Datenbanken 2

Peter Muth · Inge Schestag · Uta Störl

Hochschule Darmstadt – Fachbereich Informatik

Sommersemester 2017

Organisatorische Vorbemerkungen

- **Vorlesungsunterlagen**
  - Online unter https://www.fbi.h-da.de/organisation/personen/schestag-inge/datenbanken-2-bachelor.html
  - werden im Laufe des Semesters jeweils einige Tage vor der Vorlesung online bereit gestellt
- **Praktikum**
  - Voraussetzung für Teilnahme: PAD 1 und PAD 2 bestanden, DB 1 begonnen (d.h. mindestens einmal die Klausur mitgeschrieben)
  - Termine und Aufgabenstellungen auf der Homepage
  - Teilnahme am Praktikum + Testatpflicht spätestens eine Woche nach dem Praktikumstermin (git-push)
  - **Vorbereitungsaufgaben** sind vor dem Praktikum entsprechend des Termins auf dem Aufgabenblatt einzureichen.
- **Klausur**
  - An- bzw. Abmeldung ausschließlich über das OBS
  - Hilfsmittel: 1 (beidseitig) beschriebenes bzw. bedrucktes A4-Blatt
  - letzter Vorlesungstermin dient zur Klausurvorbereitung
  - bestandenes Praktikum ist Voraussetzung für die Teilnahme

Datenbanken 2

– Kapitel 1: Einführung –

Einführung

Inhalte des Kapitels

- Motivation für das Thema objekt-relationales Mapping (ORM)
- Überblick über die in der Lehrveranstaltung Datenbanken 2 behandelten Themen
- Vorstellung unseres Projektes im Praktikum

Lernziele

- Kennenlernen der Grundproblematiken beim Arbeiten mit
  - objekt-orientierten Strukturen in der Anwendungsschicht und
  - relationalen Strukturen im Datenbankschema
- Lösungsansätze für diesen *Impedance Mismatch* kennen und anwenden können

Impedance Mismatch · 1

- **Objektorientierte Programmiersprachen** kapseln Daten und Verhalten in Objekten und benutzen objektorientierte Konzepte wie
  - Vererbung
  - Objektidentität
  - komplexe Beziehungen mit beliebiger Graphennavigation.

- **Relationale Datenbanken** basieren auf der Relationen Algebra und repräsentieren Daten in zweidimensionalen Tabellen.
  Wichtige Konzepte:
  - Primärschlüssel mit PK-Constraints
  - Fremdschlüssel mit FK-Constraints
  - eingeschränkte Graphennavigation (Fremdschlüsselspalten dürfen nicht mengenwertig sein – warum?)

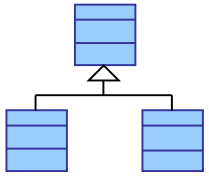
⇒ Die Konflikte, die aus den Strukturunterschieden zwischen beiden Systemen entstehen, bezeichnet man auch als **Impedance Mismatch**.

lat. impedire: hemmen, hindern · *Impedance* = Wechselstromwiderstand in der Elektrotechnik

Impedance Mismatch · 2

Objektorientierte Applikation

Objektmodell – oo Klassendiagramm
Objekte
Collections
OIDs
Vererbung
…



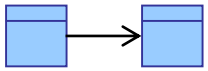
?

?

Wer mappt wie?
Welches ist das „führende“ System?

