

Objektorientierte Datenbanken

Vorlesung 2
Sebastian Iwanowski
FH Wedel

Offene Frage vom letzten Mal:

***Wie können wir objektorientiert modellieren
und dennoch eine Datenbank benutzen ?***

Technische Ziele für die Objektmodellierung zum Zweck der Datenbankanbindung

Standardisierung des Objektmodells

OMG, ODMG, Java

Persistenzkonzept

Dauerhaftigkeit von Daten

Transaktionskonzept

Verknüpfung von Operationen zu einer Einheit

Anbindung an konkrete Programmiersprachen

Berücksichtigung verschiedener Besonderheiten

Hier wird nur auf
Java eingegangen

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:

Vererbungskonzept

Typkonzept

Kapselungskonzept

Speicherverwaltung und Speicherzugriff

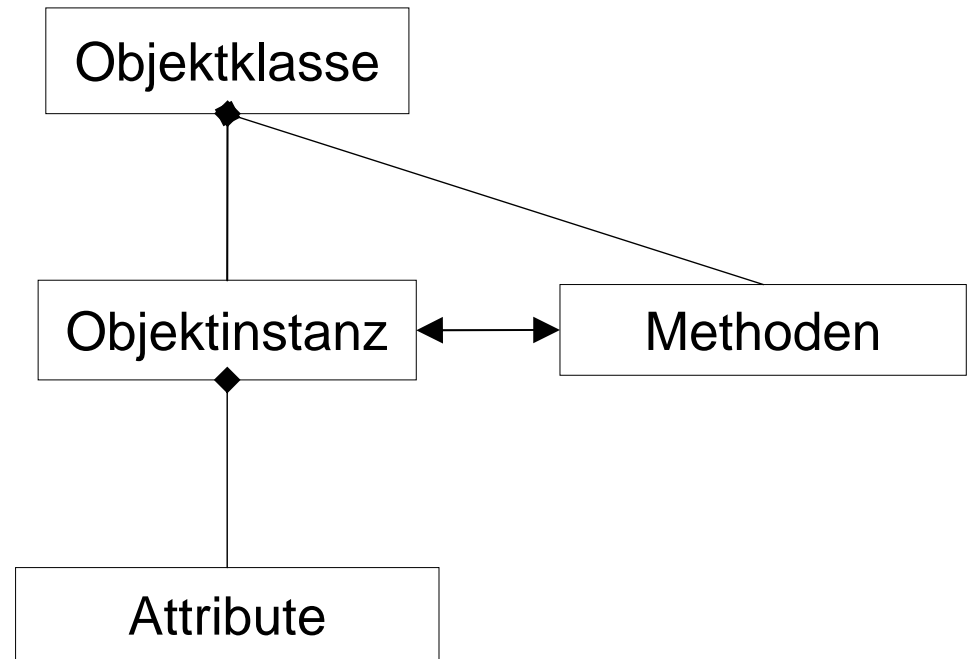
Gemeinsamkeiten aller OOP

Objekthierarchie und -aufbau:

Deklaration der Attribute
Deklaration und Festlegung der Methoden

Festlegung des Zustands:
definiert durch Werte der Attribute
Festlegung des Methodenparameters self

Festlegung der Werte



Klassen können spezialisiert und generalisiert werden (Vererbung)

Geschichte der 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 Smalltalkwelt und C++-Welt durch Objektstandard

Anbindung an die Sprachen Smalltalk und C++ (ab 1997 auch Java)

Persistenzkonzepte

Transaktionskonzepte

Abfragekonzepte

Das tat ab 1995 auch Java !

2001 aufgelöst

parallel entwickelter Standard: JDO (nur für Java)

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	<p>Unterklassen erben alle Methoden, Attribute und Beziehungen</p> <p>Überlagerung von Methoden bei Vererbung ist verboten</p> <p>Mehrfachvererbung nur bei Interfaces, nicht bei Klassen</p>
Kapselungskonzept	<p>Kapselung wird nicht beachtet: Alles ist öffentlich !</p>
Speicherverwaltung und Speicherzugriff	<p>ist Sache der Programmiersprache</p>

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 nur für nicht objektorientierten Teil der C++ - Implementierung

Das ODMG-Objektmodell

Objektspezifikationssprachen:

Object Definition Language (ODL)

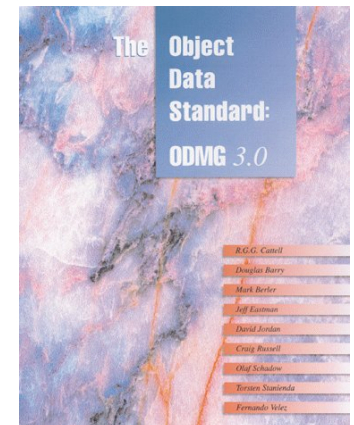
(zum Definieren von Objekten)

Object Interchange Format (OIF)

(zum Speichern und Laden von Objekten mit ihren gegenwärtigen Zuständen)

Ausführliche Syntax und Beispiele in Kap. 3 von:

Richard G.G. Cattell / Douglas K. Barry (Edts.): *The Object Data Standard: ODMG 3.0*



Persistenzkonzepte der ODMG

Persistenzkonzepte der ODMG

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

1) Automatische Persistenzfähigkeit

C++

2) Persistenzfähigkeit durch Vererbung

Java

3) Persistenzfähigkeit durch explizite Kennzeichnung

Persistenzkonzepte der ODMG

Objekte mit Beziehungen zu anderen Objekten:

C++

1) Explizite Persistenz

Persistenz eines Objekts erfolgt nur durch explizite Kennzeichnung.

