objektorientierte Datenbanken

Prof. Dr. U. Hoffmann
FH Wedel

Standard der Object Data Management Group (ODMG)

Gliederung

Technische Ziele für die Objektmodellierung zum Zweck der Datenbankanbindung

- **Standardisierung des Objektmodells**
  OMG, ODMG, UML, Java / C++ / C#
- **Persistenzkonzept**
  Dauerhaftigkeit von Daten
- **Transaktionskonzept**
  Verknüpfung von Operationen zu einer Einheit
- **Anbindung an konkrete Programmiersprachen**
  Berücksichtigung verschiedener Besonderheiten

Warum ist Standardisierung notwendig?

- **Antwort:**
  Wenn Datenbanken Objekte aus verschiedenen Programmen zum Zweck der Manipulation und des Austauschs halten sollen, dann müssen diese Programme dasselbe Verständnis von Objekten und ihren Eigenschaften haben.

- **Datenbanken haben einen de-facto-Standard:**
  Relationales Modell, SQL als DML

- **Objektorientierte Sprachen unterscheiden sich in:**
  - Typkonzept
  - Kapselungskonzept
  - Speicherverwaltung und Speicherzugriff
  - Vererbungskonzept

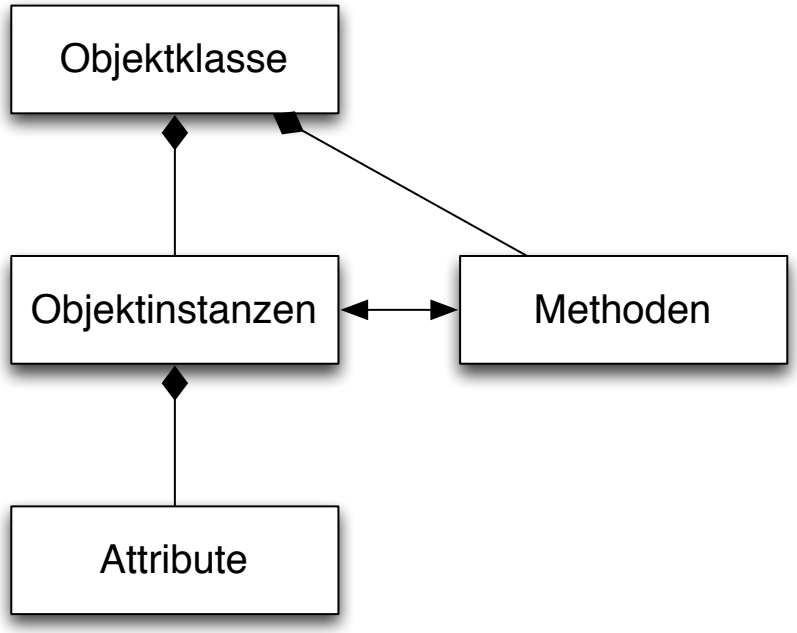
Gemeinsamkeiten aller OOP-Modell

Objekthierarchie und -aufbau:

Deklaration der Atttribute
Deklaration und Festlegung der Methoden

Festlegung des Zustands
definiert durch die Werte der Attribute
Festlegung des Methodenparameters **self**

Festlegung der Werte



Klassen können spezialisiert und generalisiert werden (Vererbung)

Geschichte der ODMG

ODMG

- gegründet 1991 als unabhängige Untergruppe der OMG
- Mitglieder: 90% des damaligen OODB-Marktes
- Leitung: Rick Cattell (Sun, später SunSoft, später JavaSoft)

- **Ziele:**
  - Vereinheitlichung der Smalltalk–Welt und C++-Welt durch einen einheitlichen Objektstandard
  - Objektmodell: ausgehend vom OMG-Modell
  - Datendefinitionssprache: (Object Definition Language, ODL)
  - Anbindung an die Sprachen Smalltalk und C++ (ab 1997 auch Java)
  - Persistenzkonzepte
  - Transaktionskonzepte
  - Abfragekonzepte (Object Query Language, OQL)

