Objektorientierte Datenbanken

Vorlesung 3

Sebastian Iwanowski
FH Wedel

Inhalt heute:

Die Java-Anbindung der ODMG

Die Realisierung des ODMG-Standards in FastObjects

Die Nützlichkeit einer Anfragesprache

Wesentliche Elemente von OQL (Object Query Language)

FastObjects-OQL

Die Java-Anbindung der ODMG

Transaktionen und Datenbanken

Es gibt die Java-Interfaces Transaction und Database:

Interface Transaction

```
public void begin();

public void commit();

public void abort();

public void checkpoint();

public boolean isOpen();

public void join();

public void leave();

public void lock()
        throws lockNotGrantedException;

public boolean tryLock();
```

Interface Database

```
public void open(String name, int accessMode)
        throws ODMGException;

public void close();

public void bind (Object object, String name)
        throws ObjectNameNotUniqueException;

public Object lookup(String name)
        throws ObjectNameNotFoundException;

public void unbind(String name)
        throws ObjectNameNotFoundException;

public void makePersistent(Object object);

public void deletePersistent(Object object);
```

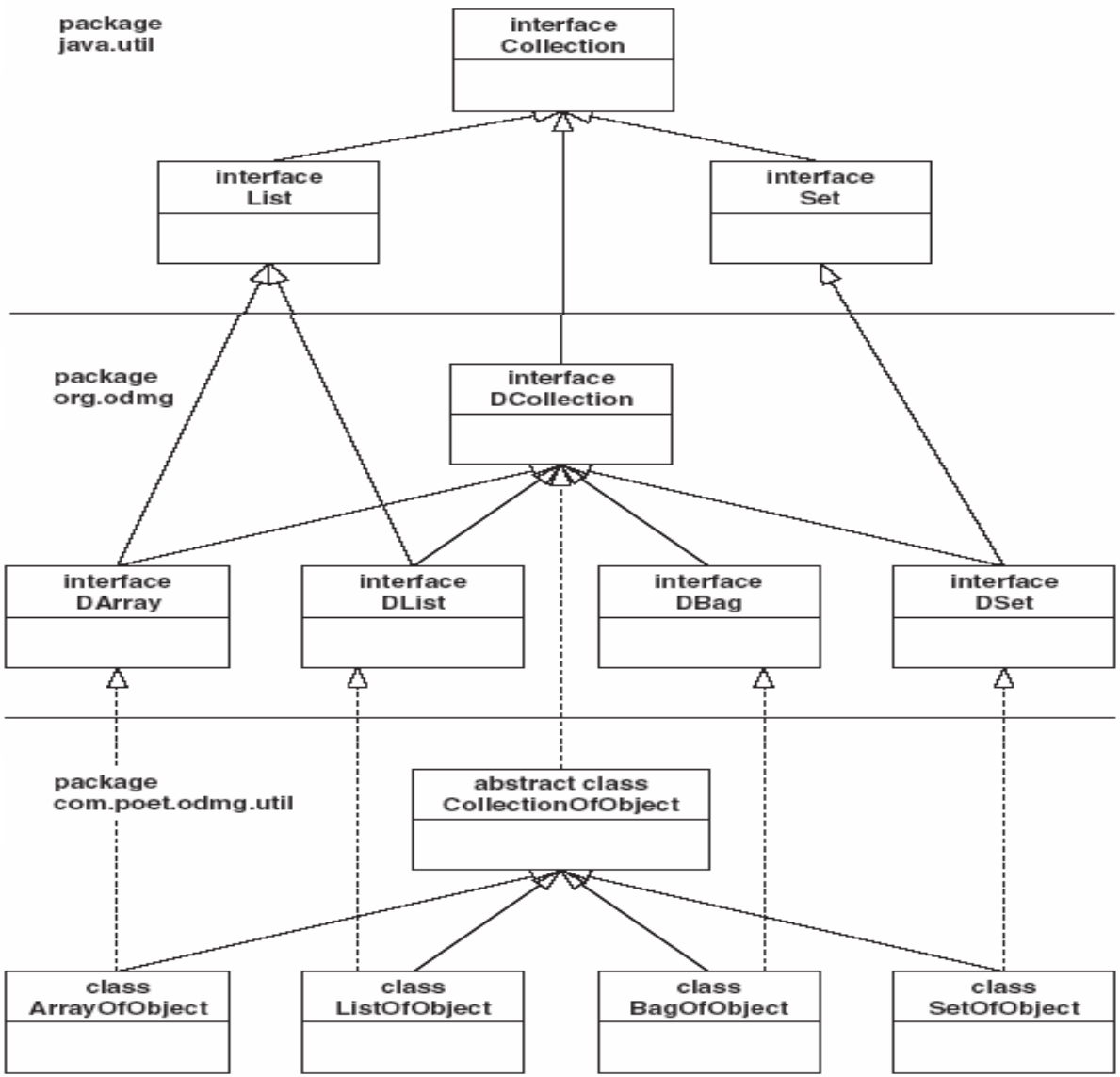
➔ ***In FastObjects: Es gibt Klassen gleichen Namens***

