Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Hibernate – eine Einführung

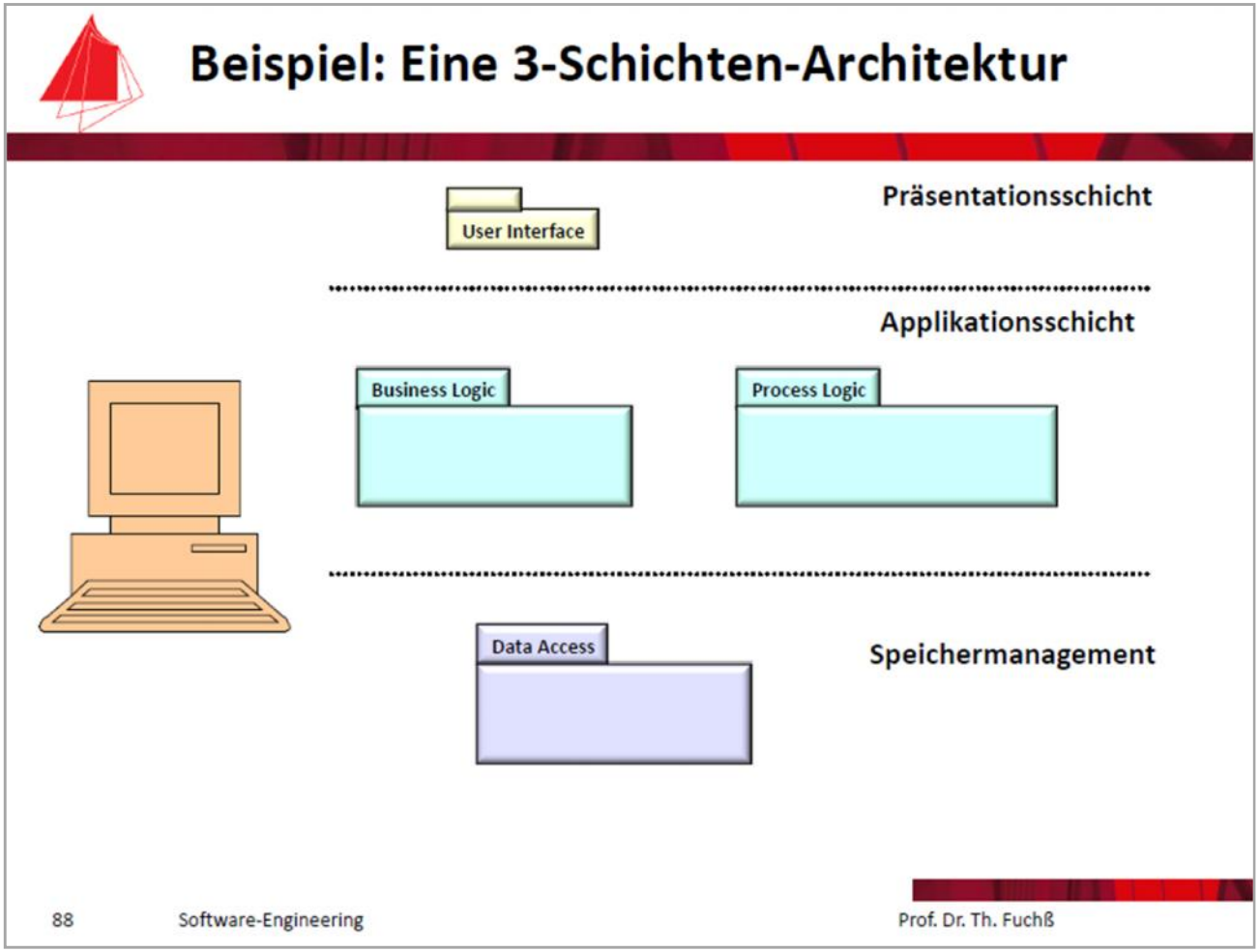
Labor Verteilte Systeme
Fachgebiet Informatik
Hochschule Karlsruhe

Dipl.-Inform.(FH) Adelheid Knodel

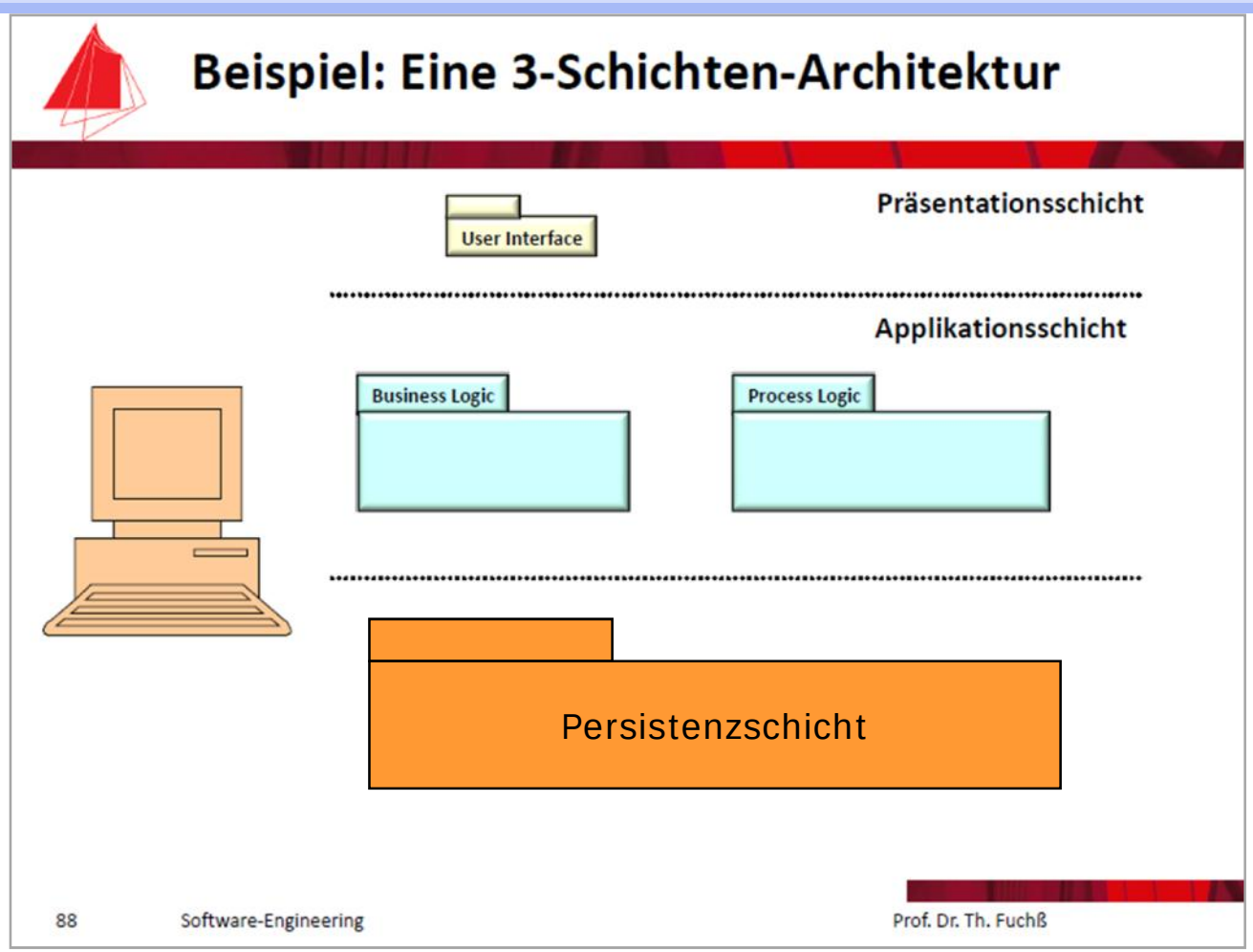
Inhalt

- Einleitung
- Architektur von Hibernate
- Connection Management
- Transaction Management
- Object relational mapping
- Objekte bearbeiten
- Bezug zu Aufgabe Teil 1