- 2001 verabschiedet *The Object Data Standard: ODMG 3.0*
- 2001 aufgelöst *Für Java entstand parallel der JDO–Standard*
- 2006 OMG bereitet einen 4. Objekt–Datenbank–Standard vor

Das ODMG-Objektmodell

- **Objekte** Unterscheidung: atomar oder strukturiert
- **Literale** Werte (Unterscheidung: atomar oder strukturiert)
- **Eigenschaften** Unterscheidung: Attribute oder Beziehungen
- **Verhalten** Methoden (Unterscheidung: Schnittstelle oder Implementierung)
- **Typen** Oberbegriff für Objekte mit gemeinsamen Eigenschaften und Verhalten (Schnittstelle mit Implementierungen)
- **Interface** Schnittstelle ohne Implementierungen
- **Klasse** Implementierung eines Typs
- **Extent** Menge aller Instanzen eines Typs bzw. einer Klasse und der Subtypen (Unterklassen)
  Beachte: Eine Instanz gehört eindeutig zu einem Typ (Klasse).
- **Schlüssel** zum eindeutigen Identifizieren von Objekten

Das ODMG-Objektmodell

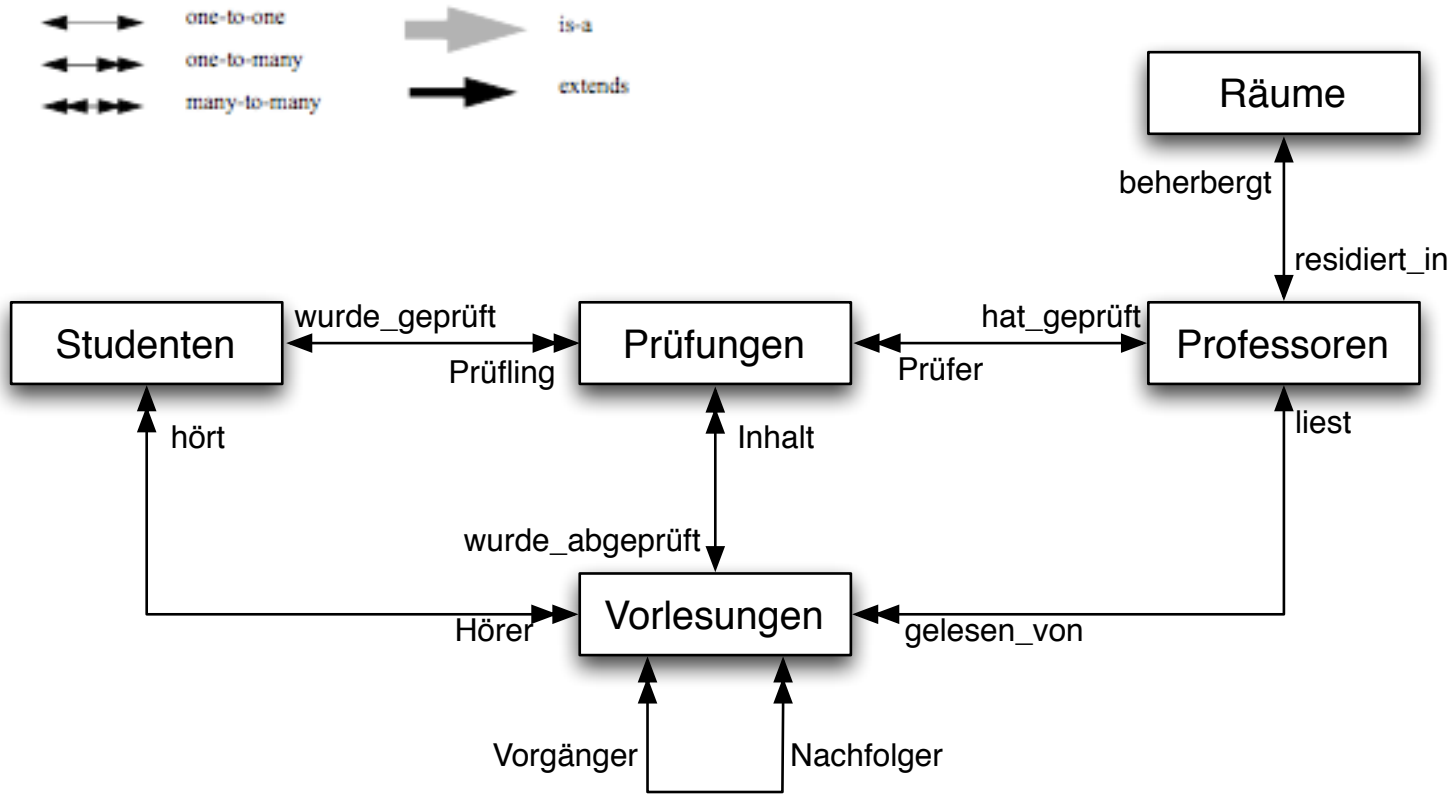
Konzept	ODMG
Vererbungskonzept	Unterklassen erben alle Methoden, Attribute und Beziehungen Überlagerung von Methoden bei Vererbung ist verboten Mehrfachvererbung nur bei Interfaces, nicht bei Klassen
Kapselungskonzept	Kapselung wird nicht beachtet: Alles ist öffentlich
Speicherverwaltung und Speicherzugriff	ist Sache der Programmiersprache