**Smalltalk
Java**

2) 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)

Das Transaktionskonzept der ODMG

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 € geblieben?

Warum brauchen wir ein Transaktionskonzept ?

Bsp.: Bankkontoführung

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

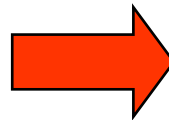
Prozess 2: Zinsgutschrift für Konto A

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

Interface Transaction:

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.*

Datenbanken werden durch Objekte des Interfaces Database repräsentiert

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;
```

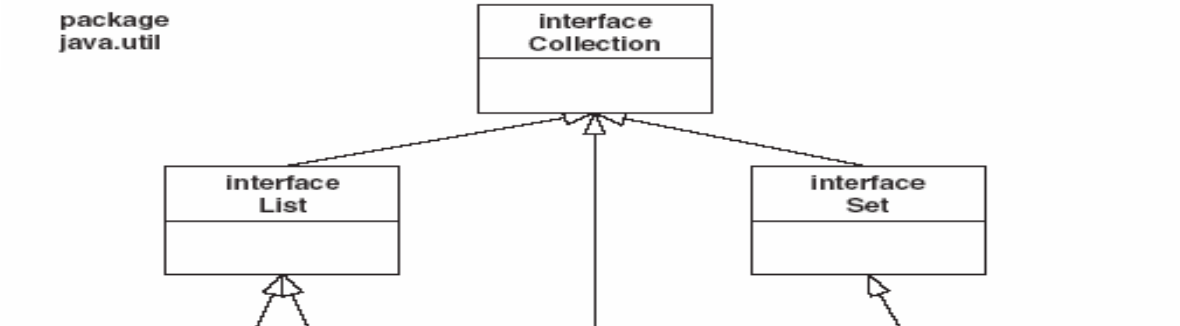
```
public void makePersistent(Object object);
```

```
public void deletePersistent(Object object);
```

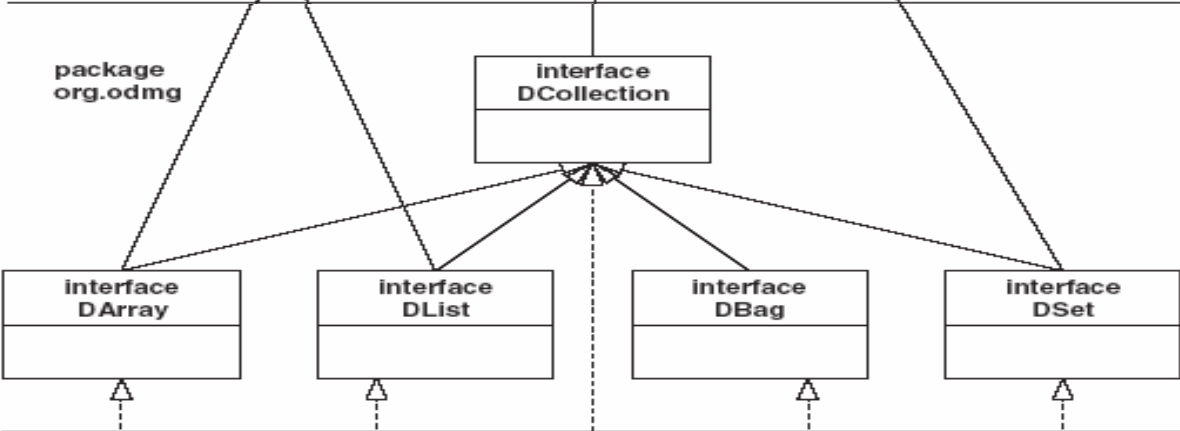
→ In FastObjects: Es gibt Klassen gleichen Namens

Collections und Sets

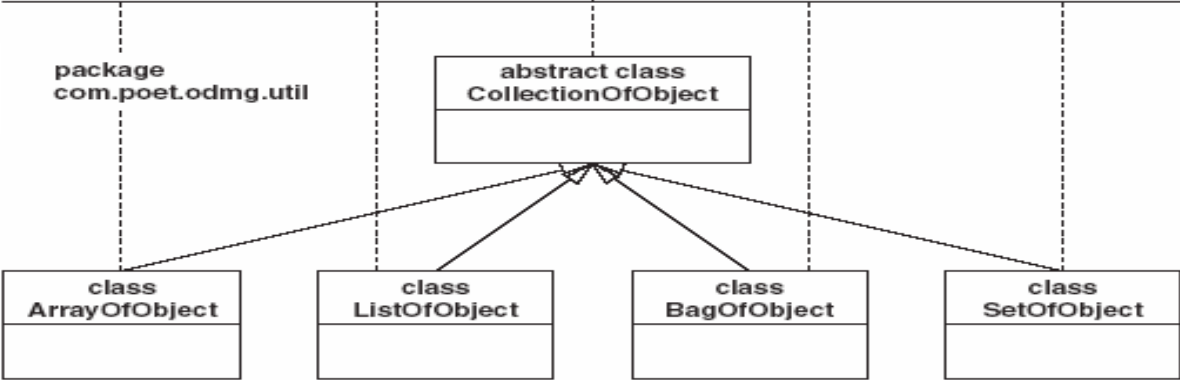
Java:



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**

ab dieser Woche:

***Testen Sie selbst den ODMG-Standard
life
in den Übungen***