Einleitung 3-Schichten-Architektur



Einleitung - 3 Schichten Architektur



Einleitung

Java-Objekte — Relationale Datenbank

Objektorientierte Programmiersprache <-> relationale Datenbanksprache

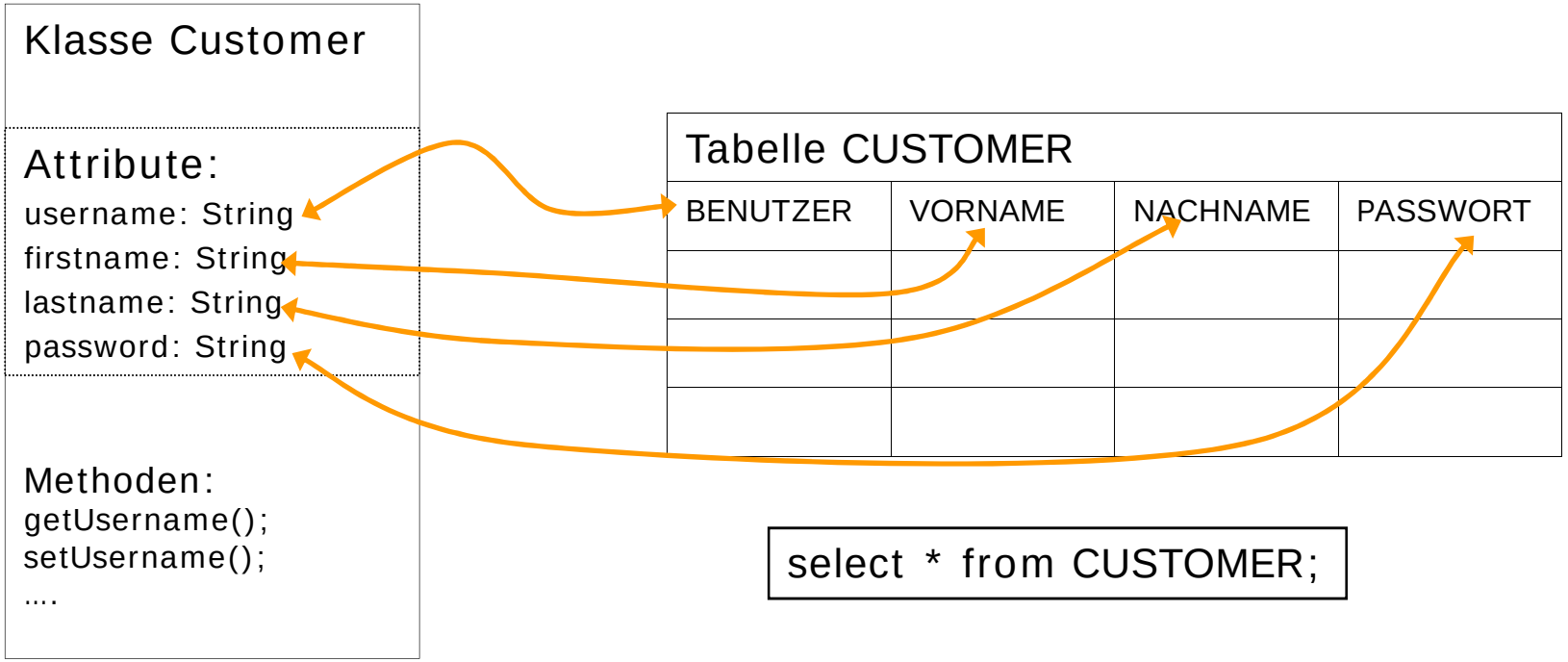
Umsetzung von

Objekten	<-> Tabellen
Vererbungsbeziehungen	<-> Tabellen
Beziehungen zwischen Objekten	<-> Fremdschlüsselbeziehungen
Java Datentypen	<-> SQL Datentypen
eindeutige Identität eines Objekts	<-> Primärschlüssel

