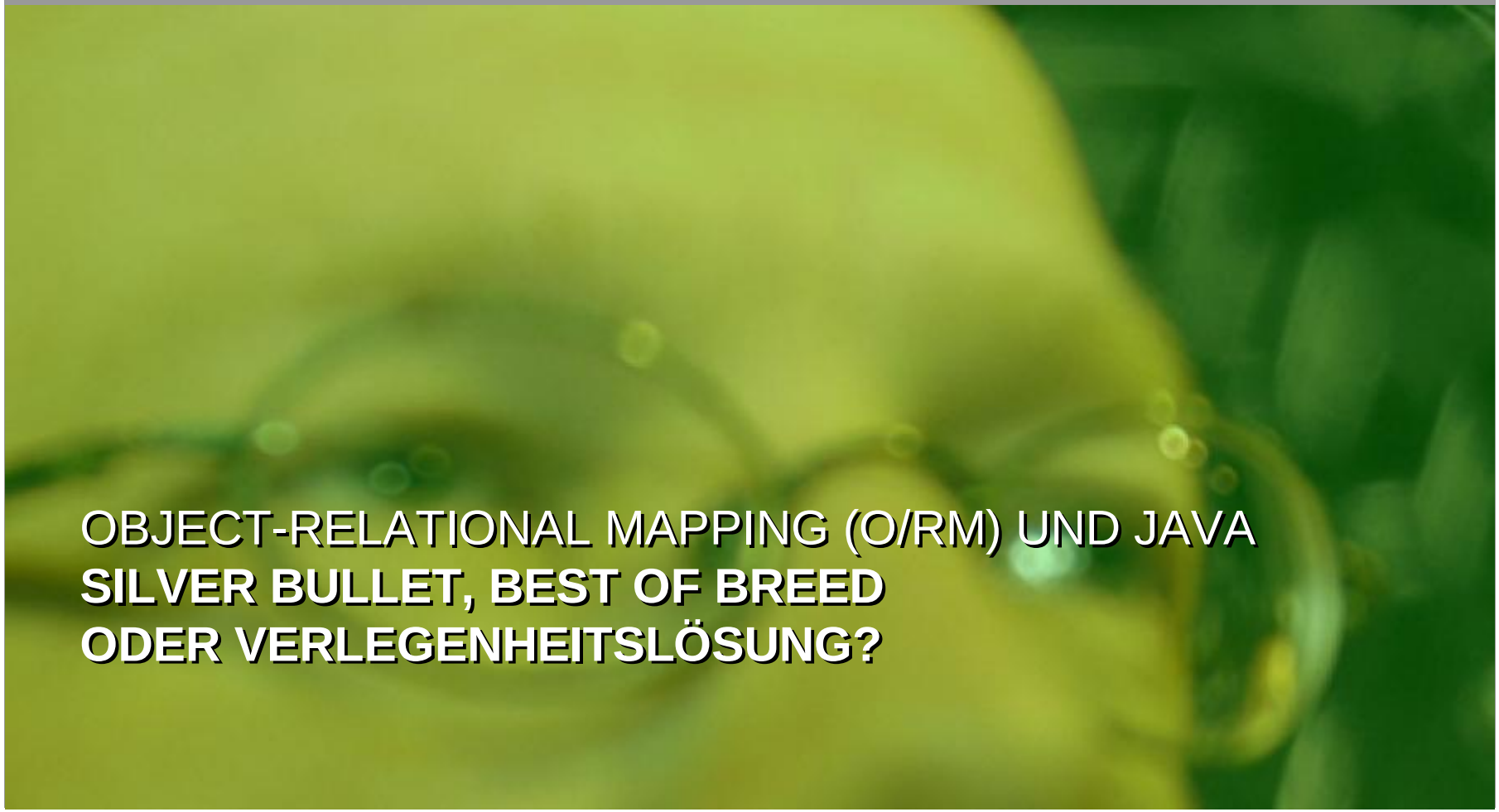
netcetera

TEKZONEFORUM 063

MO. 28. Aug. 2006, 17:00 UHR

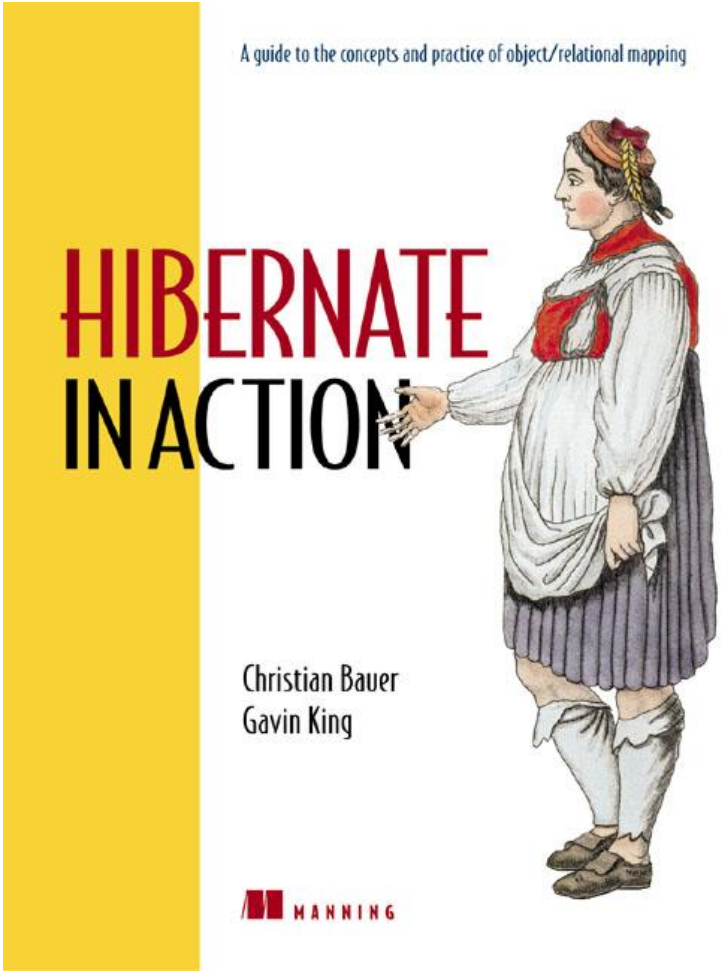


# OBJECT-RELATIONAL MAPPING (O/RM) UND JAVA
# SILVER BULLET, BEST OF BREED ODER VERLEGENHEITSLÖSUNG?

HIBERNATE

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften




# Motivation

# Konzept

# Persistenz

# Motivation



- Hibernate ist ein Open-Source-Persistenz-Framework für Java, das es ermöglicht, den Zustand eines Objekts in einer relationalen Datenbank zu speichern und umgekehrt, aus Datensätzen in einer relationalen Datenbank Objekte zu erzeugen. Dies bezeichnet man auch als O/R-Mapping-Tool (Object-Relational-Mapping-Tool).
- Nach Ansicht des Sprechers besteht der Server-Code einer Anwendung zu mehr als 50% aus JDBC Code!?

HIBERNATE

# Motivation

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

- Bei den Objekten handelt es sich um POJO's (Plain Old Java Objects). Diese Objekte sind einfacher Natur und haben Attribute und Methoden (Accessors, Mutators).
- Hibernate kann mit allen relationalen Datenbanksystemen benutzt werden, für die es einen JDBC-Treiber gibt. Anwendungsseitig kann Hibernate entweder in einer Java-Applikation benutzt werden oder in eine Servlet-Engine oder einen Applikationsserver integriert werden.

