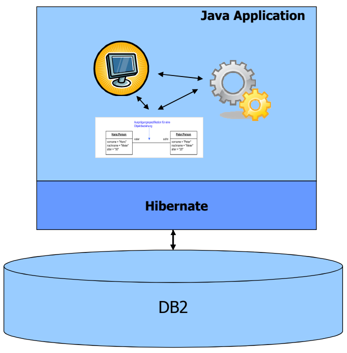
Datenbanken und Informationssysteme

O/R-Mapper: Hibernate

Aufgabe 3: Überblick



Beispiel: Dozent.java

```
public class Dozent {
    private int id;
    private String name;
    private Set vorlesungen = new HashSet();

    public Dozent() {}

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

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public Set getVorlesungen() { return vorlesungen; }
    public void setVorlesungen(Set vorlesungen) {
        this.vorlesungen = vorlsungen; }

    public boolean equals(Object o) {…}
    public int hashCode() {…}
}
```

Eindeutige ID

Default-Konstruktor

Getter/Setter für alle persitenten Attribute

Equals/Hashcode

Beispiel: Dozent.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="pkg.Dozent" table="DOZENTEN">
        <id name="id" column="DOZENT_ID">
            <generator class="native"/>
        </id>
        <property name="name" type="string" column="NAME"/>

        <set name="vorlesungen" inverse="true">
             <key column="DOZENT"/>
             <one-to-many class="pkg.Vorlesung"/>
         </set>
    </class>
</hibernate-mapping>
```

Beispiel: Verwendung

```
private static final SessionFactory sessionFactory;
static {
    sessionFactory = new Configuration().configure().buildSessionFactory();
}

public Long erzeugeDozent(String name) {
    Session session = sessionFactory.getCurrentSession();
    session.beginTransaction();
    Dozent dozent = new Dozent();
    Dozent.setName(name);
    Long did = session.save(dozent);
    session.getTransaction().commit();
    return did;
}

public Dozent ladeDozent(Long dozentId) {
    Session session = sessionFactory.getCurrentSession();
    session.beginTransaction();
    Dozent dozent = (Dozent) session.get(Dozent.class, dozentId);
    session.getTransaction().commit();
    return dozent;
}
```

Konfiguration: hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.ibm.db2.jcc.DB2Driver</property>
        <property name="connection.url">jdbc:db2://vsisls4:50001/VSISP</property>
        <property name="connection.username">vsispXX</property>
        <property name="connection.password">********</property>

        <property name="dialect">org.hibernate.dialect.DB2Dialect</property>
        <property name="current_session_context_class">thread</property>
        <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="pkg/Dozent.hbm.xml"/>
        <mapping resource="pkg/Vorlesung.hbm.xml"/>

    </session-factory>
</hibernate-configuration>
```

Mapping: Bidirektionale Beziehungen

- many-to-one / one-to-many

```
<class name="pkg.Dozent" table="DOZENTEN">
    <id name="id" column="DOZENT_ID">
        <generator class="native"/>
    </id>
    <property name="name" type="string" column="NAME"/>
    <many-to-one name="fakultät" class="pkg.Fakultät"
        column="FAKULTÄT_ID" not-null="true" />
</class>
```

```
<class name=“pkg.Fakultät">
    <id name="id" column="FAKULTÄT_ID">
        <generator class="native"/>
    </id>
    <set name= "dozenten" inverse="true">
        <key column="DOZENT_ID"/>
        <one-to-many class=“pkg.Dozent"/>
    </set>
</class>
```

Mapping: Bidirektionale Beziehungen

- many-to-many

```
<class name="pkg.Dozent" table="DOZENTEN">
    <id name="id" column="DOZENT_ID">
        <generator class="native"/>
    </id>
    <property name="name" type="string" column="NAME"/>
    <set name="studenten" table="DOZ_STUD" >
        <key column="DOZ_ID"/>
        <many-to-many column="STUD_ID" class="pkg.Student"/>
    </set>
</class>
```

```
<class name="pkg.Student" table="STUDENTEN">
    <id name="id" column="STUDENT_ID">
        <generator class="native"/>
    </id>
    <set name="dozenten" table="DOZ_STUD" >
        <key column="STUD_ID"/>
        <many-to-many column="DOZ_ID" class="pkg.Dozent"/>
    </set>
</class>
```