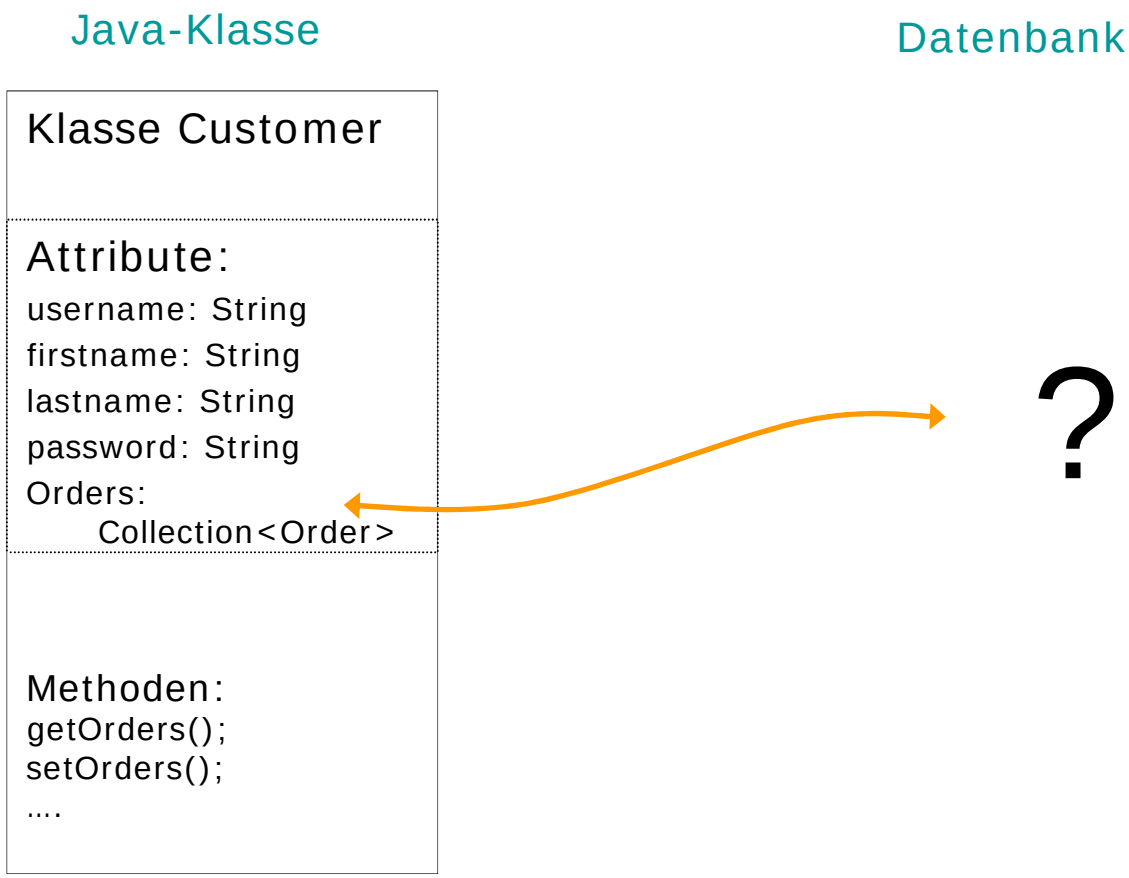
Einleitung - Beispiel 1

Java-Klasse

Datenbank Tabelle



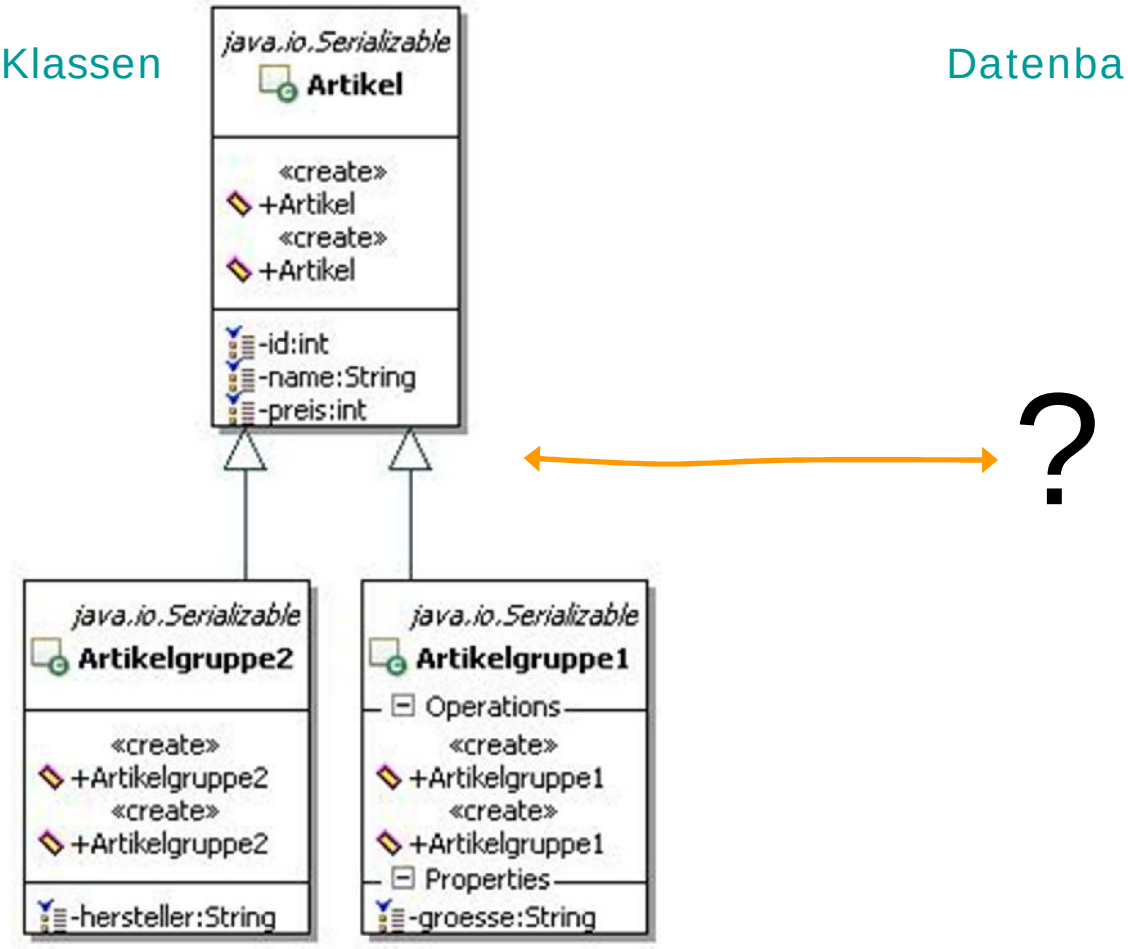
Einleitung - Beispiel 2



Einleitung - Beispiel 3

Java-Klassen

Datenbank Tabelle

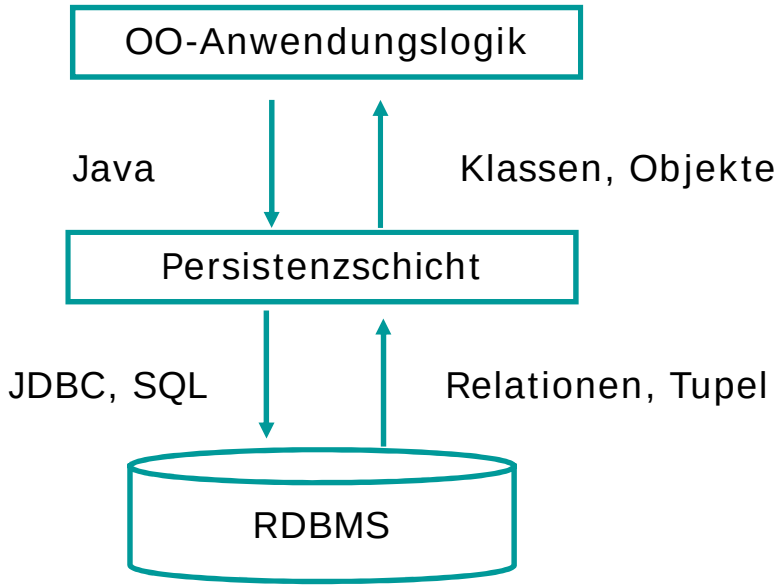


Einleitung - Datenbankabfragen bisher

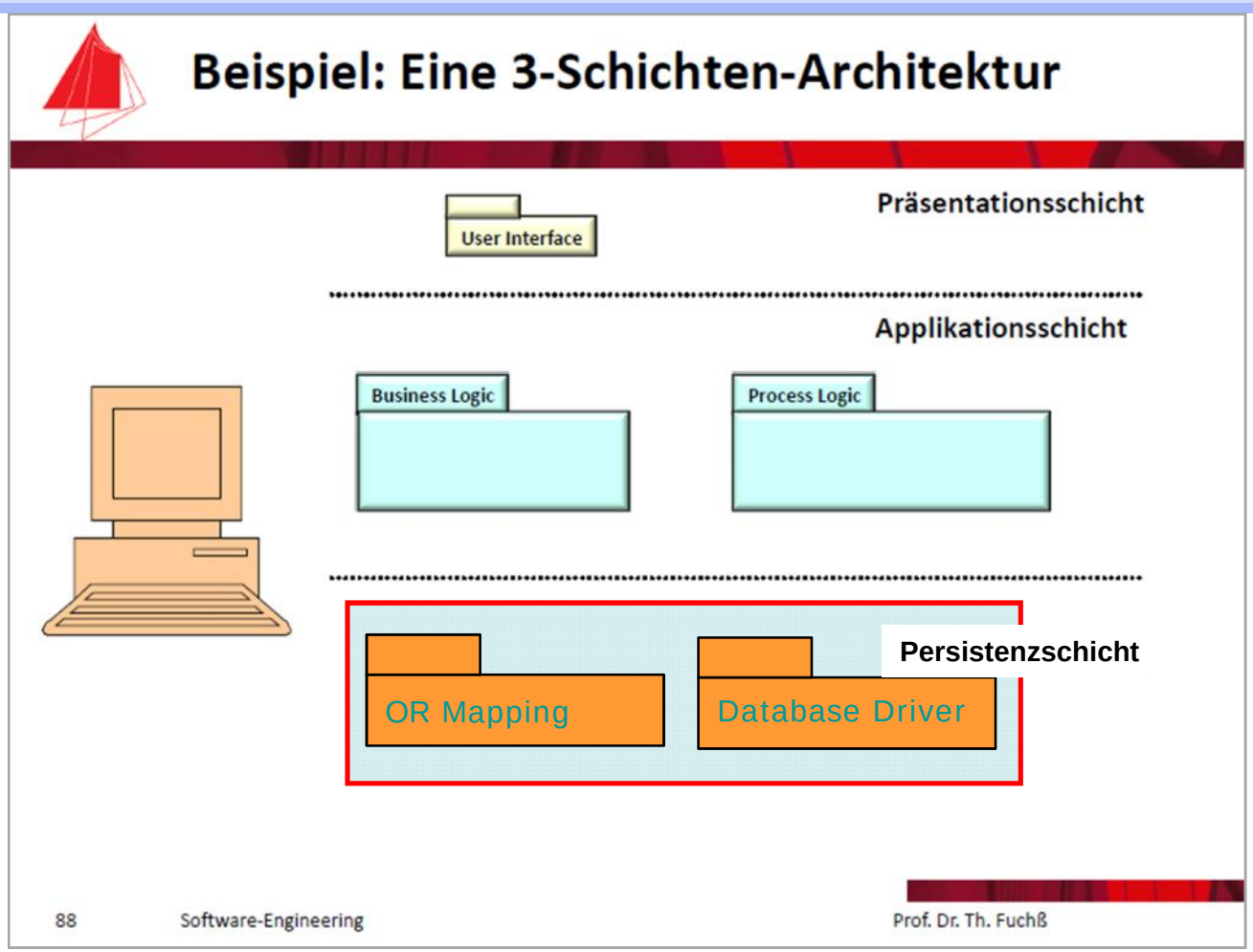
```
// Connect to the database at that URL.
Connection con =
         DriverManager.getConnection(url, props);

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(
         "SELECT sno, status, city FROM Suppliers");

// Step through the result rows.
System.out.println("Got results for " + url);

while (rs.next()) {
     // get the values from the current row:
     String sno = rs.getString(1);
     int status = rs.getInt(2);
     String city = rs.getString("city");
     // Now print out the results:
     System.out.print(" sno=" + sno);
     System.out.print(" status=" + status);
     System.out.print(" city=" + city);
     System.out.print("\n");
}
```

Was ist Hibernate?