Das ODMG-Objektmodell

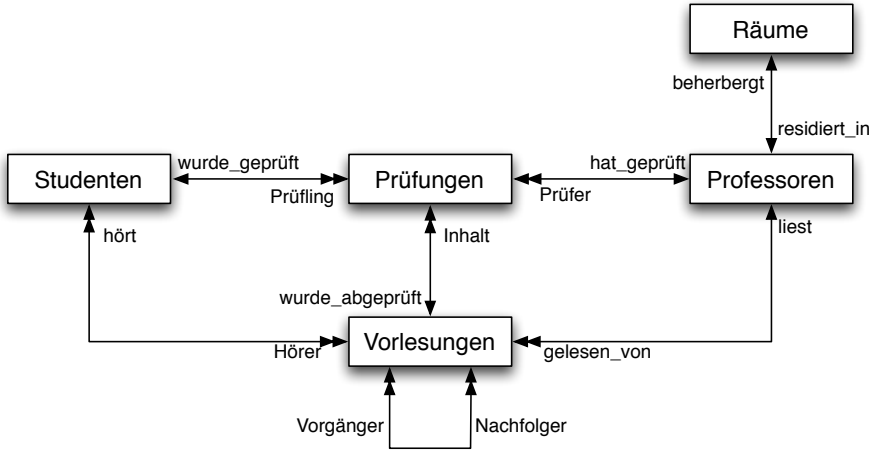
- **Vordefinierte Typen für die Objekte**:
  - keine vorgeschriebenen atomaren Typen (wird der Implementierung überlassen)
  - Collection Types: Set, Bag, List, Array, Dictionary (Typen der Elemente müssen gleich sein und explizit spezifiziert werden)
  - Structured Types: Date, Interval, Time, TimeStamp
- **Vordefinierte Typen für die Literale**:
  - atomare Typen: sehr viele (alle von Java und noch mehr)
  - Collection Types: Set, Bag, List, Array, Dictionary
  - Structured Types: Date, Interval, Time, TimeStamp

Unterscheidung zwischen Objekt und Literal tritt nur in C++ hervor.