Relationale Datenbank

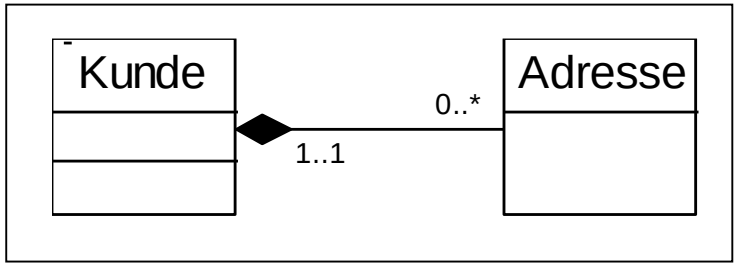
Relationenmodell
Relationen
elementare Datentypen
PK-Constraints
FK-Constraints
…



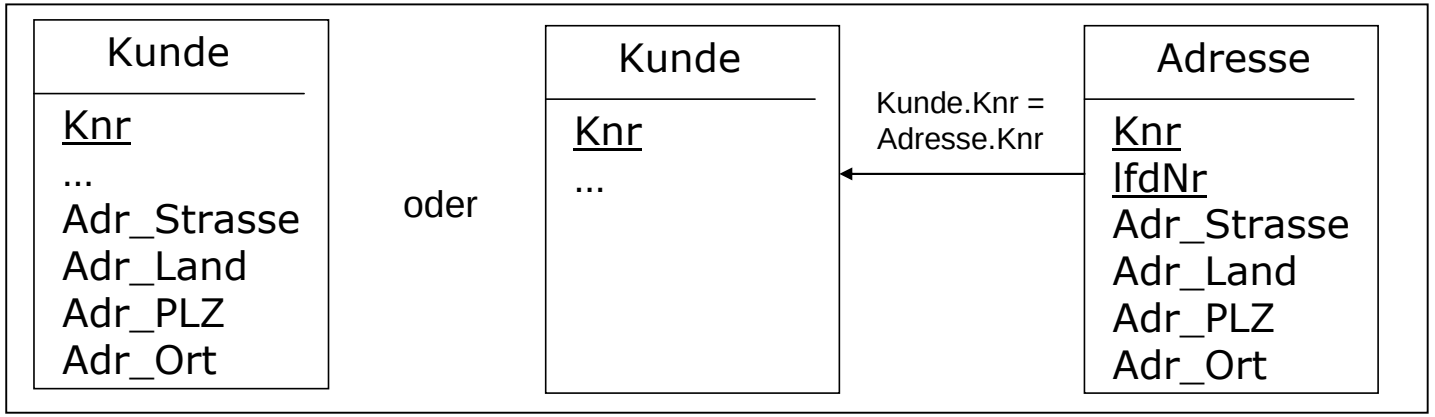
Impedance Mismatch – Datentypen

Komplexe vs. elementare Datentypen

- Objekte können beliebig komplexe Eigenschaften haben …



- … während Relationen (Tabellen) nur Attribute (Spalten) mit elementaren Datentypen enthalten dürfen:



Impedance Mismatch – Identitäten

Objektidentität …

- Zwei Objekte (in Java) sind gleich, wenn die `equals()`-Methode `true` zurück gibt. Sie sind jedoch nur dann identisch (`object1==object2` ergibt `true`), wenn `object1` und `object2` als Referenztypen auf das gleiche Objekt zeigen.
- Die Standardimplementierung der `equals()`-Methode aus der `Object`-Klasse testet die Referenzen von `object1` und `object2` und entspricht somit dem Identitätsoperator `object1 == object2`.

Objektgleichheit in Java:

```
object1.equals(object2);
```

Objektidentität in Java:

```
object1 == object2
```

→ Es empfiehlt sich deshalb immer, die **`equals()`-Methode** zu überschreiben und den eigenen Bedürfnissen anzupassen. Hierbei sollte die Annotation `@Override` genutzt werden, um ein versehentliches Verändern der Signatur in jedem Fall zu vermeiden.

- Beispiel: Die `equals()`-Methode der `String`-Klasse vergleicht die Sequenz der Zeichen beider String-Objekte.

Impedance Mismatch – …vs. PKs

Objektidentität …

→ Mit der equals()-Methode sollte auch stets die Methode **hashCode()** in eigenen Klassen überschrieben werden:

- Wenn die equals()-Methode für den Vergleich zweier Objekte o1 und o2 den Wert true zurück gibt, müssen auch die Hashwerte gleich sein.