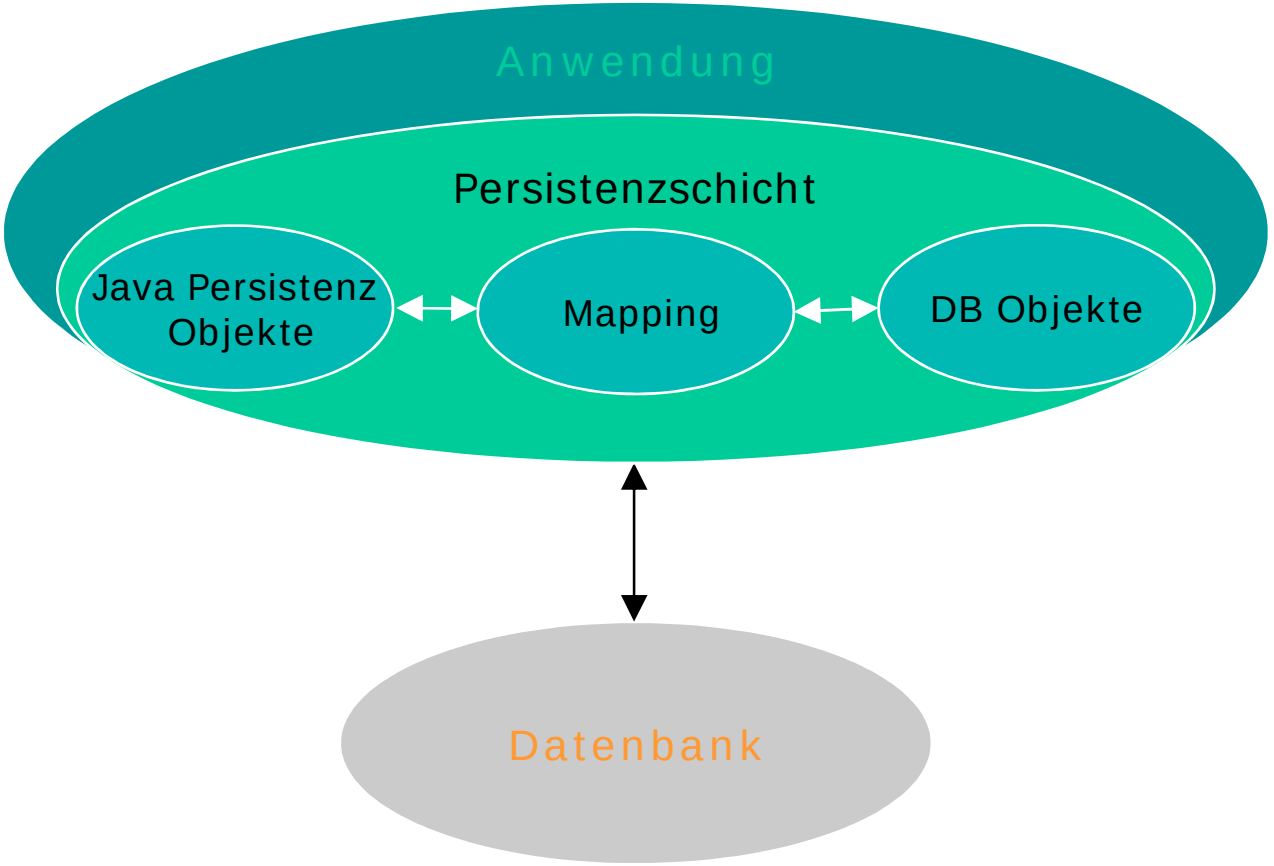
- Hibernate ist ein O/R Persistenz-Framework
- Verbindung zwischen Objektmodell und relationalem Modell



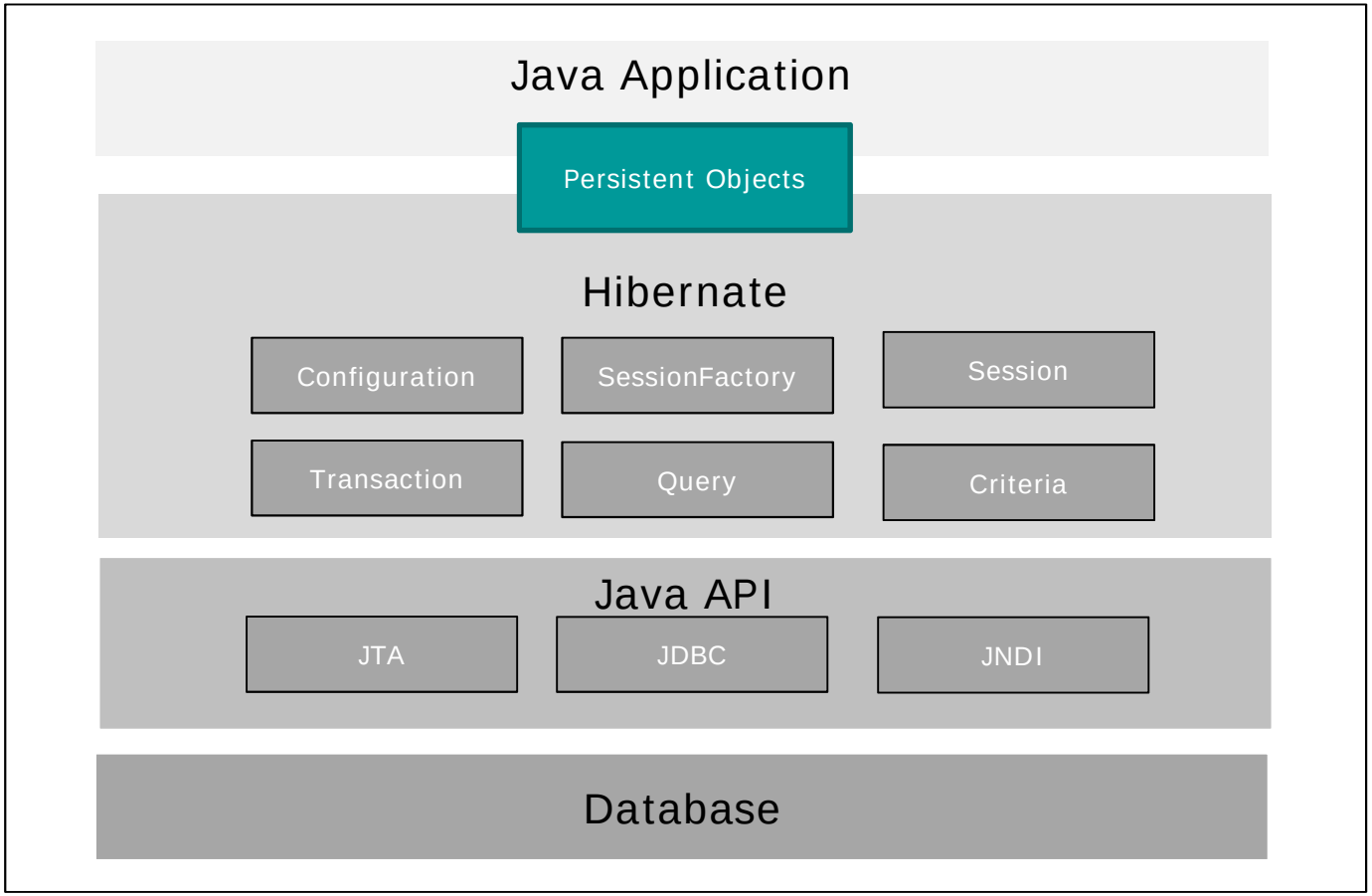
Was ist Hibernate? - Zwischenschicht OR-Mapping



