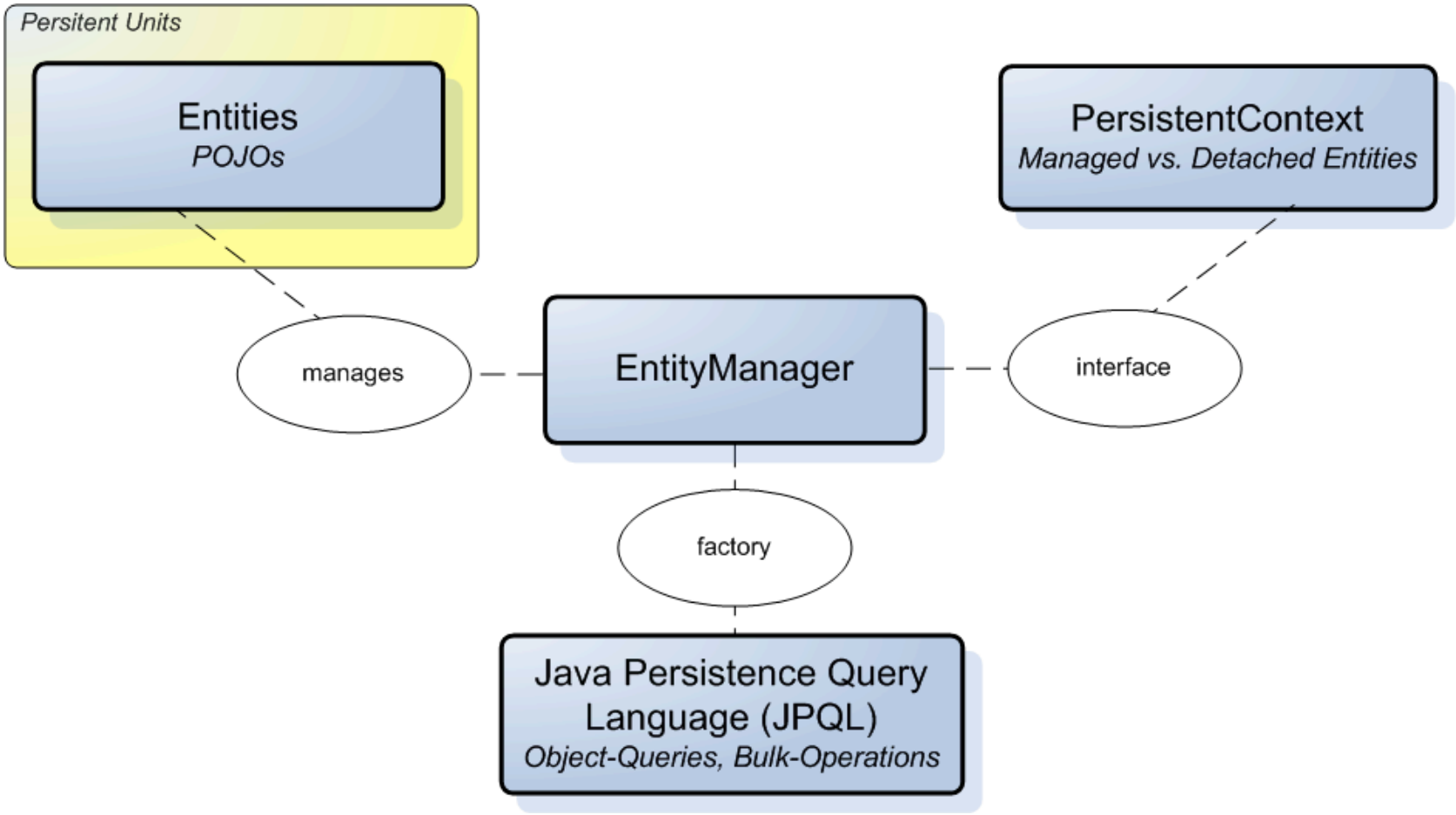
Merkmale einer JPA-Entität

- "Einfache" Java-Klasse mit
  - Default-Konstruktor
  - ausgewiesenem Id-Attribut
  - (bei Bedarf) serialisierbar
  - mit @Entity annotiert
- Keine Restriktion bzgl. Vererbung

Konsequenz

- Erzeugung über new
- Vererbung, Polymorphie möglich
- Persistenz ist nicht Teil der Entitätslogik
- Entitäten auch ohne Container testbar
- "Mobile" anstatt "entfernt aufrufbarer" Objekte