→ Die Methode hashCode() weist jedem Objekt eine (möglichst eindeutige) ganze Zahl zu, die das Objekt identifiziert, hat also den Return-Type int. Diese ganze Zahl nennt man Hashcode bzw. Hash-Wert.

- Zwei Datensätze einer Tabelle sind dann gleich, wenn ihre Spaltenwerte identisch sind.

  Sind sie dann auch identisch?

→ In Kapitel 2 werden wir erläutern, welche Implementierungen der equals()- und der hashCode()-Methode beim Mappen von Objekten auf Instanzen einer Datenbankrelation sinnvoll sind bzw. gemäß JPA-Spezifikation verlangt werden.

Impedance Mismatch – Weitere Aspekte

Weitere Probleme gibt es bei der Implementierung von

- **Beziehungen**
- **Vererbungshierarchien**
- **Graphennavigation**

Problematiken, die über den Impedance Mismatch hinaus gehen:

- **Transaktionsmanagement** – wie werden Änderungen synchronisiert?
- **Caching** – wo erfolgt ein Caching der Daten und wie werden verschiedene Chaches synchronisiert?

→ Auf alle Aspekte wird in den folgenden Kapiteln näher eingegangen.

Objekt-relationales Mapping

- Ein Lösungsansatz zur Behebung der Konflikte des (objekt-relationalen) Impedance Mismatch ist ein – möglichst automatisiertes, transparentes – objekt-relationales Mapping.

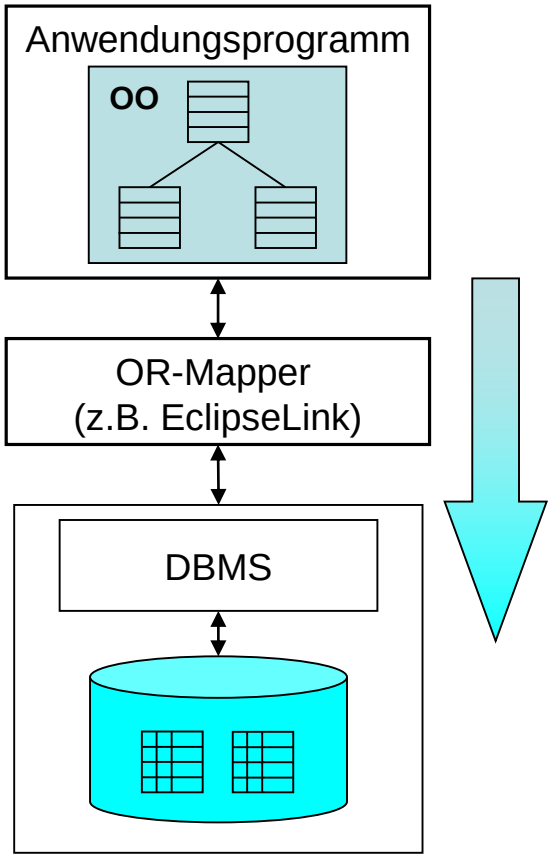
- Unter objekt-relationalem Mapping (**OR-Mapping**) versteht man ein Mapping zwischen einem objektorientierten Modell (Klassendiagramm) und einem Relationenmodell.