# Motivation



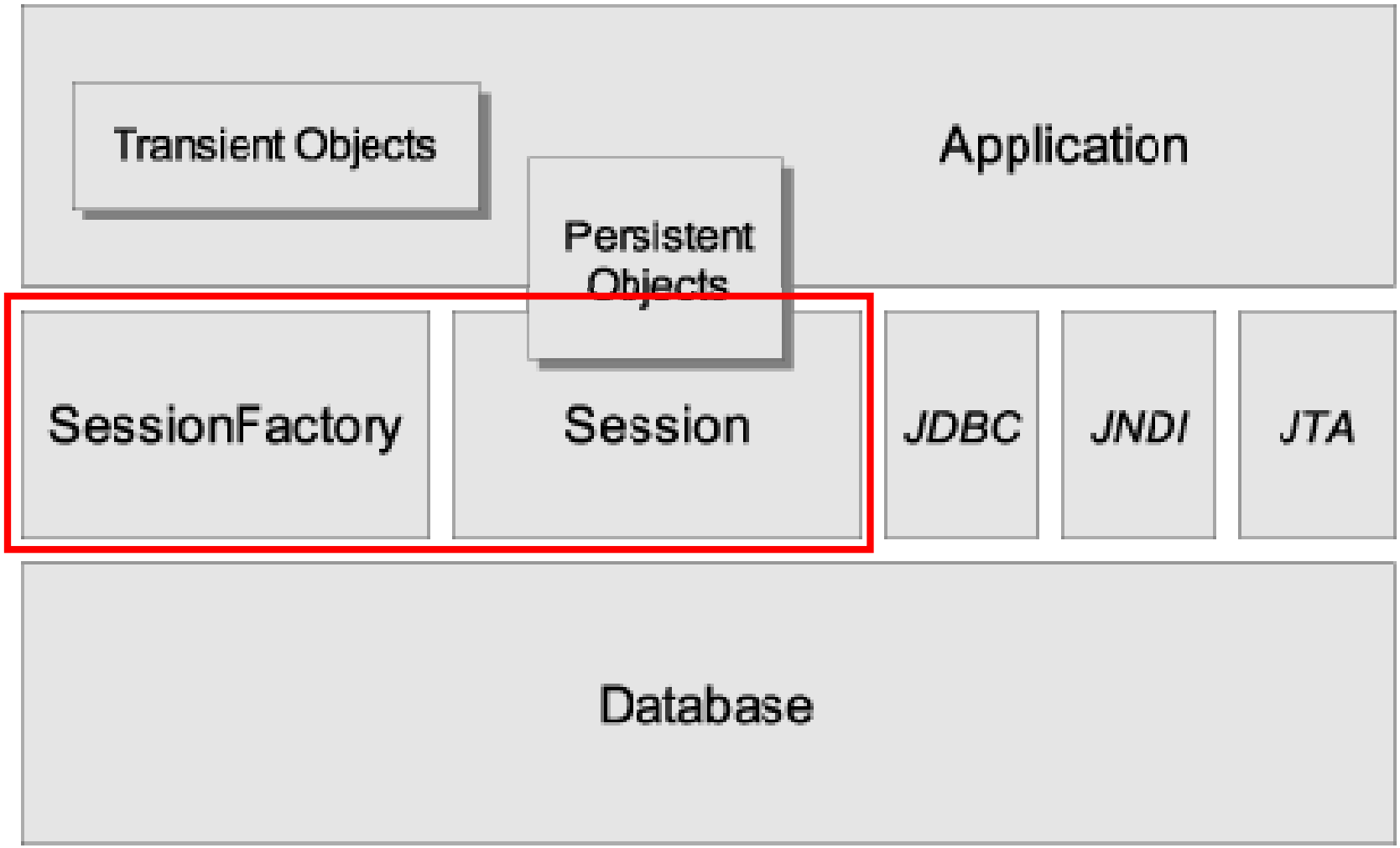
- Mit Hibernate muss kein JDBC geschrieben werden
- Mit Hibernate werden Objekt-Verknüpfungen automatisch verwaltet
- Hibernate trennt die Persistenzschicht sauber von der Anwendungsschicht
- Hibernate steht nicht in Konkurrenz zu Standards wie JDBC, sondern verwendet diese im Hintergrund