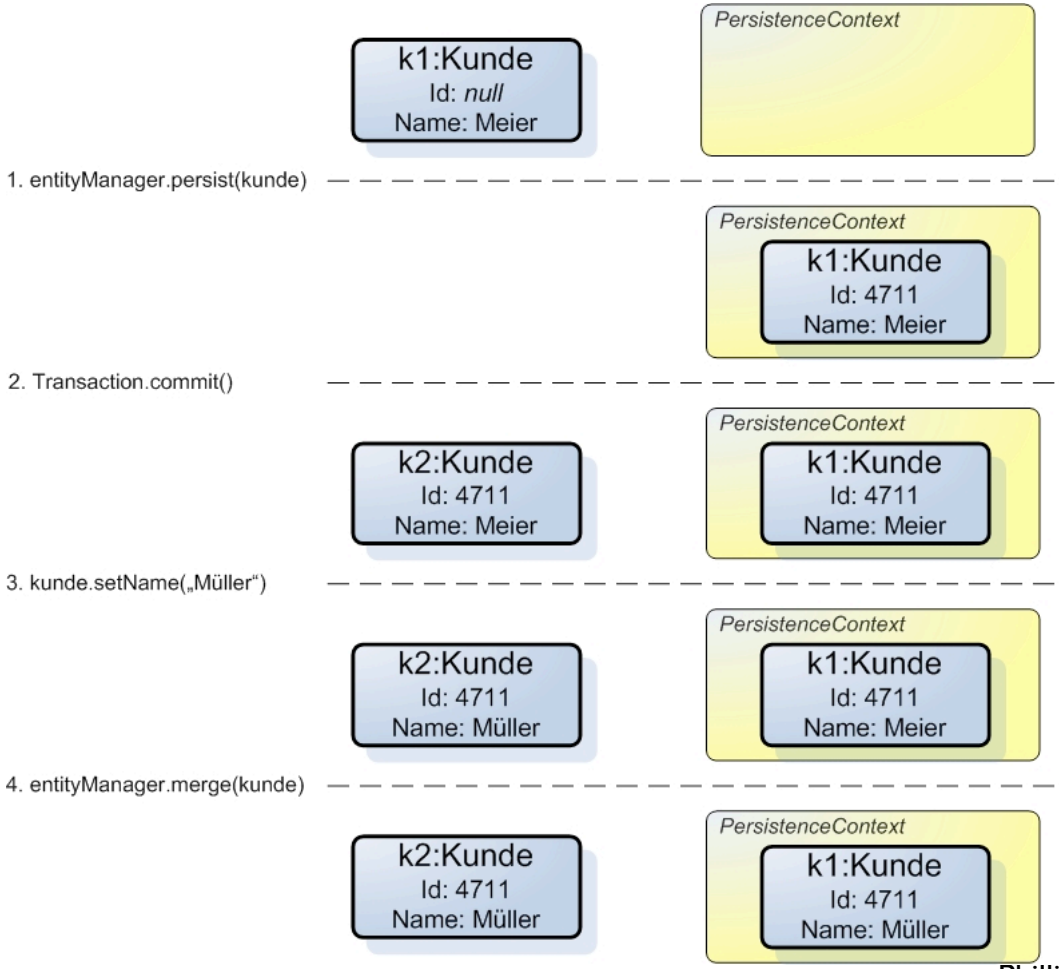
Die JPA-Basiskonzepte



Merkmal II - EntityManager

```
interface EntityManager {
    void persist(Object entity);
    T merge(T entity);
    void refresh(Object entity);
    void remove(Object entity);
    T find(Class<T> entityClass, Object primaryKey);
}
```

Transienter und Persistenter Zustand



Konsequenz

- Zentrale Stelle für Lebenszyklus-Methoden
- Interagiert mit Persistence Context
- EJB-Home gibt es nicht für JPA-Entitäten

Java Persistence Query Language (JPQL)

- Erweiterung von EJB-QL
- Erlaubt:
  - Projektionen
  - Aggregationen
  - Gruppierungen
  - Verknüpfungen
  - Massen-Updates und -Deletes

Projektionen

select w.name, w.level from Wizard w

Aggregationen

```
select w.name, COUNT(w.spells)
  from Wizard w
 group by w.name
 having COUNT(w.spells) > 2
```