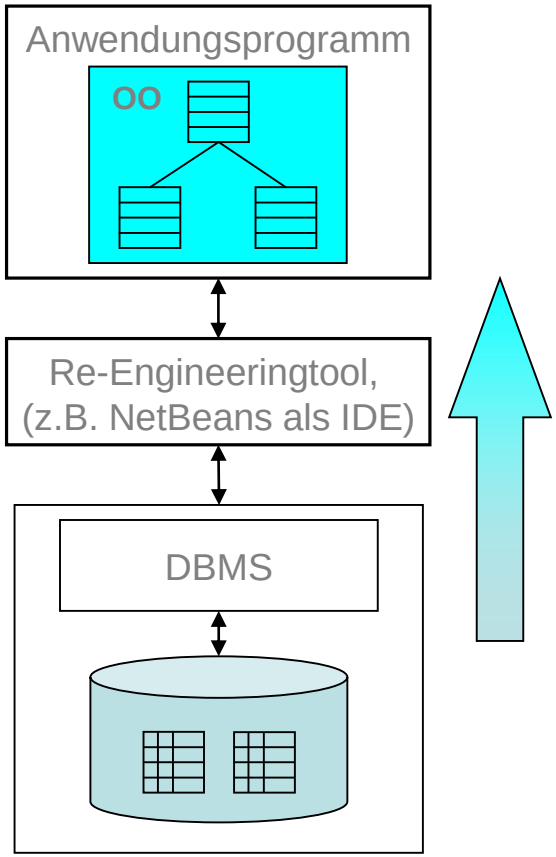
- Ziel ist es, dieses Mapping weitestgehend automatisiert vornehmen zu lassen bei gleich bleibender Nutzung der Vorteile einer objektorientierten Programmiersprache und eines relationalen Datenbankschemas.

- Frameworks wie z.B. EclipseLink (http://www.eclipse.org/eclipselink/) oder Hibernate (www.hibernate.org) unterstützen ein solches OR-Mapping. Man spricht auch von **OR-Mappern**.

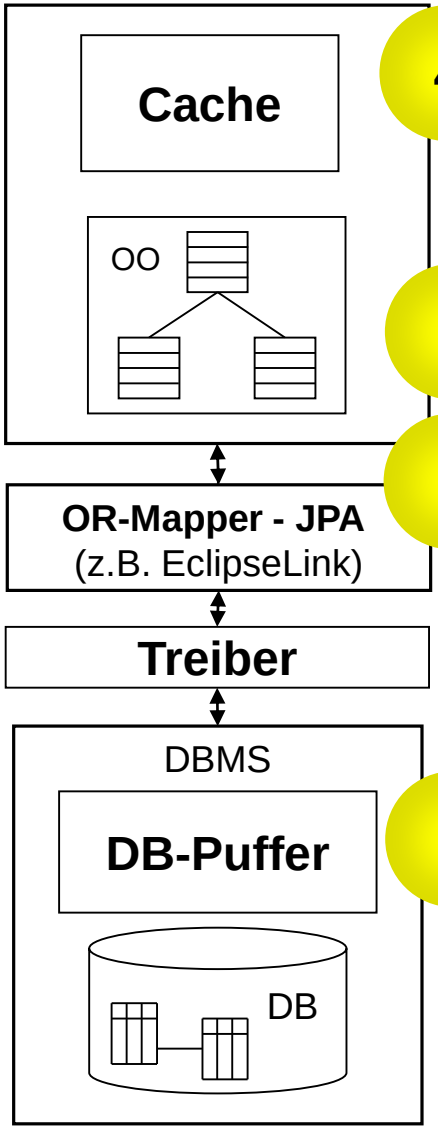
Objekt-relationales Mapping – Strategien

- **Top-Down**: Erstellen eines Klassendiagramms und mappen auf ein relationales DB-Schema.
- **Bottom-Up**: Erstellen eines relationalen DB-Schemas und Re-Engineering auf Entity-Klassen.





Aufbau der Vorlesung Datenbanken 2



4 **Transaktionsmanagement** und **Caching**
auf der Anwendungsseite
Ladestrategien: wann werden welche Daten geladen?