Was ist Hibernate? - Persistenzschicht



Architektur von Hibernate



http://www.tutorialspoint.com/hibernate/hibernate_architecture.htm

Architektur von Hibernate - Komponenten

wichtig für Anwendungsentwickler

- Connection management
  - Configuration
  - Sessionfactory
  - Session
- Transaction management
  - Transactionfactory
  - Transaction
- Object relational mapping
  - Persistenzklassen
  - Tabellen
  - Mapping Files

Connection Management – Session Factory

Session Factory

- wird einmal erzeugt
- Dient als Factory für Sessions
- Konfiguration durch hibernate.cfg.xml
- Lädt und kennt alle Class-Mappings
- Session Factory ist thread-safe kann von vielen Threads gleichzeitig aufgerufen werden
- hält eine ThreadLocal Variable für die Session des aktuellen Threads

Connection Management - Session

Session

- Bindeglied zwischen der Java-Applikation und den Hibernate-Diensten
- Bietet Methoden für Insert-, Update-, Delete- und Query-Operationen.
- Session ist nicht thread-safe und darf nur von einem Thread verwendet werden
- Factory für Transaktionen

Connection Management - Vorgehen

- Erstellen der Hibernate-Konfigurationsdatei (hibernate.cfg.xml)
- Erstellen der Klasse für SessionFactory
- Erstellen der Klasse für Session

Connection Management - Hibernate Konfiguration

hibernate.cfg.xml

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

        <property name="connection.driver_class"> DriverKlasse </property>
        <property name="connection.url"> url zur Datenbank </property>
        <property name="connection.username">username</property>
        <property name="connection.password">passwort</property>
        <property name="dialect">Datenbankdialekt</property>
        <property name="show_sql">true</property>

        <!-- mapping files -->
        <mapping resource="mappingfile 1 "/>
                .
                .
        <mapping resource="mappingfile n "/>

    </session-factory>

</hibernate-configuration>
```

Connection Management - Hibernate Konfiguration

hibernate.cfg.xml

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

   <property name="connection.driver_class"> DriverKlasse </property>
   <property name="connection.url"> url zur Datenbank </property>
   <property name="connection.username">username</property>
   <property name="connection.password">passwort</property>
   <property name="dialect">Datenbankdialekt</property>
   <property name="show_sql">true</property>

      <!-- mapping files -->
      <mapping class="classA "/>
      .
      .
      <mapping class="classZ "/>
    </session-factory>

</hibernate-configuration>
```

Connection Management - Hibernate Konfiguration(2)

Konfiguration für Oracle Labordatenbank

```
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
                </property>
    <property name="connection.url">
                jdbc:oracle:thin:@iwi-lkit-db-01:1521:LAB1
                </property>
   <property name="connection.username">vislabxx</property>
   <property name="connection.password">vislabxx</property>
     <property name="dialect">org.hibernate.dialect.OracleDialect</property>
   <property
        name="hibernate.current_session_context_class">thread</property>
   <property name="show_sql">true</property>

      <!-- mapping files -->
       <mapping resource="vislabExample/model/db/Customer.hbm.xml" />

  </session-factory>

</hibernate-configuration>
```

Connection Management - SessionFactory

Erzeugen und Initialisieren der Session Factory

```
public class HibernateUtil {
    .
    .
    private static final SessionFactory sessionFactory;

    static {  try {
                // Create the SessionFactory from hibernate.cfg.xml

          Configuration configuration = new Configuration().configure();
          StandardServiceRegistryBuilder builder =
                                    new StandardServiceRegistryBuilder().
                                    applySettings(configuration.getProperties());
          sessionFactory = configuration.buildSessionFactory(builder.build());

          } catch (Throwable ex) {
               System.out.println("Initial SessionFactory creation failed." + ex);
          throw new ExceptionInInitializerError(ex);
                  }
          }
```

Transaction Management

Transaktion

- Abstrahiert die Anwendung von den darunterliegenden JDBC Transaktionen
- Ein Transaktionsobjekt ist immer mit einem Sessionobjekt verbunden, innerhalb einer Session können nacheinander mehrere Transaktionen stattfinden.
- Jede DB-Operation, auch lesende, muss in Transaktion eingebettet werden
- Transaktionsobjekt instantiieren:

```
Transaction tcx=session.beginTransaction();
```

- Transaktionsgrenzen müssen definiert werden durch

```
session.beginTransaction();
        .
session.getTransaction().commit();
```

- Bei fehlerhafter Ausführung der Transaktion session.getTransaction().rollback();

Object Relational Mapping (ORM)

Anforderungen an Persistenzklasse

Persistenzklasse ⇔ Java Bean(Plain Old Java Object (POJO))

- Muss als Persistenzobjekt deklariert werden
- Default-Konstruktor
- get/set-Methoden für Properties
- Persistenzobjekt benötigt einen Identifier

Beispiel Persistenzobjekt

Persistenz-Objekte müssen der JavaBean Spezifikation entsprechen

```
public class Customer  implements java.io.Serializable {

    private String username;
    private String password;
    private String lastname;
    private String firstname;

   /** default constructor */
   public Customer() {    }

   // Property accessors

   public String getUsername() {
      return this.username;    }

   public void setUsername(String username) {
      this.username = username;    }

   public String getPassword() {
      return this.password;    }

   public void setPassword(String password) {
      this.password = password;    }
```

Deklaration des Object Mappings

- XML- Mapping Files (<classname>.hbm.xml)
- Java Annotationen in den zu persistierenden Klassen

Entity

Entity = Java Objekt = Persistenz-Objekt

Im XML File:

```
<class name="Classname" table= "Tablename" >
    ......
</class>
```

Durch Annotation in der zu persistierenden Klasse:

```
@Entity
@Table(name="Tablename")
public class Classname implements Serializable {
.....
}
```

Identifier

```
<class name="Classname" table= "Tablename" >
    <id name="id"
       type="Integer "
       column="column_name"
       <generator class="generatorClass"/>
    </id>
</class>
```

```
@Entity
public class Classname {
        @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
        Integer getId() { ... }
}
```

Generator class: native
assigned
sequence (Oracle, PostgreSQL)
identity (MySQL, DB2)

Property

```
<property name="propertyName"
          column="column_name"
          type="typename"
/>
```

```
@Column(name="column_name")
   private String propertyName;
```

Beispiel Hibernate Mapping File

```
<hibernate-mapping>
   <class name="vislabExample.model.db.Customer" table="customer" >
      <id name="username" type="string">
         <column name="username" length="8" />
         <generator class="assigned" />
      </id>

      <property name="password" type="string">
         <column name="password" length="8" /> </property>

      <property name="lastname" type="string">
         <column name="lastname" length="20" /> </property>

      <property name="firstname" type="string">
         <column name="firstname" length="20" /> </property>

   </class>
</hibernate-mapping>
```

Beispiel annotierte Klasse

```
@Entity
@Table(name="CUSTOMER")
public class Customer  implements java.io.Serializable {

    @Id
    @Column(name="username")
    private String username;

    @Column(name="firstname")
    private String firstname;

    @Column(name="lastname")
    private String lastname;

    @Column(name="password")
    private String password;
}
```