Verknüpfungen (= Joins)

```
select w.name, s.name
from Wizard w
join w.spellbook s
```

Beziehungen in JPA

- Uni- oder Bidirektional
- Annotation per @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
- Kaskadierung von Operationen (Löschen, Neu-Laden, Speichern) konfigurierbar
- Konfigurieren des Ladeverhaltens (erst bei Zugriff, lazy, bzw. vorsorglich, eager)

Beziehungen in JPA

- Mögliche Rückgabetypen für "ToMany"-Beziehungen
  - Collection
  - List
  - Set
  - Map

Merkmal - Persistent Unit

- Verknüpfung einer Menge von Entitäten und einer Datenquelle
- Konfiguration in META-INF/ persistence.xml:

```
<persistence ... >
   <persistence-unit name="default" ... >
      <provider>
         oracle.toplink.essentials.PersistenceProvider
      </provider>
      <class>de.jfs2007.jpa.SampleEntity</class>
      <properties>
            <!-- Datasource-specific stuff -->
      </properties>
   </persistence-unit>
</persistence>
```

Programmier-Modell in Java SE

- Konfiguration einer Persistent Unit
- Instanziierens eines EntityManagers für eine Persistent Unit per:

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("default");

EntityManager em = emf.createEntityManager();
```

- Schreibende Operationen in Transaktionsklammern

Programmier-Modell in Java EE

- Injection des EntityManagers per annotiertem Attribut (in Session Bean, Servlet, o.ä.)
- Bereitstellung von JPA-Entities in eigenem Persistent Archive
  (ein Standard-JAR mit META-INF/persistence.xml)

```java
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class KundeServiceBean {

    @PersistenceContext(unitName="demoPU")
    private EntityManager entityManager;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void createKunde(){
        Kunde kunde = new Kunde();
        kunde.setName("Herr Müller");
        entityManager.persist(kunde);
    }

    @TransactionAttribute(TransactionAttributeType.SUPPORTED)
    public Kunde findKunde(Long kundennr){
        Kunde kunde = entityManager.find(Kunde.class, kundennr);
        return kunde;
    }
}
```

Programmier-Modell für Entitäten

```
@Entity
public class Wizard {

    @Id
    private String uuid;

    @OneToMany
    private Collection<Spell> spells;
}
```

Ergänzung um statische Queries

```
@Entity
@NamedQuery(name="findAllByLevel",
            query="SELECT wiz FROM Wizard wiz " +
                  " WHERE level = :level")
public class Wizard { ... }
...
Query query =
    entityManager.createNamedQuery("findAllByLevel");
```

Ergänzung um dynamische Queries

```
public void setSpellsActiveFor(String uuid, boolean active ) {
    String queryString = "UPDATE Spell s "+
                         " SET active = :active "+
                         " WHERE s.wizard.uuid = :uuid";
    Query query = em.createQuery(queryString);
    query.setParameter( "active", active );
    ...
    query.executeUpdate();
}
```

JPA-Entitäten und Verteilte Systeme

- Entitäten sind nicht entfernt aufrufbar
- Entitäten können serialisiert werden
- Serialisierung führt dazu, dass “Kopien unterwegs” sind.
- Kopien können über den EntityManager wieder in eine Transaktion eingefügt werden.

Möglichkeiten

- Einsparung von Klassen (im Vergleich zu EJB 2.x)
- DAO- (und TransferObject-) Pattern obsolet
- Testbarkeit der Fachlogik
- Reports können auf der Fachlogik realisiert werden!
- Mengenoperationen werden über die selbe Kapsel gebaut.

Fazit

- JPA ist der Persistenz-Standard für Java SE und EE
- verschiedene JPA-Implementierungen sind vorhanden und einsetzbar
- JPA bringt eine drastische Verbesserung gegenüber EJB 2 - Persistenz
- Akzeptanz bei Herstellern, IT-Management und Entwicklern

Fazit II

- Empfehlung für neue Projekte:
  - "Wenn Hibernate dann mit JPA"
  - "Wenn EJB dann JPA"

Referenzen

- EJB 3 - Java Persistence API Spec
- M. Keith & M. Schincariol:
  Pro EJB 3 - Java Persistence API; APress; 2006
- Ghadir, Pagop, Tigges:
  Einfach Speichern - Java Persistence API;
  JavaSPEKTRUM, 3/07
- ... und noch gaaanz viel mehr ...

Besuchen Sie uns im Netz!

http://innoq.com