ODMG–Beispiel: Modell



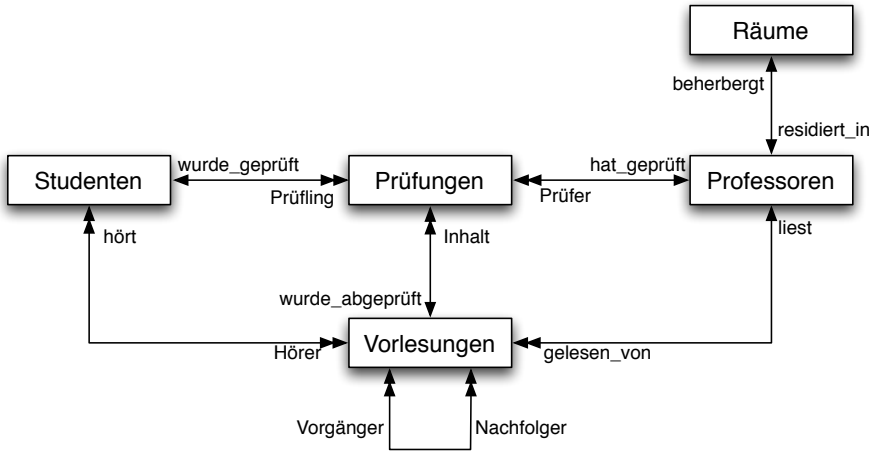
ODMG–Beispiel: Object Definition Language




```
class Professoren {   // ODL
  attribute long PersNr;
  attribute string Name;
  attribute string Rang;
  relationship Raeume residiert_in inverse Raeume::beherbergt;
  relationship set(Vorlesungen) liest inverse Vorlesungen::gelesen_von;
  relationship set(Pruefungen) hat_geprueft inverse Pruefungen::Pruefer;
};

class Vorlesungen {
  attribute long VorlNr;
  attribute string Titel;
  attribute short SWS;
  relationship Professoren gelesen_von inverse Professoren::liest;
  relationship set(Studenten) Hoerer inverse Studenten::hoert;
  relationship set(Vorlesungen) Nachfolger inverse Vorlesungen::Vorgaenger;
  relationship set(Vorlesungen) Vorgaenger inverse Vorlesungen::Nachfolger;
  relationship set(Pruefungen) wurde_abgeprueft inverse Pruefungen::Inhalt;
};
```

ODMG–Beispiel: Object Definition Language




```
class Studenten { // ODL
  attribute string Name;
  attribute boolean female;
  attribute int Fachsemester;
  relationship set(Vorlesungen) hoert inverse Vorlesungen::Hoerer;
  relationship set(Pruefungen) wurde_geprueft inverse Pruefungen::Pruefling;
};

class Pruefungen {
  attribute struct Datum
  { short Tag; short Monat; short Jahr} Pruefdatum;
  attribute float Note;
  relationship Professoren Pruefer inverse Professoren::hat_geprueft;
  relationship Studenten Pruefling inverse Studenten::wurde_geprueft;
  relationship Vorlesungen Inhalt inverse Vorlesungen::wurde_abgeprueft;
};

class Raeume {
  attribute string RaumId;
  relationship Professoren beherbergt inverse Professoren::residiert_in;
}
```

Was ist eine Integritätsbedingung?

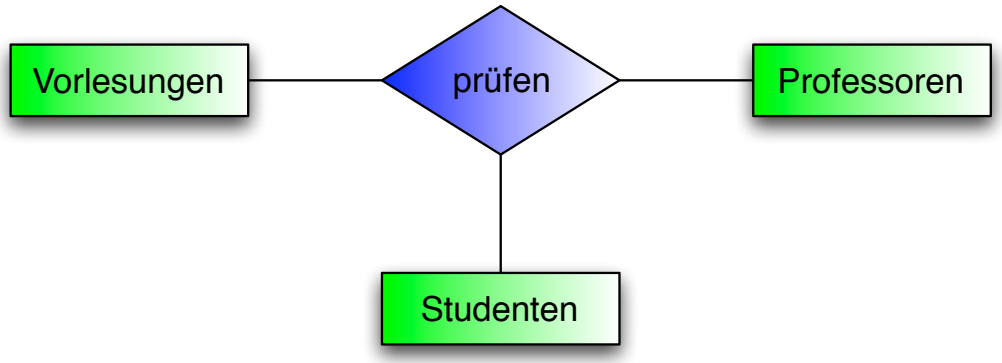
Eine Integritätsbedingung ist eine beliebige logische Bedingung, die zwischen den betrachteten Daten gelten muss.