Assoziationen .hbm.xml

```
<class name="Product">
    <id name="serialNumber"
        column="productSerialNumber"/>
    <set name="parts">
        <key column="productSerialNumber" not-null="true"/>
        <one-to-many class="Part"/>
    </set>
</class>
```

```
public class Product {
    private String serialNumber;

    private Set<Part> parts = new HashSet<Part>();
     ......
}
```

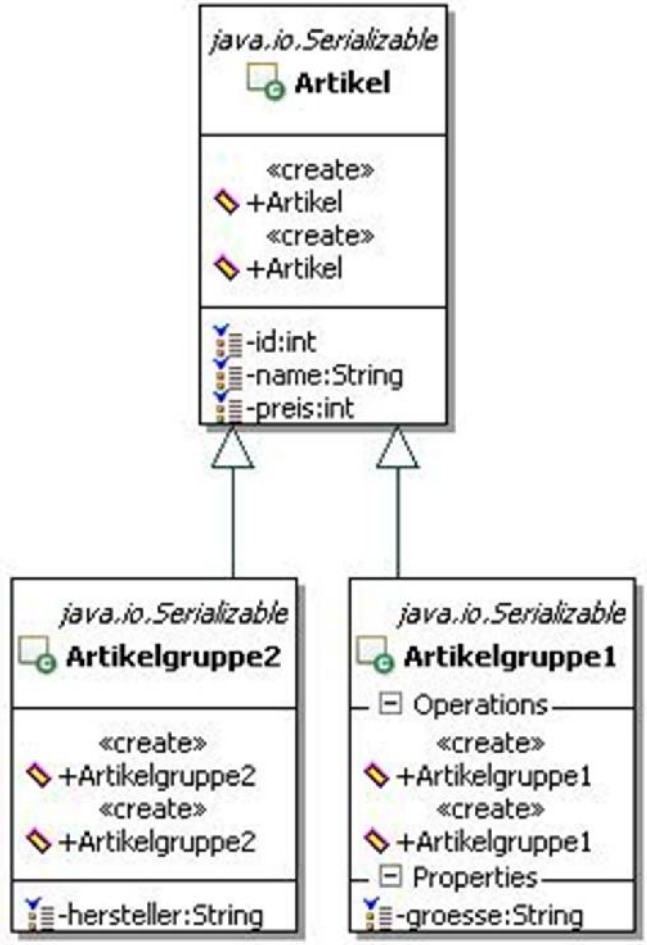
weitere Mapping Typen: many-to-one, one-to-one, many-to-many
weitere Collection Typen: java.util.Set, java.util.List, java.util.Map, java.util.Collection

Assoziationen annotiert

```
@Entity
@Table(name="PRODUCT")
public class Product {

    @Id
    private String serialNumber;

    @OneToMany
    @JoinColumn(name="PARTID")
    private Set<Part> parts = new HashSet<Part>();
     ......
}
```

```
@Entity
@Table(name="PART")

public class Part {

@Id
@Column(name="PARTID")
private int id;

 ...
}
```

Beispiel für Vererbung



Vererbung mit Hibernate modellieren

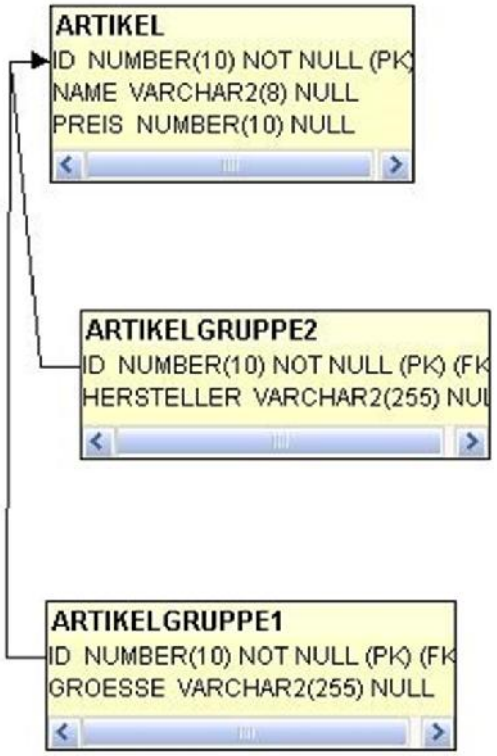
Verschiedene Strategien

- eine Tabelle pro Klasse
- eine Tabelle pro konkreter Klasse
- eine Tabelle pro Klassen Hierarchie

Beispiel Vererbung

Eine Tabelle pro Klasse

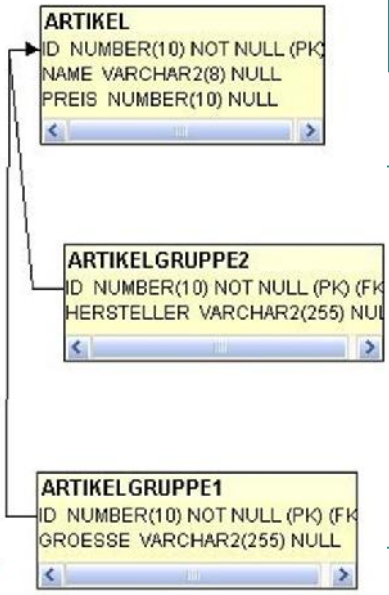
```
<hibernate-mapping>

   <class name="model.Artikel" table="ARTIKEL">
            <id name="id" type="int">
               <column name="ID"  />
               <generator class="native" />
            </id>
            <property name="name" type="string">
               <column name="NAME" length="8" />
            </property>
            <property name="preis" type="int">
               <column name="PREIS" />
            </property>

<joined-subclass name="model.Artikelgruppe1" table="ARTIKELGRUPPE1">
   <key column="id" />
   <property name="groesse" type="string" />
 </joined-subclass>
<joined-subclass name="model.Artikelgruppe2" table="ARTIKELGRUPPE2">
   <key column="id" />
   <property name="hersteller" type="string"/>
 </joined-subclass>

   </class>
</hibernate-mapping>
```



Beispiel Vererbung

Eine Tabelle pro Klasse

```
@Entity
@Table(name = "Artikel")
@Inheritance(strategy =
      InheritanceType. TABLE_PER_CLASS)

public class Artikel implements java.io.Serializable {

@Id
@GeneratedValue
@Column(name = "ID", nullable = false)
private int id;


@Column(name = "NAME", length = 8)
private String name;

@Column(name = "PREIS")
private int preis;

}
```




```
@Entity
@Table(name = "ARTIKELGRUPPE1")
public class Artikelgruppe1 extends
            Artikel implements
                  java.io.Serializable {

@Column(name = "groesse")
private String groesse;
}
```

```
@Entity
@Table(name = "ARTIKELGRUPPE2")
public class Artikelgruppe2 extends
            Artikel implements
                  java.io.Serializable {

@Column(name = "hersteller")
private String hersteller;
```

Beispiel Vererbung — Eine Tabelle pro konkreter Klasse

```
<hibernate-mapping>

   <class name="model.Artikel" abstract="true">
            <id name="id" type="int">
               <column name="ID"  />
               <generator class="native" />
            </id>
            <property name="name" type="string">
               <column name="NAME" length="8" />
            </property>
            <property name="preis" type="int">
               <column name="PREIS" />
            </property>

      <union-subclass name="model.Artikelgruppe1" table="ARTIKELGRUPPE1"
         <property name="groesse" type="string" />
       </union-subclass>
      <union-subclass name="model.Artikelgruppe2" table="ARTIKELGRUPPE2"
         <property name="hersteller" type="string"/>
       </union-subclass>

   </class>
</hibernate-mapping>
```

ARTIKELGRUPPE1
ID NUMBER(10) NOT NULL (PK)
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
GROESSE VARCHAR2(255) NUL