Collections und Sets

Java:

ODMG:

FastObjects:



Collections und Sets

Warum gibt es Subinterfaces von Collections und Sets ?

Wie erreicht man die schärferen ODMG-Spezifikationen bzgl. Collections und Sets in Java ?

➔ durch Property Files

☹ wird in FastObjects nicht berücksichtigt ☹

Extents

In ODMG für Java nicht festgelegt

☺ Extents in FastObjects implementiert ☺

Class Extent

public Extent (Database db, String className);

public Object next ();

public boolean hasNext ();

viele weitere Methoden . . .

Die Realisierung des ODMG-Standards in FastObjects

ODMG-Standard in FastObjects

- **Klassendefinitionen in Java (also kein ODL / OIF)**
- **Persistenzfähige Klassen in Datei ptj.opt festgelegt**
- **Die persistenzfähigen Klassen müssen gesondert übersetzt werden**
- **Mehrere Datenbanken werden in einem Schema zusammengefasst**
- **Die Datenbanknamen sind grundsätzlich URL-Namen**

Die Nützlichkeit einer Anfragesprache

Datenbankanfragemöglichkeiten ohne Anfragesprache

a) Extraktion eines einzelnen Objects (spezifiziert durch Name)

b) Extraktion des Extents einer Klasse

Ungeeignet für das Problem:

Finde Objekte mit bestimmten Eigenschaften !