- **statische Integritätsbedingung** (Invariante)
  Daten müssen Bedingung zu jeder Zeit erfüllen.
- **dynamische Integritätsbedingung**
  Bedingungen werde an die Änderung gestellt.
- *weiche* Bedingung:
  Zulässige Daten hängen vom Zustand der zuletzt angenommenen Datenbelegung ab.
- *scharfe* Bedingung:
  Zulässige Daten hängen von der Entwicklung der in der Vergangenheit angenommenen Zustände ab.

In OOP können Integritätsbedingungen gewährleistet werden durch:

- Typvereinbarungen oder
- Implementierung entsprechender Manipulationsmethoden

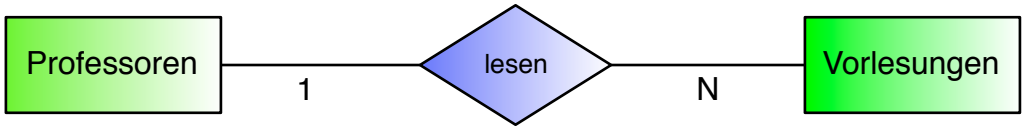
Beispiel — Integritätsbedingung



Typvereinbarungen legen fest, ob Studenten von Professoren oder Assistenten geprüft werden:

```
class Pruefungen { // ODL
  attribute Datum Pruefdatum; // ist Klasse, aber dennoch Attribut
  attribute float Note;
  relationship Professoren Pruefer inverse  Professoren::hat_geprueft;
  relationship Studenten Pruefling inverse  Studenten::wurde_geprueft;
  relationship Vorlesungen Inhalt  inverse  Vorlesungen::wurde_abgeprueft;
};
```

Beispiel — statische Integritätsbedingung



Der Set-Typ sorgt dafür, dass ein Professor mehrere Vorlesungen halten kann.

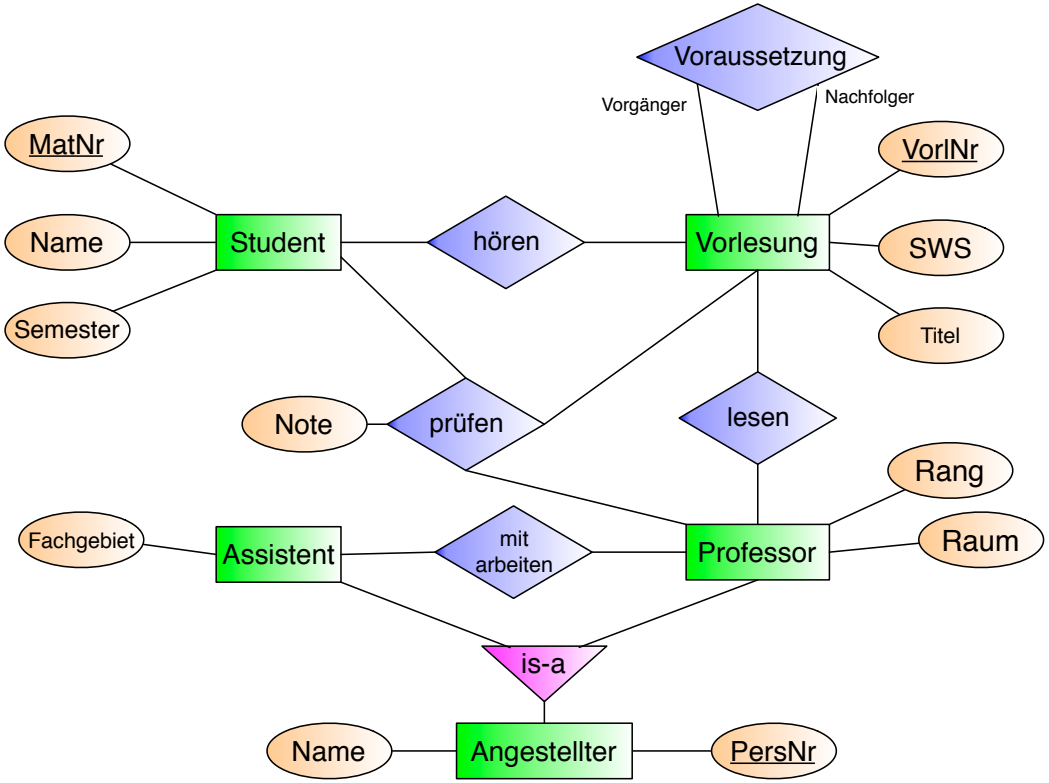
Die Zugriffsmethode sorgt dafür, dass ein Professor nicht mehr als N Vorlesungen hält:

```
class Professoren { // ODL
  ...
  relationship set(Vorlesungen) liest inverse Vorlesungen::gelesen_von;
  ...
}

class Professoren { // Java
   ...
  boolean attachVorlesung (Vorlesung vorlesung)
  {
     if ((Vorlesungen.size() >= N) &&
          !(Vorlesungen.contains (vorlesung)))
        return false;
     else
     {
        Vorlesungen.insert (vorlesung);
        vorlesung.gelesen_von (this);
        return true;
     }
  }
  ...
};
```