# Persistenz

Zürcher Hochschule Winterthur
Departement Technik, Informatik und Naturwissenschaften

- **Automatische Persistenz -> kein Aufwand**
  - In Hibernate wird die Persistenz der Daten automatisch sichergestellt
  - Die Persistenzschicht in Java ist dafür verantwortlich, dass alle Daten mit der Datenbank synchron sind
- **Transparente Persistenz -> keine Aenderung**
  - Bei transparenter Persistenz werden die zu speichernden Java Objekte codemässig nicht verändert

# Architektur



# Architektur



- Session Factory Klasse
  - Einmalig (pro Datenbankverbindung)
  - Bezieht die Informationen aus dem XML File (hibernate.cfg.xml) oder Property File (hibernate.properties)
    - Verwendeter Treiber (JDBC)
    - Pfad zur Datenbank
    - Dialekt der Datenbank
    - Bestehende Mappings in der Datenbank, auf welche aus Java zugegriffen werden kann
    - Weitere Details wie Cache, etc.

# Architektur



- Session
  - Wird von der Session Factory geliefert
  - Mit ihr wird von der Java Anwendung aus auf die Datenbank zugegriffen!!!!!!!
  - Kümmert sich um alles im Bezug auf das Starten und das Beenden von Transaktionen (commit, rollback, etc.)
  - Kennt!!!! und kümmert!! sich um alles im Bezug auf Objekte (speichern, löschen, ändern, etc.)

# Beispiel



- Das folgende Beispiel beginnt mit einer Java Klasse, die einen ‚Event' repräsentiert und geht über die Hibernate Mapping- und Configuration- Dateien bis zum Laden und Speichern der Objekte.
- Entspricht dem Beispiel in Kapitel 2 der Hibernate Dokumentation

# Event.java



- Die erste Klasse repräsentiert einen Event, der in die Datenbank gespeichert wird
- Verwendung von Standard JavaBean Namenskonventionen und somit getter und setter Methoden
- Dies ist die empfohlene Vorgehensweise, Hibernate könnte aber auch direkt auf die Felder zugreifen

# Event.java

```
public class Event {
  private Long id; // Schlüsselattribute
          // Preis!! für die Persistenz
  private String title;
  private Date date;

  Event() {}
  public Long getId() {
    return id;
  }
  private void setId(Long id) {
    this.id = id;
  }
  public Date getDate() {
    return date;
  }
```

# Event.java



```
public void setDate(Date date) {
    this.date = date;
  }
  public String getTitle() {
    return title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
}
```

# Mapping-File



- Es muss Hibernate mitgeteilt werden, wie es die Objekte der persistenten Klasse laden und speichern kann
- Mit dem Mapping-File wird Hibernate mitgeteilt, auf welche Tabellen in der Datenbank es zugreifen soll und welche Attribute zu verwenden sind
- Ab HIBERNATE 3 zusätzlich mit JAVA 1.5 Annotations.

# Mapping-File

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

```
<?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>

  [...]

</hibernate-mapping>
```

# Mapping-File

```
<hibernate-mapping>

  <class name="Event" table="EVENTS">

  </class>

</hibernate-mapping>
```

# Mapping-File

```
<hibernate-mapping>

  <class name="Event" table="EVENTS">
    <id name="id" column="EVENT_ID">
      <generator class="native"/>
    </id>
  </class>

</hibernate-mapping>
```

# Mapping-File

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

```
<hibernate-mapping>

<class name="Event" table="EVENTS">
  <id name="id" column="EVENT_ID">
    <generator class="native"/>
  </id>
  <property name="date" type="timestamp"
    column="EVENT_DATE"/>
  <property name="title"/>
</class>

</hibernate-mapping>
```

# Config-File



- Es kann eine vereinfachte hibernate.properties Datei oder die etwas weiter entwickelte hibernate.cfg.xml Datei verwendet werden
- Hibernate sucht automatisch nach dieser Datei im Hauptklassenpfad