a) Hoher Programmieraufwand

b) Hoher Hauptspeicherbedarf

Motivation für Objektanfragesprache

- **gezielte Extraktion von Objekten mit bestimmten Eigenschaften**

  mehr Komfort, mehr Effizienz

- **Abfragemöglichkeiten ohne OOP-Kenntnisse**

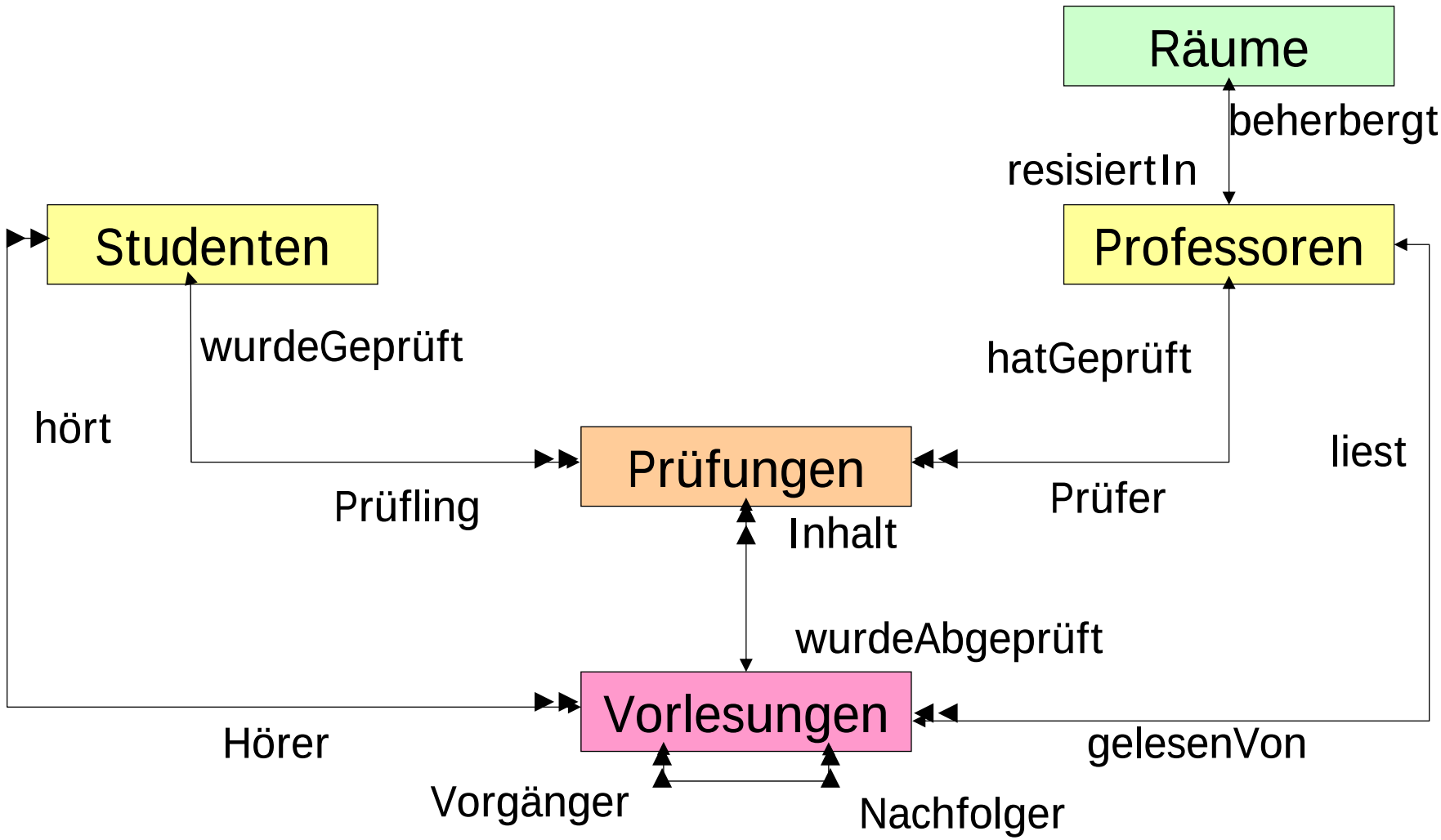
  politischer und sozialer Grund

Historie von OQL (Object Query Language)

- **implementiert vom Datenbankhersteller O_2 ca. 1990**
- **aufgenommen in den ODMG-Standard als konkurrenzloser Kandidat**

Wesentliche Elemente von OQL

Modell für die kommenden Beispiele



Modell für die kommenden Beispiele

```
class Professoren {
        attribute long PersNr;
        attribute string Name;
        attribute string Rang;
        relationship Räume residiertIn inverse Räume::beherbergt;
        relationship set(Vorlesungen) liest inverse Vorlesungen::gelesenVon;
        relationship set(Prüfungen) hatGeprüft inverse Prüfungen::Prüfer;
};

class Vorlesungen {
        attribute long VorlNr;
        attribute string Titel;
        attribute short SWS;
        relationship Professoren gelesenVon inverse Professoren::liest;
        relationship set(Studenten) Hörer inverse Studenten::hört;
        relationship set(Vorlesungen) Nachfolger inverse Vorlesungen::Vorgänger;
        relationship set(Vorlesungen) Vorgänger inverse Vorlesungen::Nachfolger;
        relationship set(Prüfungen) wurdeAbgeprüft inverse Prüfungen::Inhalt;
};
```

Modell für die kommenden Beispiele

```
class Studenten {
        attribute string Name;
        attribute boolean female;
        attribute int Fachsemester;
        relationship set(Vorlesungen) hört inverse Vorlesungen::Hörer;
        relationship set(Prüfungen) wurdeGeprüft inverse Prüfungen::Prüfling;
};


class Prüfungen {
    attribute struct Datum
        { short Tag; short Monat; short Jahr} Prüfdatum;
    attribute float Note;
    relationship Professoren Prüfer inverse Professoren::hatGeprüft;
    relationship Studenten Prüfling inverse Studenten::wurdeGeprüft;
    relationship Vorlesungen Inhalt inverse Vorlesungen::wurdeAbgeprüft;
};

class Räume {
     attribute string RaumId;
     relationship Professoren beherbergt inverse Professoren::residiertIn;
```

SQL-ähnliche Anfragen

Einfachster Fall:

Fragen nach allen Daten mit gemeinsamen Attributen:

- Finde die Namen der C4-Professoren

```
select p.Name
from p in AlleProfessoren
where p.Rang = "C4"
```

```
select p.Name
from AlleProfessoren as p
where p.Rang = "C4"
```

Navigieren in Objektpfaden

- Finde die Namen aller Studenten, die von Sokrates geprüft wurden:

```
select s.Name
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
```

- Finde die Namen aller Studenten, die bei Sokrates Vorlesung haben:

```
select s.Name
from s in AlleStudenten, v in s.hört
where v.gelesenVon.Name = „Sokrates“
```

- Finde die Namen aller weiblichen Studenten, die bei Sokrates Vorlesung haben:

```
select s.Name
from s in AlleStudenten, v in s.hört
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

Navigieren in Objektpfaden

☹ das geht nicht in FastObjects: ☹

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from s in AlleStudenten, v in s.hört
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

☺ so geht es: ☺

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from v in AlleVorlesungen, s in v.Hörer
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

„Selection of collection members not supported yet“

Rückgabe von vollständigen Objekten

Ziel: Objekte sollen in Programmiersprache weiterbearbeitet werden können

- Finde alle Studenten, die von Sokrates geprüft wurden:

```
select s
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
```

Existenzquantoren

Beispiel für Existenzquantor:

- Finde die Titel der Vorlesungen, in denen weibliche Studenten sitzen und die von Sokrates gehalten werden:

```
select v.Titel
from v in AlleVorlesungen
where (v.gelesenVon.Name = „Sokrates“) and (exists s in v.Hörer: s.female)
```

Zur Erinnerung: So ging es ohne Existenzquantor:

```
select v.Titel
from v in AlleVorlesungen, s in v.Hörer
where (v.gelesenVon.Name = „Sokrates“) and s.female
```

Allquantoren

Beispiel für Allquantor:

- Finde die Titel der Vorlesungen, in denen nur weibliche Studenten sitzen und die von Sokrates gehalten werden:

select v.Titel

from v **in** AlleVorlesungen

where (v.gelesenVon.Name = „Sokrates“) and (**for all** s in v.Hörer: s.female)

Weitere Funktionalitäten

Sortierfunktion mit **`order by`**:

- Finde alle Studenten, die von Sokrates geprüft wurden:

```
select s
from s in AlleStudenten
where s.wurdeGeprüft.Prüfer.Name = „Sokrates“
order by s.Semesterzahl DESC, s.Name ASC
```

Zählfunktion mit **`count`**:

- Ordne die Vorlesungen von Sokrates nach der Zahl der Hörer:

```
select v.Titel
from v in AlleVorlesungen
where (v.GelesenVon.Name = „Sokrates“)
order by count (select s from s in v.Hörer) DESC
```

In FastObjects nicht implementierte OQL-Funktionalitäten

- **Direkte Anfragen** (ohne select-from-where-Struktur)
- **Fragen nach zusammengesetzten Objekten**
- **Geschachtelte Anfragen**
- **Partitionierung** (group-by)
- **Benutzung von Objektmethoden in der Anfrage (inkl. Polymorphismuskonzept)**
- **...**

FastObjects-OQL

Wie werden Anfragen konkret gestellt ?

Class **OQLQuery**:

Konstruktoren:

- **`OQLQuery()`**
- **`OQLQuery(String query)`** *der „normale“ Konstruktor*
- **`OQLQuery (Transaction tn)`**
- **`OQLQuery (Database db)`**

Methoden:

- **`void create(String query)`** *nötig bei Nichtverwendung des normalen Konstruktors*
- **`Object execute()`** *immer nötig: stellt die Anfrage*

Grundsatz:

Jede OQLQuery muss einer Transaktion und einer Datenbank zugeordnet sein

Zusammenfassung:
Eine Fragestellung mit OQL in FastObjects

```
db = new Database ();
db.open("FastObjects://LOCAL/ToolsBase", Database.OPEN_READ_WRITE );
Transaction txn = new Transaction( db );
txn.begin();
// Frage formulieren:
String queryString = "SELECT c FROM ToolkitExtent AS c " +
                                        "WHERE c.year_ = 1997";
// Frageobjekt erzeugen:
OQLQuery query = new OQLQuery( queryString );
// Frage stellen:
Object result = query.execute();
// Ergebnis auswerten . . .
txn.abort();
db.close()
```

Beim nächsten Mal:
Einführung in JDO