Beispiel – dynamische Integritätsbedingung

Semesterzahl von Studenten wächst monoton.



Weiche dynamische Bedingung:

erfordert nur Betrachtung eines Objekts

Dynamische Integritätsbedingungen: Weitere Beispiele

- Für die Mitarbeiter der Testabteilung gilt, dass das Durchschnittsgehalt unter dem Durchschnittsgehalt aller Projektleiter liegen muss.
  - *Weiche* dynamische Bedingung:
    erfordert aber die Betrachtung vieler Objekte

- Der Durchschnittsverkaufspreis eines Produktes bezogen auf die letzten zwölf Monate darf nicht mehr als fünf Prozent vom Durchschnittspreis der letzten beiden Jahre abweichen.
  - *Scharfe* dynamische Bedingung:
    erfordert die Analyse (und damit die Abspeicherung) vergangener Zustände

Was ist Persistenz?

- **Persistenz** ist die Fähigkeit von Daten, beliebig lange Lebensdauern anzunehmen unabhängig von dem Programm, in dem sie erzeugt wurden.
- Nicht persistente Daten heißen **transient**:
  Transiente Daten leben nur so lange wie das Programm (der Programmteil), in dem sie erzeugt wurden.

- Die Funktionsweise von Programmen darf sich nicht ändern, wenn das Programm mit persistenten statt mit transienten Daten arbeitet. *(Persistenz ist* **Programm-orthogonal***)*

- Persistenz muss prinzipiell mit jedem Typ funktionieren
  (Persistenz ist **Typ-orthogonal***)*

Persistenzkonzepte der ODMG

- **klassenabhängige Persistenz**
  - Jede Klasse kann persistent gemacht werden.
  - Alle Instanzen einer persistenten Klasse sind persistent.

ODMG:

- **objektabhängige Persistenz**
  Jedes Objekt einer persistenzfähigen Klasse kann persistent gemacht werden.

  Die Persistenz eines Objekts erfolgt durch explizite Kennzeichnung.

Smalltalk	Automatische Persistenzfähigkeit
C++	Persistenzfähigkeit durch Vererbung
Java	Persistenzfähigkeit durch explizite Kennzeichnung

Persistenzkonzepte der ODMG

Objekte mit Beziehungen zu anderen Objekten:

C++: **Explizite Persistenz**

Persistenz eines Objekts erfolgt nur durch explizite Kennzeichnung.