3 **Datenbankabfragen**

2 **Java Persistence API – JPA**

- Entity-Klassen
- Persistenz
- Beziehungen

5 Caching auf DB-Seite: **Pufferverwaltung**
Optimierung des Zugriffspfades durch entsprechende Speichertechniken

Unser Projekt im Praktikum

WISSENSTEST

Die Datenbasis

- Auf dem Datenbankserver werden zu unterschiedlichen Kategorien Fragen gespeichert. Zu jeder Frage gibt es genau vier Antworten, von denen genau eine richtig ist und die anderen falsch sind.
- Jede Frage gehört genau einer Kategorie an.
- Eine Stammdaten-Basis von Kategorien, Fragen und Antworten wird in einer entsprechenden csv-Datei zur Verfügung gestellt und darf nach Belieben erweitert werden.

Wer gilt als der Verfasser der ältesten bekannten indischen Dramen?	Banabhatta	Jayakanthan	Akhandanand	Ashvaghosha	4	Literatur
Die Stadt Pittsburgh gehört zu welchem amerikanischen Bundesstaat?	Ohio	Oklahoma	Oregon	Pennsylvania	4	Erdkunde
Welches war Hitchcocks letzter Film?	Familiengrab	Frenzy	Cocktail für eine Leiche	Marnie	1	Unterhaltung
Abdul Ahad Mohmand war erster Raumfahrer welchen Landes?	Syrien	Saudi-Arabien	Kuwait	Afghanistan	4	Technik
Welcher griechische Gelehrte formulierte die Gesetze des Schwerpunktes?	Kreon	Sokrates	Archimedes	Thales	3	Physik
Welche Haarfarbe hat das "Mädchen mit den Schwefelhölzern" im gleichnamigen Märchen?	rot	schwarz	braun	blond	4	Literatur
In welchen gängigen Größen werden externe Festplatten angeboten?	1,8 / 2,5 / 3,5 Zoll	7,5 / 5,25 Zoll	3,5 / 5,25 / 8 Zoll	3,25 / 3,5 Zoll	1	Digital
Welche Früchte wachsen an einem Johannisbrotbaum?	Beerenfrüchte	Hülsenfrüchte	Nüsse	Steinfrüchte	2	Pflanzen
Wo befindet sich die Einkaufsstraße Zeil?	Köln	Frankfurt/M	Hamburg	Düsseldorf	2	Reisen

Unser Projekt im Praktikum

WISSENSTEST*

Das Spiel

- Wenn ein Spieler ein Spiel spielt, wählt er eine beliebige Anzahl von Kategorien aus (mindestens 2) und lässt sich hierzu zufällig Fragen generieren.
- Die Anzahl der Fragen wird pro Spiel zu Beginn ebenfalls festgelegt.
- Aus den angebotenen Antworten muss der Spieler genau eine auswählen – er bekommt unmittelbar eine Auskunft darüber, ob die ausgewählte Antwort richtig oder falsch war.

→ Sämtliche Spieldaten aller Spieler werden auf dem Datenbankserver gespeichert, um anschließend darauf Analysen durchführen zu können.