ARTIKELGRUPPE2
ID NUMBER(10) NOT NULL (PK)
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
HERSTELLER VARCHAR2(255) NU

Beispiel Vererbung — Eine Tabelle pro konkreter Klasse

```
@Entity
@Table(name = "Artikel")
@Inheritance(strategy =
      InheritanceType. JOINED)

public class Artikel implements java.io.Serializable {

@Id
@GeneratedValue
@Column(name = "ID", nullable = false)
private int id;


@Column(name = "NAME", length = 8)
private String name;

@Column(name = "PREIS")
private int preis;

}
```

ARTIKELGRUPPE1
ID NUMBER(10) NOT NULL (PK)
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
GROESSE VARCHAR2(255) NUL

ARTIKELGRUPPE2
ID NUMBER(10) NOT NULL (PK)
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
HERSTELLER VARCHAR2(255) NU

```
@Entity
@Table(name = "ARTIKELGRUPPE1")
@PrimaryKeyJoinColumn(name="ID")
public class Artikelgruppe1 extends
      Artikel implements
            java.io.Serializable {

@Column(name = "Groesse")
private String groesse;
}
```

```
@Entity
@Table(name = "ARTIKELGRUPPE2")
@PrimaryKeyJoinColumn(name="ID")
public class Artikelgruppe2 extends
            Artikel implements
java.io.Serializable {

@Column(name = "Hersteller")
private String hersteller;
}
```

Beispiel Vererbung

Eine Tabelle pro Klassenhierarchie

```
<hibernate-mapping>
   <class name="model.Artikel" >
            <id name="id" type="int">
               <column name="ID"  />
               <generator class="native" />
            </id>
 <discriminator column="discriminator" type="string" length="2" />
      <property name="name" type="string">
         <column name="NAME" length="8" />
      </property>
      <property name="preis" type="int">
         <column name="PREIS" />
      </property>

  <subclass name="model.Artikelgruppe1" discriminator-value="A1">
          <property name="groesse" type="string" />
 </subclass>
 <subclass name="model.Artikelgruppe2" discriminator-value="A2">
  <property name="hersteller" type="string"/>
 </subclass>

   </class>
</hibernate-mapping>
```

ARTIKEL
ID NUMBER(10) NOT NULL (PK)
DISCRIMINATOR VARCHAR2(2) NOT NUL
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
GROESSE VARCHAR2(255) NULL
HERSTELLER VARCHAR2(255) NULL

Beispiel Vererbung Annotationen

```
@Entity
@Table(name = "ARTIKEL")
@Inheritance(strategy=
            InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
   name="discriminator",
   discriminatorType=DiscriminatorType.STRING
)
@DiscriminatorValue(value="A")
public class Artikel {

@Id
@GeneratedValue
@Column(name = "Id")
private Long id;

@Column(name = "NAME")
private String name;

@Column(name = "PREIS")
private int preis;
```

```
@Entity
@Table(name=" ARTIKELGRUPPE1")
@DiscriminatorValue("A1")
public class Artikelgruppe1 extends Artikel{

@Column(name = "Groesse")
private String groesse;

}
```

ARTIKEL

ID NUMBER(10) NOT NULL (PK)
DISCRIMINATOR VARCHAR2(2) NOT NUL
NAME VARCHAR2(8) NULL
PREIS NUMBER(10) NULL
GROESSE VARCHAR2(255) NULL
HERSTELLER VARCHAR2(255) NULL

```
@Entity
@Table(name="ARTIKELGRUPPE2")
@DiscriminatorValue("A2")
public class Artikelgruppe2 extends Artikel{

@Column(name = "Hersteller")
private String hersteller;
}
```

Object relational mapping

- Java-Objekte
- Tabellen der Datenbank
- Definition in XML-Files

Tools zur Unterstützung

- Eclipse Plugin Hibernate Tools
  http://tools.jboss.org/features/hibernate.html
- Ant build Tool

Vorgehensweisen zur Umsetzung

- Top down — Java Klassen des Business-Modells erstellen, Mapping-Dateien generieren, DB-Schema generieren (Annotationen)
- Middle — Mapping Dateien erstellen, Java Klassen des Business-Modells generieren, DB-Schema generieren
- Bottom up — DB-Schema erstellen, generieren der Mapping-Dateien, Java Klassen generieren (Reverse- Engineering)
- Meet in the middle — Java Modell erstellen, DB-Schema erstellen, Mapping-Dateien schreiben

Objekte bearbeiten

session.load(object,id) ein vorhandenes Objekt mit id lesen

session.get(object,id) lesen, falls vorhanden, sonst null

session.save(object) speichern, macht nicht nur Objekt persistent, sondern auch evtl. assoziierte Objekte

session.update(object)

session.flush() Änderungen automatisch gespeichert

session.delete(object)

session.createQuery(HQLquery)

session.createCriteria(Class);

Objekt lesen (Beispiel)

Lesen von Objekten

```
public Customer getCustomerByPrimaryKey(String primaryKey) {    /* a
   Hibernate session */
   Session session =
         HibernateUtil.getSessionFactory().getCurrentSession();
   session.beginTransaction();
   Customer customer =
         (Customer) session.get(Customer.class, primaryKey);
   session.getTransaction().commit();
   return customer;  }
```

SQL:

```
select customer0_.username as username0_0_,
    customer0_.password as password0_0_,
    customer0_.lastname as lastname0_0_,
    customer0_.firstname as firstname0_0_ from vislab00.customer customer0_ where
    customer0_.username=?
```

```
DEBUG AbstractBatcher:413 - preparing statement
DEBUG StringType:79 - binding 'Knodel' to parameter: 1
```

Objekte speichern (Beispiel)

Speichern eines Objekts

```
public void saveCustomer(Customer customer) {
   /* a Hibernate session */

    Session session = HibernateUtil.getSessionFactory().getCurrentSession();

    session.beginTransaction();
          session.save(customer);
    session.getTransaction().commit();}
```

SQL:

```
insert into vislab00.customer (password, lastname, firstname, username)
    values (?, ?, ?, ?)
```

Objekte löschen (Beispiel)

```
Session session = getSessionFactory().openSession();

session.beginTransaction();
Customer customer =
        (Customer) session.get(Customer.class, primaryKey);

// deleting the loaded customer
session.delete(customer);

session.getTransaction().commit();
```

Suchen nach Objekten

Wenn der Identifier(primary key) eines Objektes nicht bekannt ist, wird eine Query benötigt, um dieses Objekt zu finden.

Möglichkeiten in Hibernate:

- QBC (Query by Criteria)
- HQL (Hibernate Query Language) = Objektorientierte Suchsprache
  - Objektorientierte Erweiterung zu SQL
  - Hibernate übersetzt HQL nach SQL
- Query in native SQL (eventuell nicht portabel)

Criteria Query (Beispiel)

```
public void listAllCustomerByCriteria(){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();

    Criteria crit = session.createCriteria(Customer.class);
    crit.add(Restrictions.like("username", "%xyz%"));
    List results = crit.list();

    Iterator iter = results.iterator();
    if (!iter.hasNext()){
      System.out.println("No Usernames to display.");
       return;      }
    while (iter.hasNext()){
        Customer customer = (Customer) iter.next();
       System.out.println("Username : " + customer.getUsername() );      }
    session.getTransaction().commit();

}
```

