Agenda

CRUD

Create

Read

Update

Detaching

Delete

Hibernate

Ausgangspunkt

- Annotierte POJO's

```
import javax.persistence.*;
@Entity
public class Spieler {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long spielerID;
    // Attribute,  Getter und Setter
      private String position;
    ….
}
```

Objekt erzeugen und in DB einfügen (JPA)


```
// EntityManager erzeugen
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("jpaDatabase");
EntityManager em = entityManagerFactory.createEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();

// Anwendungslogik
Spieler spieler = new Spieler("Neuer", "Torwart");
em.persist(spieler);
tx.commit();
em.close ();
```

Agenda

CRUD

Create

Read

Update

Detaching

Delete

Hibernate

Suche nach Geschäftsobjekten

- Finden anhand:
  - Primärschlüssel (Primary Key/PK)
  - JPA-QL - JPA Query Language
  - HQL – Hibernate Query Language
  - Native Query
  - JPA Criteria API
  - QBC Queries by Criteria (Hibernate)
  - QBE Queries by Example (Hibernate)

Lesen mit Primary Key

- **find(Class clazz, Serializable id)**
  - führt direkt zu einem Datenbankzugriff
  - Gibt null zurück wenn Instanz nicht vorhanden ist

- **getReference(Class clazz, Serializable id)**
  - führt zu einem Datenbankzugriff
  - Wirft Exception wenn Instanz nicht vorhanden ist
  - Kann Proxy zurück geben
    - **Hibernate: @Proxy(lazy=false|true)**

```
Spieler s = em.getReference(Spieler.class,  1L);
```

Agenda

CRUD

Create

Read

Update

Detaching

Delete

Hibernate

Objektattribute ändern

- Innerhalb eines Persistenz-Kontext werden Zustandsänderungen automatisch erkannt (**automatic dirty checking**).
  - Ein persist() benötigt man nicht!
  - abhängige Objekte werden aktuell gehalten (Transitive Persistenz)

Beispiel für Änderung


```
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Spieler spieler = em.find(Spieler.class, id);
spieler.setPosition("Innenverteidiger");
em.getTransaction().commit();
em.close () ;
```