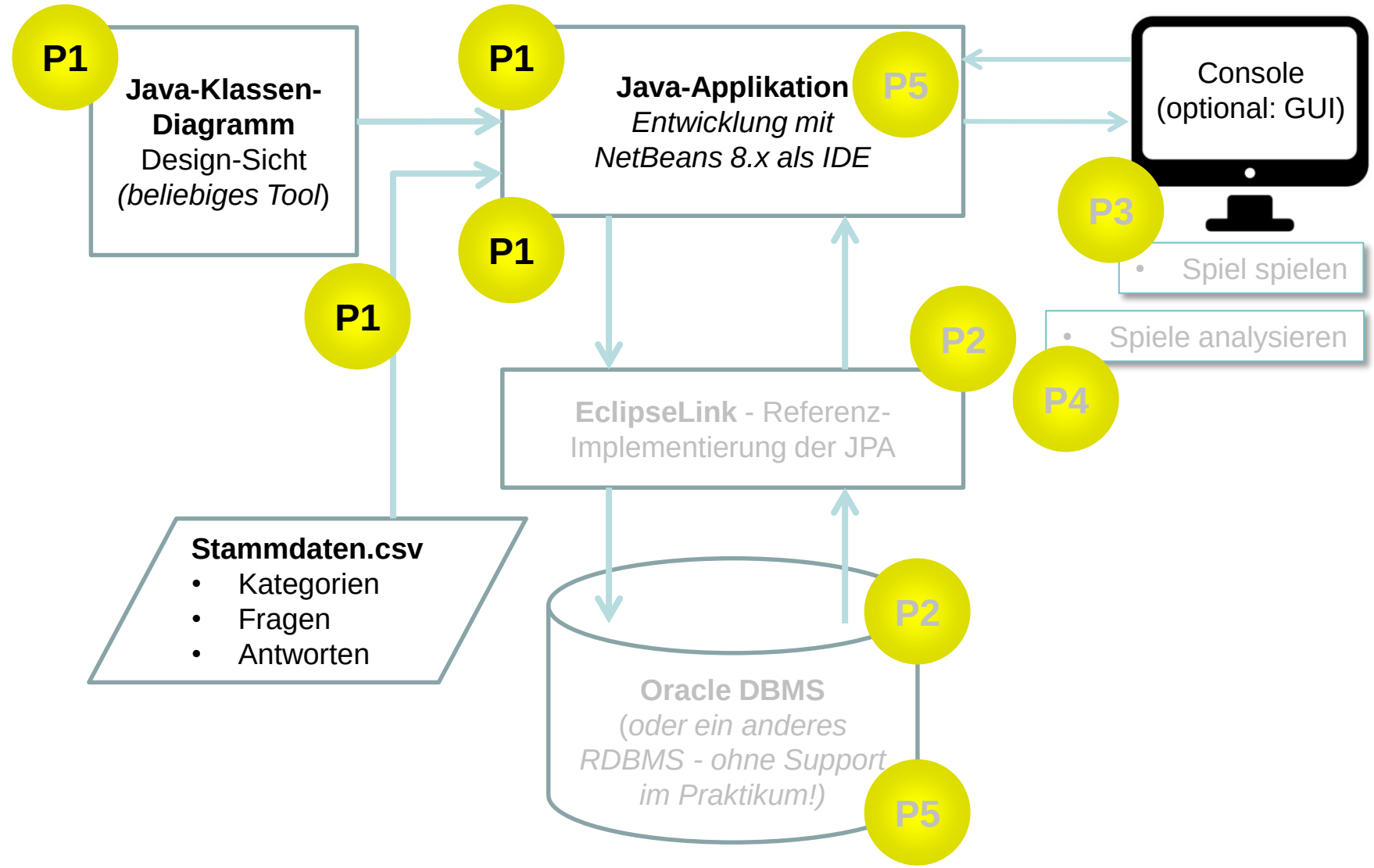
---

* Die vollständigen Projektanforderungen werden vor Praktikum 1 zum Download zur Verfügung gestellt.

Die Themen der Praktika

• Objekt-orientierte Modellierung eines Java-Klassendiagramms „Wissenstest“ • Implementierung der Entity-Klassen für die initialen Spieldaten • Einlesen der „Stammdaten“(= initiale Spieldaten) aus einer csv-Datei	**P1**
• Erwartetes Relationenmodell entwerfen • JPA-Annotationen für die Java-Entity-Klassen implementieren und Top-Down-Mapping durchführen • CRUD-Operationen auf den „Stammdaten“	**P2**
• Szenariobeschreibung zum Use Case „Wissenstest spielen“ • Implementierung des Spiels	**P3**
• Analyse der Ergebnisse – Vergleich der Spieler	**P4**
• Massendaten generieren – Performancetests und –optimierung (Applikation und Datenbankserver)	**P5**