# Config-File

```
<?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>

    <!-- Database connection settings -->
    <property name="connection.driver_class">
      org.hsqldb.jdbcDriver</property>
    <property name="connection.url">
      jdbc:hsqldb:data/tutorial</property>
    <property name="connection.username">sa</property>
    <property name="connection.password"></property>
```

# Config-File

```
    <!-- JDBC connection pool (use the built-in) -->
    <property name="connection.pool_size">1</property>

    <!-- SQL dialect -->
    <property name="dialect">
      org.hibernate.dialect.HSQLDialect</property>

    <!-- Echo all executed SQL to stdout -->
    <property name="show_sql">true</property>

    <!-- Drop and recreate on startup -->
    <property name="hbm2ddl.auto">create</property>

    <mapping resource="Event.hbm.xml"/>

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


# EventManager

Zürcher
Hochschule
Winterthur

Departement Technik,
Informatik und
Naturwissenschaften

- Eine kleine Anwendung (EventManager) zeigt im folgenden, wie man Objekte aus der Datenbank holt bzw. in die Datenbank speichert. (Ausschnitt)

# EventManager

Zürcher Hochschule Winterthur
Departement Technik, Informatik und Naturwissenschaften

```
public void createAndStoreEvent(String title, Date
  theDate) {
  Session session = HibernateUtil.getSession();
  Transaction tx = session.beginTransaction();

  Event theEvent = new Event();
  theEvent.setTitle(title);
  theEvent.setDate(theDate);

  session.save(theEvent);

  tx.commit();
  HibernateUtil.closeSession();
}
```

# EventManager

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

```
private List listEvents() {
  Session session = HibernateUtil.getSession();
  Transaction tx = session.beginTransaction();

  List result =
    session.createQuery("from Event").list();

  tx.commit();
  session.close();

  return result;
}
```

# EventManager

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

```
……..
List events = EventMgr.listEvents();

for (int i = 0; i < events.size(); i++) {
    Event theEvent = (Event) events.get(i);
    System.out.println("Event: " +
                       theEvent.getTitle() +
                       " Time: " +
                       theEvent.getDate());
  }
}
```

# HQL

- HQL ist die ***Abfrage***sprache von Hibernate und bildet ein Analogon zu SQL
- Beispiel aus dem EventManager:

```
List result =
  session.createQuery("from Event").list();
```

- Beispiel aus der Hibernate Dokumentation:
  “`from Cat cat where cat.mate.name is not null`”
  Auf diese Art kann eine Relation direkt abgefragt werden

# Person.java

- Als nächstes geht es nun darum eine zweite Tabelle zu implementieren
- Die Klasse Person.java repräsentiert diese Tabelle
- Parallel muss ein neues Mapping-File (Person.hbm.xml) erstellt werden und im Hibernate Konfigurations-File eingetragen werden

# Person.java

Zürcher Hochschule Winterthur

Departement Technik, Informatik und Naturwissenschaften

```
public class Person {

  private Long id;
  private int age;
  private String firstname;
  private String lastname;

  Person() {}

  // Accessor methods for all properties, private
  // setter for 'id'

}
```

# Mapping

```
<hibernate-mapping>

  <class name="Person" table="PERSON">
    <id name="id" column="PERSON_ID">
      <generator class="increment"/>
    </id>
    <property name="age"/>
    <property name="firstname"/>
    <property name="lastname"/>
  </class>