Agenda

CRUD

Create

Read

Update

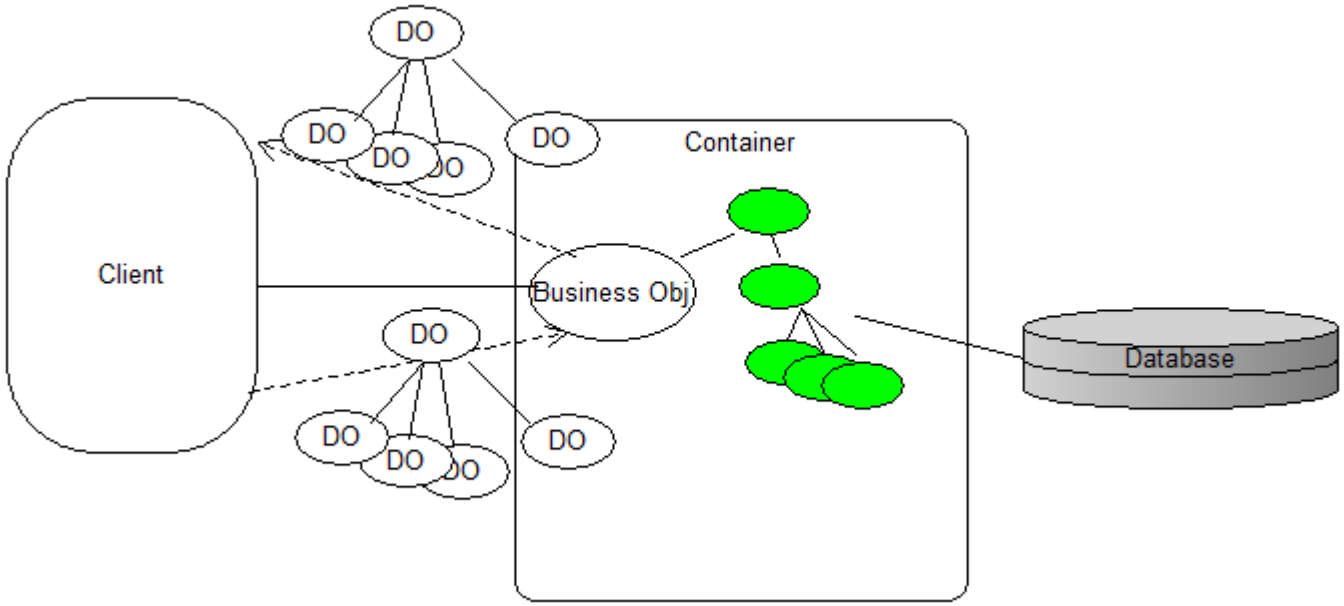
Detaching

Delete

Hibernate

Umgang mit Detached Objects

- Detached Objekte bzw. ganze Teilbäume (vom Persistenzkontext entkoppelt) können über das Netzwerk hin- und her übertragen werden
- Ihre Identität wird beibehalten
- Änderungen kann man wieder persistent machen



Beispiel Detached Object


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

Spieler spieler = em.find(Spieler.class, id);

em.close();

String d = spieler.getPosition();     // lesbar
spieler.setPosition("Mittelfeld");     // änderbar
```

Speichern von detachten Objekten

- **merge(object)**
  - Finde im Persistenzkontext ein Objekt mit gleicher id
  - Falls es existiert, ändere es und gebe Referenz auf dieses zurück
  - Falls es nicht existiert, erzeuge ein neues Objekt im Persistenzkontext, übernehme die Änderungen und gebe dieses zurück
  - Achtung:
    - **object** bleibt weiterhin detached
    - mit der Rückgabe von merge(..) arbeiten

Beispiel detached Objects


```
// Finden
EntityManager em1 = emf.createEntityManager() ;
Spieler spieler =  (Spieler)em1.get(Spieler.class, id) ;
em1.close();

// Ändern
spieler.setPosition("Torwart");

// Speichern
EntityManager em2 = emf.createEntityManager() ;
em2.getTransaction().begin();
em2.merge(spieler);
em2.getTransaction().commit();
```

Agenda

CRUD

Create

Read

Update

Detaching

Delete

Hibernate

Löschen von Objekten

- Löschen funktioniert nicht mit einem detached Objekt

```
EntityManager em = emf.createEntityManager() ;
em.getTransaction().begin();
Spieler spieler = em.find(Spieler.class, id)
em.remove(spieler);
em.getTransaction().commit();
em.close();
```

Aufgabe

- Aufgabe 1:
  - Klasse <Entity> erstellen/ergänzen
  - Mapping erstellen/ergänzen
  - <Entity> persistieren
  - <Entity> finden
  - <Entity> ändern
  - Detaching
  - <Entity> löschen

Agenda

CRUD

Create

Read

Update

Detaching

Delete

Hibernate

Java Klasse - POJO

```
public class Employee {
    private Long id;
    private String firstName;
    private Set addresses = new HashSet();

    public Employee()   {}
    public Employee(String firstName,String lastName…{…}

    public Long getId() {return id; }
    private void setId (Long id)   {this.id = id; }

    ... //Getter,  Setter und Business Methoden
}
```

Eigenschaften von persistenten Klassen

- Muss einen public-Konstruktor ohne Parameter haben
- Kann von anderen Klassen erben
- Kann abstrakt sein
- Referenzierte Klassen müssen auch Entities oder serialisierbar sein

Mapping - Employee.hbm.xml

```
<?xml version="1.0"?>
<hibernate-mapping>
     <class name="de.example.Employee " table="MITARBEITER">
     <id name="id" column="MITARBEITER _ID" type="long">
        <generator class="increment"/>
     </id>
     <property name="firstName" column="VORNAME"/>
     <property name="lastName" column="NACHNAME"/>
     <property name="email" column="EMAIL" type="string"/>
     <set name="addresses">
           <key column="user_id"/>
           <one-to-many class="de.example.Address"/>
     </set>
     </class>
</hibernate-mapping>
```

Objekt erzeugen und in DB einfügen (Hibernate)