Praktikum 1


P1
Java-Klassen-Diagramm
Design-Sicht
(beliebiges Tool)
P1
P1
P1
Java-Applikation
Entwicklung mit
NetBeans 8.x als IDE
P5
Console
(optional: GUI)
P3
• Spiel spielen
• Spiele analysieren
P2
P4
EclipseLink - Referenz-Implementierung der JPA
Stammdaten.csv
• Kategorien
• Fragen
• Antworten
P2
Oracle DBMS
(oder ein anderes RDBMS - ohne Support im Praktikum!)
P5

Literaturempfehlungen

- B. Müller, H. Wehr: Java Persistence API 2 : Hibernate, EclipseLink, OpenJPA und Erweiterungen, Hanser, 2012

  Aus der Hanser eLibrary kostenlos kapitelweise als pdf zum Download:

→ http://www.hanser-elibrary.com/action/showBook?doi=10.3139%2F9783446431294&

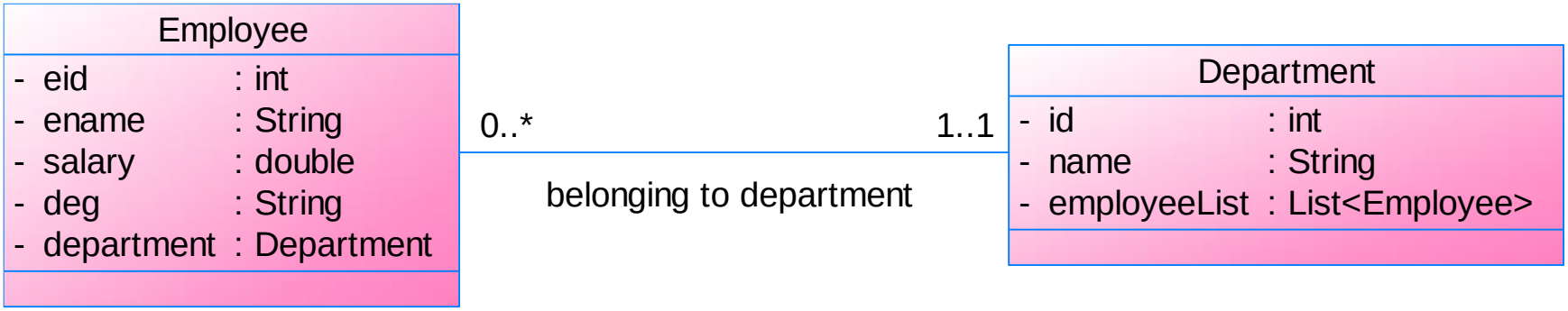
- G. Saake, K.-U. Sattler, A. Heuer: Datenbanken: Implementierungstechniken, mitp Verlag, 3. Auflage 2011

Onlinequellen zu Spezifikation und Referenzimplementierung:

- EclipseLink – die Referenzimplementierung der Java Persistence API: http://www.eclipse.org/eclipselink/
- Enterprise JavaBeans 3.2 und Java Persistence API 2.1: https://jcp.org/en/jsr/detail?id=345 bzw. https://jcp.org/en/jsr/detail?id=338

Hörsaalbeispiel · Mitarbeiterverwaltung (1)

Für eine Applikation, die die Abteilungszugehörigkeit von Mitarbeitern in einem Unternehmen verwalten soll, wird zunächst folgendes Klassen-diagramm*) für die Entity-Klassen entworfen:



- Jede Abteilung (Department) hat keinen oder beliebig viele Mitarbeiter (Employee) – „keinen“ wahrscheinlich nur zum Zeitpunkt einer Abteilungsgründung ;-)
- Jeder Mitarbeiter ist zum Zeitpunkt genau einer Abteilung zugeordnet.
- Welche Navigationsrichtung ist implementiert?
- … mit welchem Referenz-Attribut / welchen Referenz-Attributen?

*) Die getter- und setter-Methoden sowie Konstruktoren sind der Übersichtlichkeit halber nicht dargestellt!

Hörsaalbeispiel · Mitarbeiterverwaltung (2)

- Wir lassen dieses Klassendiagramm auf ein relationales Schema mappen.