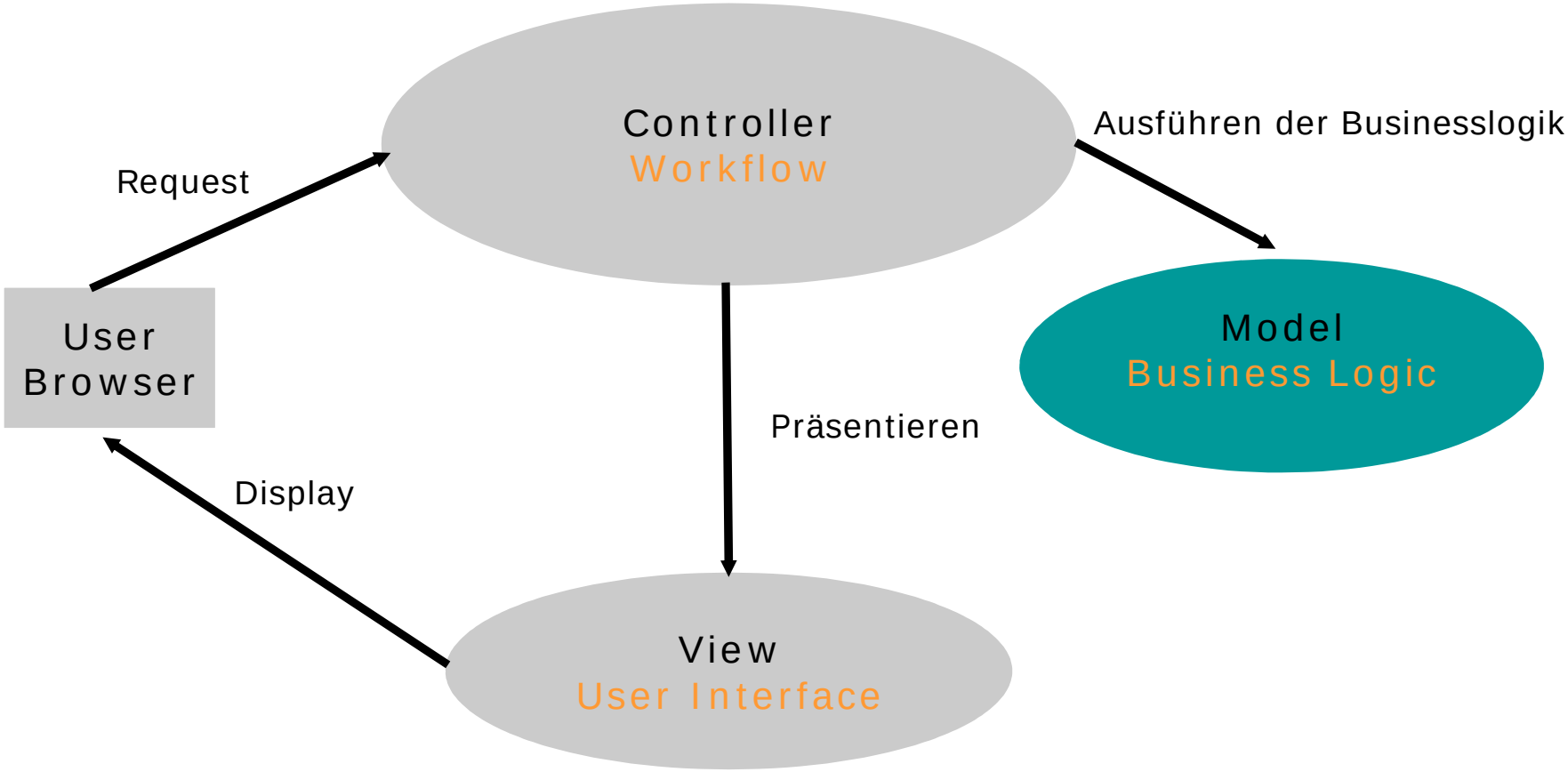
weitere Restrictions: gt, lt, between, isEmpty, and, or, ..
siehe http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html_single/#querycriteria

HQL Beispiel

Suche alle Customer, die im Usernamen die Zeichenfolge ‘xyz‘ enthalten

```
public void listAllCustomerByHql(){
    Session session = HibernateUtil.getSessionFactory().getCurrentSession();
    session.beginTransaction();
   Query query =
    session.createQuery("from Customer as c where c.username like '%xyz%'");
   for (Iterator it = query.iterate(); it.hasNext();){
         Customer customer = (Customer) it.next();
          System.out.println("Username : " + customer.getUsername() );
    }
    session.getTransaction().commit();
}
```

Bezug zu Aufgabe Teil 1 - MVC Architektur



Beispiel aus Struts-Einführung

```
public class LoginAction extends ActionSupport {

private String username; private String firstname;     // getter und setter fehlen aus Platzgründen
private String password; private String lastname;      // getter und setter fehlen aus Platzgründen

public String execute() throws Exception {

      User dbUser = new User(); dbUser.setUsername("User1"); dbUser.setFirstname("Vorname");
                                    dbUser.setLastname("Musteruser");
          dbUser.setPassword("pas123");

      /** hier ist die Schnittstelle zur Geschäftslogik, Verarbeitung der eingegebenen Daten */

      if (dbUser.getUsername().equals(getUsername())) {
          if (dbUser.getPassword().equals(getPassword())) {
                                    setFirstname(dbUser.getFirstname()); setLastname(dbUser.getLastname());
                                    return SUCCESS;
          } else {
             addActionError(getText("error.user.passwordforgotten"));
             addActionError("Bitte geben Sie das richtige Passwort ein!");
             return "input";
             }
      }
      else { addActionError(getText("error.username.register"));
           return INPUT;
      }}
```

Beispiel aus Struts-Einführung

```
public class LoginAction extends ActionSupport {

private String username; private String firstname;          // getter und setter fehlen aus Platzgründen
private String password; private String lastname;           // getter und setter fehlen aus Platzgründen

public String execute() throws Exception {

/** hier ist die Schnittstelle zur Geschäftslogik, Verarbeitung der eingegebenen Daten */
     CustomerManager customerManager = new CustomerManager();

     Customer customer =
         customerManager.getCustomerByPrimaryKey(getUsername());

       if (customer == null)
       {
             // do something
             }
        else {
             if (customer.getPassword().equals(getPassword())) {
          }
     }}
```

Manager-Klasse Businesslogik

```
public Customer getCustomerByPrimaryKey(String primaryKey)
{

 /* a Hibernate session */
  Session session =
          HibernateUtil.getSessionFactory().getCurrentSession();

  session.beginTransaction();
  Customer customer =
          (Customer) session.get(Customer.class, primaryKey);
  session.getTransaction().commit();

  return customer;
 }
```

Hibernate Anforderungen

Eine Hibernate Anwendung benötigt:

- Hibernate-Konfigurationsdatei (hibernate.cfg.xml)
- Persistenz-Java-Klassen
- pro persistenter Klasse eine Hibernate Mapping Datei (*.hbm.xml) (nicht für annotierte Persistenzklassen)
- Hibernate Libraries (hibernate-commons-annotations-4.x.x.jar
  hibernate-core-4.x.x.jar
  hibernate-jpa-2.1-api-1.x.x.jar, u.a.)
- Abfragen an die Datenbank
- Datenbank mit dem Datenbank-Schema

Vorgehensweise Hibernate

- Erstellen des Hibernate Konfigurationsfiles hibernate.cfg.xml
- Generieren bzw. Erstellen der Mapping Files und des Java Codes und der Tabellen
- Erstellen einer SessionFactory singleton Klasse
- Beschaffen einer Session, bearbeiten von persistenten Objekten in Transaktionen

Links

Tutorials:

http://www.roseindia.net/hibernate/index.shtml
http://www.developer.com/open/article.php/3559931
http://www.tutorialspoint.com/hibernate/

References:

http://hibernate.org/orm/

http://hibernate.org/orm/documentation/
http://www.xmarks.com/site/www.xylax.net/hibernate/
http://www.java2s.com/Code/Java/Hibernate/CatalogHibernate.htm
http://docs.jboss.org/hibernate/orm/4.3/manual/en-US/html_single

Sonstiges:

http://www.agiledata.org/essays/mappingObjects.html
http://ndpsoftware.com/HibernateMappingCheatSheet.html
http://viralpatel.net/blogs/introduction-to-hibernate-framework-architecture/