Unidirektionale Beziehungen

- many-to-one

```
<class name="pkg.Dozent" table="DOZENTEN">
    <id name="id" column="DOZENT_ID">
        <generator class="native"/>
    </id>
    <property name="name" type="string" column="NAME"/>

    <many-to-one name="fakultät" class="pkg.Fakultät"
        column="FAKULTÄT_ID" not-null="true" />
</class>
```

```
<class name=“pkg.Fakultät">
    <id name="id" column="FAKULTÄT_ID">
        <generator class="native"/>
    </id>
</class>
```

Mapping: Vererbung

- table-per-subclass (Partitionierungsmodell)

```
<class name=“UniMitarbeiter" table=“MITARBEITER“ abstract=“true”>
    <id name="id" type="long" column=“MID">
        <generator class="native"/>
    </id>
    <property name=“name" column=“NAME"/>
    ...
    <joined-subclass name=“Professor" table=“PROFESSOR">
        <key column=“MID"/>
        <property name=“ProfessurTyp" column=“PROF_TYP"/>
        ...
    </joined-subclass>
    <joined-subclass name=“Wissenschaftler" table=“WISSENSCHAFTLER">
        <key column="MID"/>
        ...
    </joined-subclass>
    <joined-subclass name=“Techniker" table=“TECHNIKER">
        <key column=“MID"/>
        ...
    </joined-subclass>
</class>
```

→ 4 Tabellen

Mapping: Vererbung

- table-per-concrete-class (Hausklassenmodell)

```
<class name=“UniMitarbeiter“ abstract=“true”>
    <id name="id" type="long" column="PAYMENT_ID">
        <generator class=“sequence"/>
    </id>
    <property name="amount" column="AMOUNT"/>
    ...
    <union-subclass name=“Professor" table=“PROFESSOR">
        <property name=“ProfessurTyp" column=“PROF_TYP"/>
        ...
    </union-subclass>
    <union-subclass name=“Wissenschaftler" table=“WISSENSCHAFTLER">
        ...
    </union-subclass>
    <union-subclass name=“Techniker" table=“TECHNIKER">
        ...
    </union-subclass>
</class>
```

→ 3 Tabellen

Mapping: Vererbung

- table-per-class-hierarchy

```
<class name="UniMitarbeiter" table="MITARBEITER“ abstract=“true”>
    <id name="id" type="long" column="MID">
        <generator class="native"/>
    </id>
    <discriminator column="MITARBEITER_TYP" type="string"/>
    <property name="name" column="NAME"/>
    ...
    <subclass name="Professor" discriminator-value="PROF">
        <property name="ProfessurTyp" column="PROF_TYP"/>
        ...
    </subclass>
    <subclass name="Wissenschfter" discriminator-value="WIMI">
        ...
    </subclass>
    <subclass name="Techniker" discriminator-value="TECH">
        ...
    </subclass>
</class>
```

→ 1 Tabelle

Objekte

- Objektzustände

	Transient	Persistent	Detached
Session-Zuordnung	✗	✓	✗
DB-Repräsentation	✗	✓	✓

- Objekte speichern

```
public Long erzeugeDozent(String name) {
    Dozent dozent = new Dozent(); //transientes Objekt dozent
    dozent.setName(name);
    Long did = session.save(dozent); //persistentes Objekt dozent
    return did;
}
```

Objekte

- Objekte laden

```
Dozent dozent = (Dozent) session.get(Dozent.class, did);
```

- Existenz prüfen

```
Dozent dozent = (Dozent) session.get(Dozent.class, did);
if (dozent==null) {
    dozent = new Dozent();
    session.save(dozent, did);
}
```