- Welche Informationen müssen wir dem OR-Mapper hierfür bekannt machen
  - bzgl. der Datenbankverbindung?
  - bzgl. der Attribute in den Entity-Klassen:
    Welche Meta-Informationen sind notwendig, damit der OR-Mapper ein eindeutiges Schema generieren kann?

→ Welches Schema erwarten Sie nach dem Mapping auf das RDBMS?

→ Welche PK- und FK-Constraints erwarten Sie?

→ Wie ist die Beziehung auf dem RDBMS implementiert?

Hörsaalbeispiel (3) – OR-Mapping$_{informell}$

```java
@Entity
public class Employee
{
   @Id
   @GeneratedValue( strategy= GenerationType.AUTO )
   private int eid;
   private String ename;
   private double salary;
   private String deg;
   @ManyToOne
   private Department department;
```

```java
@Entity
public class Department
{
   @Id
   @GeneratedValue( strategy=GenerationType.AUTO )
   private int id;
   private String name;
   @OneToMany( targetEntity=Employee.class )
   private List employeelist;
```

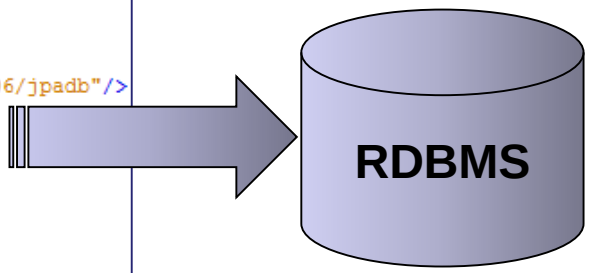
```xml
<?xml version="1.0" encoding = "UTF-8"?>

<persistence version = "2.0"
   xmlns = "http://java.sun.com/xml/ns/persistence"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://java.sun.com/xml/ns/persiste
   http://java.sun.com/xml/ns/persistence/persistence_2_0.

   <persistence-unit name = "Eclipselink_JPA" transaction-type = "RESOURCE_LOCAL">
      <class>com.tutorialspoint.eclipselink.entity.Employee</class>
      <class>com.tutorialspoint.eclipselink.entity.Department</class>

      <properties>
         <property name = "javax.persistence.jdbc.url" value = "jdbc:mysql://localhost:3306/jpadb"/>
         <property name = "javax.persistence.jdbc.user" value = "root"/>
         <property name = "javax.persistence.jdbc.password" value="root"/>
         <property name = "javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
         <property name = "eclipselink.logging.level" value = "FINE"/>
         <property name = "eclipselink.ddl-generation" value = "create-tables"/>
      </properties>

   </persistence-unit>
</persistence>
```



Beispiel-Projekte - vorlesungsbegleitend

Das vorhergehende Beispiel sowie weitere Beispiele, die im Rahmen der Vorlesung vorgestellt werden, lehnt sich an die folgende Tutorial-Website an:

→ https://www.tutorialspoint.com/jpa/index.htm

Diese Website enthält innerhalb der einzelnen JPA-Tutorials Code-Fragmente.

Wir haben aus diesen Code-Fragmenten für Sie lauffähige kleine Beispiel-Projekte auf unserem git-Repository zum Clonen bereit gestellt:

http://dakala.fbi.h-da.de/JPADatenbanken2

Alle Praktikumsteilnehmer haben auf diesen Projekten lesenden Zugriff.

In den jeweiligen Vorlesungseinheiten werden wir auf die entsprechenden Beispiel-Projekte verweisen.

Datenbanken 2

- ✓ Einführung

2. Java Persistence API
   - Entity-Klassen
   - Persistenz
   - Beziehungen

3. Datenbankabfragen

4. Transaktionsmanagement, Caching und Ladestrategien

5. Pufferverwaltung und Optimierung von Zugriffspfaden