Smalltalk, Java: **Transitive Persistenz**

Auch Persistenz durch Erreichbarkeit genannt: Jedes Objekt, das durch ein persistentes Objekt erreicht werden kann, ist ebenfalls persistent, sofern es einer persistenzfähigen Klasse angehört (anderenfalls ist es transient)

Persistenz in DB4O

- DB4O besitzt objektabhängige Persistenz
- keine explizite Markierung persistenter Objekte
- CRUD Schnittstelle (Create-Read-Update-Delete) `store`, Queries, `store`, `Delete`
- Transitive Persistenz mit steuerbarer Tiefe (beim Schreiben und Löschen)

Warum brauchen wir ein Transaktionskonzept?

Bsp.: Bankkontoführung

Prozess 1: Umbuchung eines Betrages von Konto A nach Konto B

geplant:

Umbuchung
```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
b1 := b1 + 300
write (B, b1)
```

tatsächlicher Verlauf:

Umbuchung
```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
```
Störung

Wo sind die 300 Euro geblieben?

Warum brauchen wir ein Transaktionskonzept?

Bsp.: Bankkontoführung

Prozess 1: Umbuchung eines Betrages von Konto A nach Konto B

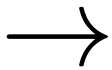
Prozess 2: Zinsgutschrift für Konto A

Umbuchung

```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
b1 := b1 + 300
write (B, b1)
```

Zinsgutschrift

```
read (A, a2)
a2 := a2 *
1.03
write (A, a2)
```



Möglicher verzahnter Ablauf:

Umbuchung	Zinsgutschrift
`read (A, a1)`	
`a1 := a1 - 300`	
	`read (A, a2)`
	`a2 := a2 * 1.03`
	`write (A, a2)`
`write (A, a1)`	
`read (B, b1)`	
`b1 := b1 + 300`	
`write (B, b1)`	

Wo ist die Zinsgutschrift geblieben?

Transaktionskonzept der ODMG

- **Was ist eine Transaktion?**
  Eine **Transaktion** ist eine Folge von Operationen, die entweder alle vollständig oder alle überhaupt nicht durchgeführt werden sollen.

Eine Transaktion muss das ACID-Prinzip erfüllen:

- **Atomicity** unteilbar
- **Consistency** hinterlässt immer konsistenten Datenbestand
- **Isolation** beeinflusst keine andere Transaktion
- **Durability** Wirkung ist permanent

Transaktionskonzept der ODMG

- Operationen von Transaktionen:

begin()	Anfang einer Transaktion
commit()	Ende einer Transaktion (erfolgreich)
abort()	Ende einer Transaktion (nicht erfolgreich)

- Alle zwischen Anfang und Ende einer Transaktion angetroffenen Operationen gehören zur selben Transaktion.
- Alle Operationen, die sich auf die Datenbank auswirken sollen, müssen innerhalb einer Transaktion ausgeführt werden.
- Eine Transaktion kann nur ausgeführt werden, wenn eine Datenbank geöffnet worden ist.

Transaktionen in DB4O

- Jede Operation findet im Kontext einer Transaktion statt.
- Wird implizit durch Öffnen der Datenbank (`openFile`) erzeugt.
- `db.commit()`, `db.rollback()`
- Transaktionen sind nicht vollständig voneinander isoliert: Isolationsgrad READ COMMITED: Mit `commit` werden Änderungen in anderen Transaktionen sichtbar.
- **Achtung!**
  Transiente Objekte werden nicht automatisch aktualisiert. Sie müssen explizit neu geladen werden (`refresh`)

Example (db4o Transaktionen)

```
Student student = new Student(12345, "John", "Deo");
ObjectContainer db = Db4o.openFile("trans.yap");
db.store(student);
db.commit();  // Student ist gespeichert
student.setLastName("Doe-Dalton");
db.store(student);
db.rollback();
// Keine Aenderung in der DB,
// transientes und persistentes Objekt inkonsistent
db.ext().refresh(student, Integer.MAX_VALUE);
```

Quelle: *db4o*-Tutorial und *db4o* Römer/Visengeriyeva, entwickler.press, 2007

Ausblick

Einführung
- Standardisierung

ODMG
- Objektmodell
- Integritätsbedingungen
- Persistenzkonzept
- Transaktionskonzept

Wie sehen die ODMG–Nachfolger aus?

- Fragen?