- Objekte aktualisieren

```
session.save(dozent);
session.flush(); //SQL INSERT erzwingen
session.refresh(dozent); /* Objekt enthält Werte, die z.B. durch einen
                            Trigger bestimmt wurden */
```

Objekte

- Objekte (persistent) ändern

```
Dozent dozent = (Dozent) session.get(Dozent.class, did);
dozent.setName("Müller");
session.flush();
```

- Objekte (detached) ändern

```
Session session1 = sessionFactory.getCurrentSession();
session1.beginTransaction();
Dozent dozent = (Dozent) session1.get(Dozent.class, did);
Dozent.setName(name);
session1.getTransaction().commit();
session1.close();

[…]
dozent.setFakultät(fak);          //dozent ist im Zustand "detached"

session session2 = sessionFactory.getCurrentSession();
session2.beginTransaction();
session2.update(dozent);
```

Objekte

- Objekte löschen

```
session.delete(dozent);
```

- Objekte suchen (HSQL)

```
List dozenten = session.createQuery(
    "from Dozent as dozent where dozent.alter < ?")
    .setInteger(0, 40)
    .list();

List kittens = session.createQuery(
    "from Dozent as dozent where dozent.fakultät = ?")
    .setEntity(0, fak)
    .list();

Fakultät fak = (Fakultät) session.createQuery(
    "select dozent.fakultät from Dozent as dozent where dozent = ?")
    .setEntity(0, doz)
    .uniqueResult();
```

DB-Verbindung/Transaktionen

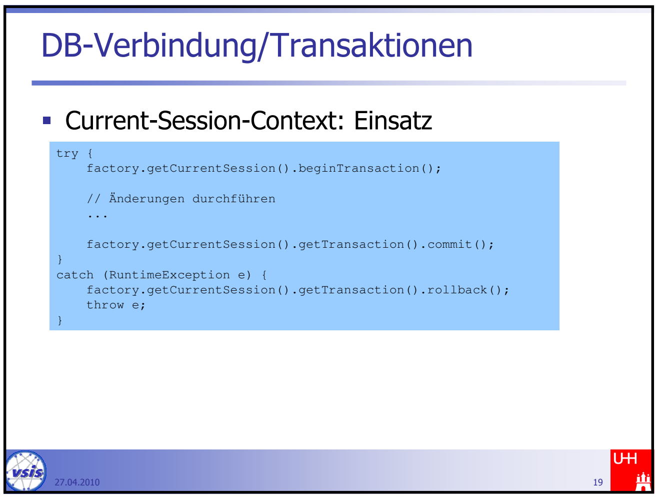
- SessionFactory
  - Verwaltet Konfigurationseinstellungen
  - Verwaltet Abbildungsregeln
  - Delegiert Connection-Verwaltung (ConnectionProvider)
  - Erzeugt Sessions
- Session
  - Kurzlebig (Empfehlung: Session pro Transaktion)
  - Erzeugt Transaktion(en)
  - Cache für persistente Objekte
  - Gebunden an Kontext (Thread)
- Transaction
  - Atomare "Units of Work"
  - Explizite Steuerung (Begin/Commit/Rollback) notwendig

DB-Verbindung/Transaktionen

- Current-Session-Context

```
<property name="current_session_context_class">thread</property>
```

  - Bindet Session an aktuellen Thread
  - Erzeugt Session bei erstem Aufruf von `getCurrentSession()`
  - Nachfolgende Aufrufe von `getCurrentSession()` im aktuellen Thread liefern gleiche Session
  - Löst Session vom Thread und schließt Session bei Transaktionsende
  - Nachfolgende Aufrufe von `getCurrentSession()` im aktuellen Thread liefern neue Session



DB-Verbindung/Transaktionen

- Current-Session-Context: Einsatz

```
try {
    factory.getCurrentSession().beginTransaction();

    // Änderungen durchführen
    ...

    factory.getCurrentSession().getTransaction().commit();
}
catch (RuntimeException e) {
    factory.getCurrentSession().getTransaction().rollback();
    throw e;
}
```