</hibernate-mapping>
```

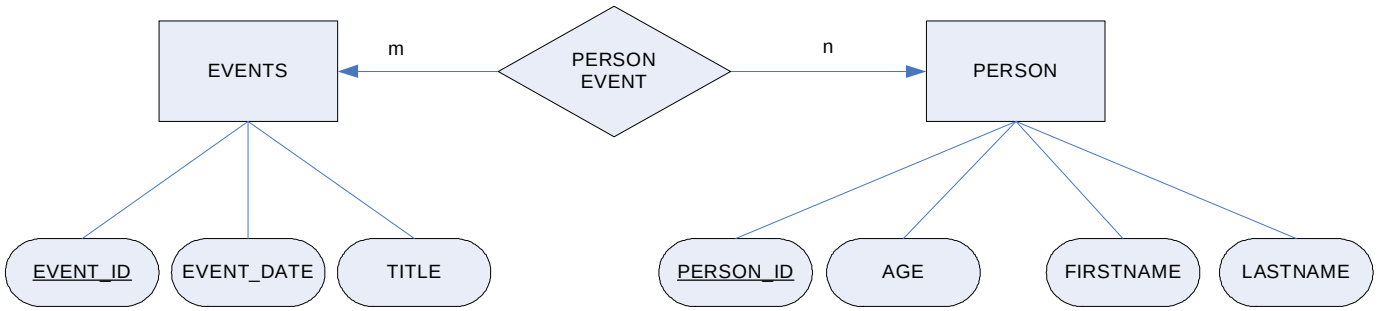

# Config-File


```
<mapping resource="Event.hbm.xml"/>
<mapping resource="Person.hbm.xml"/>
```

# Beziehung

- Im nächsten Schritt wird nun die Beziehung zwischen unseren beiden Tabellen gemäss der folgenden Grafik erstellt
- In diesem Fall ist es eine m:m (many-to-many) Beziehung

# Beziehung


EVENTS
m
PERSON
EVENT
n
PERSON
EVENT_ID
EVENT_DATE
TITLE
PERSON_ID
AGE
FIRSTNAME
LASTNAME

# Beziehung

```
public class Person {
  private Set events = new HashSet();

  public Set getEvents() {
    return events;
  }

  public void setEvents(Set events) {
    this.events = events;
  }
  ….
  ….
}
```

# Beziehung

```
<class name="Person" table="PERSON">

  <id name="id" column="PERSON_ID">
    <generator class="increment"/>
  </id>
  <property name="age"/>
  <property name="firstname"/>
  <property name="lastname"/>

  <set name="events" table="PERSON_EVENT">
    <key column="PERSON_ID"/>
    <many-to-many column="EVENT_ID" class="Event"/>
  </set>

</class>
```

# EventManager

Zürcher Hochschule Winterthur
Departement Technik, Informatik und Naturwissenschaften

```
private void addEventToPerson(Long personId,
  Long eventId) {

  Session session = HibernateUtil.getSession();
  Transaction tx = session.beginTransaction();

  Person aPerson = (Person) session.load(Person.class,
                   personId);
  Event anEvent = (Event) session.load(Event.class,
                     eventId);

  aPerson.getEvents().add(anEvent);

  tx.commit();
  HibernateUtil.closeSession();
}
```

# Bidirektional

Zürcher Hochschule Winterthur
Departement Technik, Informatik und Naturwissenschaften

- Mit den folgenden Ergänzungen wird eine bidirektionale Assoziationen realisiert.
- Wichtig ist dabei die Sicht aus Java und nicht aus SQL!!!!!
- Somit muss die Bidirektionalität in Java selber geschrieben werden!!!
- Beziehungen in relationalen Datenbanken sind immer **bidirektional** aufgrund von Schlüssel und Fremdschlüssel!!!

# Bidirektional

Zürcher Hochschule Winterthur
Departement Technik, Informatik und Naturwissenschaften

```
public class Event {
…….

private Set participants = new HashSet();

public Set getParticipants() {
  return participants;
}

public void setParticipants(Set participants) {
  this.participants = participants;
}
```

# Bidirektional

```
public class Person {
……..
protected Set getEvents() {
  return events;
}

protected void setEvents(Set events) {
  this.events = events;
}

public void addToEvent(Event event) {
  this.getEvents().add(event);
  event.getParticipants().add(this);
}

public void removeFromEvent(Event event) {
  this.getEvents().remove(event);
  event.getParticipants().remove(this);
}
```

# Bidirektional

Erweiterung im XML File:

```
<class name="Event" table="EVENTS">
    <set name="participants" table="PERSON_EVENT"
      inverse="true">
      <key column="EVENT_ID"/>
      <many-to-many column="PERSON_ID" class="Person"/>
    </set>
    ……
```


Zürcher
Hochschule
Winterthur

Departement Technik,
Informatik und
Naturwissenschaften

# Beziehungstypen in Hibernate

# I. Vererbung in Hibernate

## Modellierungsvarianten

***Table per concrete class***--Discard polymorphism and inheritance relationships completely from the relational model

***Table per class hierarchy***—Enable polymorphism by denormalizing the relational model and using a type discriminator column to hold type information

***Table per subclass***—Represent “is a” (inheritance) relationships as “has a” (foreign key) relationships

# II. Relationships in Hibernate

## Richtung von Beziehungen

In relationalen Datenbanken sind Relationen immer bidirektional (über Schlüssel und Fremdschlüssel).

In Java (und somit in Hibernate) müssen diese explizit programmiert und konfiguriert werden.

Java (und somit Hibernate) erlaubt ***unidirektionale*** Relationen. (Obwohl diese in einer relationalen Datenbank immer bidirektional sind)

# II. Relationships in Hibernate Multiplizität

Neben der Richtung muss auch die Multiplizität einer Beziehung in Java und somit in Hibernate programmiert und konfiguriert werden.
Zwischen Instanzen der Klassen A und B unterscheiden wir zwischen folgenden Typen der Multiplizität:

# III. Relationships im Ueberblick

- one to one, unidirektional
- one to one, bidirektional
- one to many, unidirektional
- many to one, unidirektional
- one to many, bidirektional
- many to one, bidirektional
- many to many, unidirektional
- many to many, bidirektional

# Entity-Objects und Value-Objects

Entity-Objects Klassen:

- eigene Identität
- Primärschlüssel in der Datenbank
- Beispiele: Person, Termin, Auftrag..

Value-Objects Klassen:

- Value-Objects Klassen sind Java Klassen.
- Werden in der Datenbank gespeichert.
- Haben keine Identität, nur ihre Attributwerte spielen eine Rolle.
- In UML 2 sind dies <<datatype>>
- Diese Werte Klassen können singulär oder auch in Collections auftreten.
- Beispiele: Adresse (Strasse, Nr, PLZ, ORT), EmailAdresse,
  Note (Fach, Notenwert),....

# Weiter Basiseigenschaften von HIBERNATE

- Abbildungkonstrukte für jegliche Art von Relationships
- Unterstützung von optimistischem Locking
- steuerbare, transitive Persistenz (by reachability)
- Klassenabhängige Fetching Strategie (immediate, **lazy,** eager, batch)
- Unterstützung von normalen und verteilten Transaktionen (Java Transaction API)
- First- und Second Level Caching
- Mächtiges HQL
- (Reverse) Engineering Tools
- …und…. und…

# Hibernate 3.0

- Hibernate 3.0 bildet den Persistenz-Kern von EJB 3.0
- Begriffe sind z. T. verschieden (Session = Entity Manager)
- Konfiguration by exception und Namenskonvention
- Java 1.5 Annotations zusätzlich zu  XML File Mapping
- NHIBERNATE für .NET

# Hibernate an der ZHW

- Hibernate wir im Pflicht-Modul DBP behandelt
- Entwicklung eines Eclipse basierendem Modellierungs- und Code Generierungs-Tools in Zusammenarbeit mit Kunden.
- Anwendungen(Projekt- und Diplomarbeiten) mit relationalen Datenbanken immer mit HIBERNATE
- Erste Messungen mit HIBERNATE Caches und NHIBERNATE.


Zürcher
Hochschule
Winterthur

Departement Technik,
Informatik und
Naturwissenschaften

# These für Diskussion

**JAVA + relationale Datenbanken => benutze  HIBERNATE**