```
// Konfiguration
Configuration cfg = new Configuration();
SessionFactory factory = cfg.configure ( )
.buildSessionFactory();

// Verbindungsaufbau
Session session = factory.openSession() ;
Transaction tx = session.beginTransaction();

// Business Logik
Employee u - new Employee("Peter", "Pan",
   "president@neverland.nl");
session.save (u) ;
tx.commit ( ) ;
session.close();
```

get() / load()

- **get(Class clazz, Serializable id)**
  - führt direkt zu einem Datenbankzugriff
  - Gibt null zurück wenn Instanz nicht vorhanden ist
  - SELECT & UPDATE wenn Attribute geändert werden
  - user u = (User) session.**get**(User.class, 1L);

- **load(Class clazz, Serializable id)**
  - führt direkt zu einem Datenbankzugriff
  - Wirft HibernateException wenn Instanz nicht vorhanden ist
  - Kann Proxy zurück geben (default ab Hibernate 3)
    - Interessant wenn man keine getter-Methoden, sondern nur setter()-Methoden aufruft. Proxy enthält nur ID.

Beispiel Update nach get()


```
Session session =
sessionFactory.openSession();
Transaction tx = session.beginTransaction();
Employee u =
  (Employee)session.get(Employee.class, 1L);
u.setLastName("Maier");
tx.commit();
session.close();
```

Detached Objects updaten


```
Session session1 = sf.openSession();
Employee employee = (Employee)
   session1.get(Employee.class, employeeId);
session1.close();
employee.setFirstname("Hans") ;
Session session2 = sf.openSession();
Transaction tx2 = session2.beginTransaction();
session2.update(employee);
tx2.commit();
```

Alternativen (1)

session.save(entity)

- geht davon aus, dass das Geschäftsobjekt neu in die Datenbank geschrieben werden soll.
- kann außerhalb einer Transaktion ausgeführt werden
  - In einer Assoziation wird nur master persistiert.
  - Andere Seite erst beim flush() oder commit()
- Ruft man zweimal save für dasselbe Geschäftsobjekt auf:
  - Schreibt in die DB, wenn Hibernate die IDs selbst vergibt
  - Exception, wenn die ID von der Anwendung vergeben wird

Alternativen (2)

session.persist(entity)

- wie save() aber void Rückgabetyp
- muss transaktional ausgeführt werden

session.update(entity)

- wenn man ein existierendes Objekt in der Datenbank aktualisieren möchte
- muss transaktional ausgeführt werden
- Objekt wird dem Persistenzkontext hinzugefügt und DB-Änderungen werden implizit bei Transaktionscommit ausgeführt

Alternativen (3)

session.merge(entity)

- übergebene **entity** wird nicht persistent, aber Rückgabewert ist persistent
- muss transaktional ausgeführt werden
- Zustand von **entity** wird in persistente Instanz kopiert
  - ggf. wird persistente Instanz vorher geladen bzw. erstellt

session.saveOrUpdate(entity)

- im Gegensatz zu save/update egal ob
  - Speichern einer neuen Instanz
  - oder Update einer abgehängten Instanz
- kann außerhalb einer Transaktion ausgeführt werden
  - In einer Assoziation wird nur master persistiert
  - Andere Seite erst beim flush() oder commit()

DB-Eintrag löschen

- **session.delete(Object o)**
  - Nach Löschen ist Objekt wieder transient
  - Parameter o kann auch ein detached Objekt sein

```
User u =  (User) session....
Session session = sessionFactory.openSession
Transaction tx = session.beginTransaction();
session.delete(u);
tx.commit();
session.close() ;
```

Aufgabe

- Aufgabe 2 Hibernate nativ (optional):
  - Klasse <Entity> erstellen/ergänzen
  - Mapping erstellen/ergänzen
  - <Entity> persistieren
  - <Entity> finden
  - <Entity> ändern
  - Detaching
  - <Entity> löschen