Konzepte der Datenbanktechnologie

Prof. Dr. U. Hoffmann
FH Wedel

Data Base for Objects
(DB4O Teil 2)

das letzte mal

- DB4O
  - Objekte im Speicher und in der Datenbank
  - Identitätskonzept
  - Integritätsbedingungen
  - Persistenzkonzept
  - Transaktionskonzept

Weitere DB4O–Themen?

Gliederung

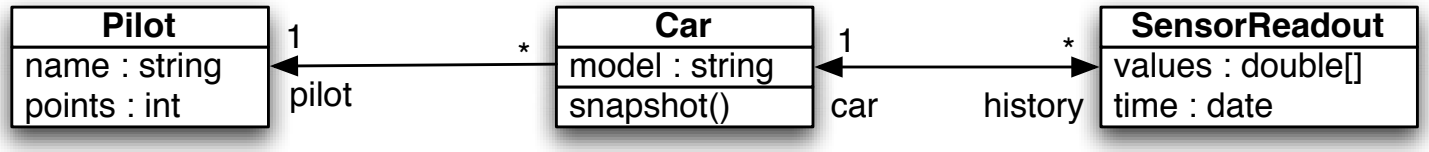
Arrays und Collections

Vererbung

Transparente Aktivierung

Client–Server–Betrieb

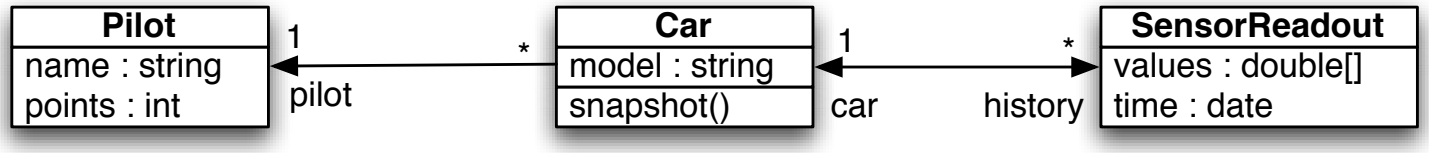
Arrays und Collections



Implementierung in Java:

- 1:n–Beziehung zwischen `Pilot` und `Car` in eine Richtung navigierbar:
  - Instanzvariable `pilot` in `Car`
  - Navigation von `Pilot` zu `Car` durch Datenbankabfragen/Suchen
- 1:n–Beziehung zwischen `Car` und `SensorReadout` in beide Richtungen navigierbar:
  - Array– oder Listen–Instanzvariable `history` in `Car`
  - Instanzvariable `car` in `SensorReadout`

Collections

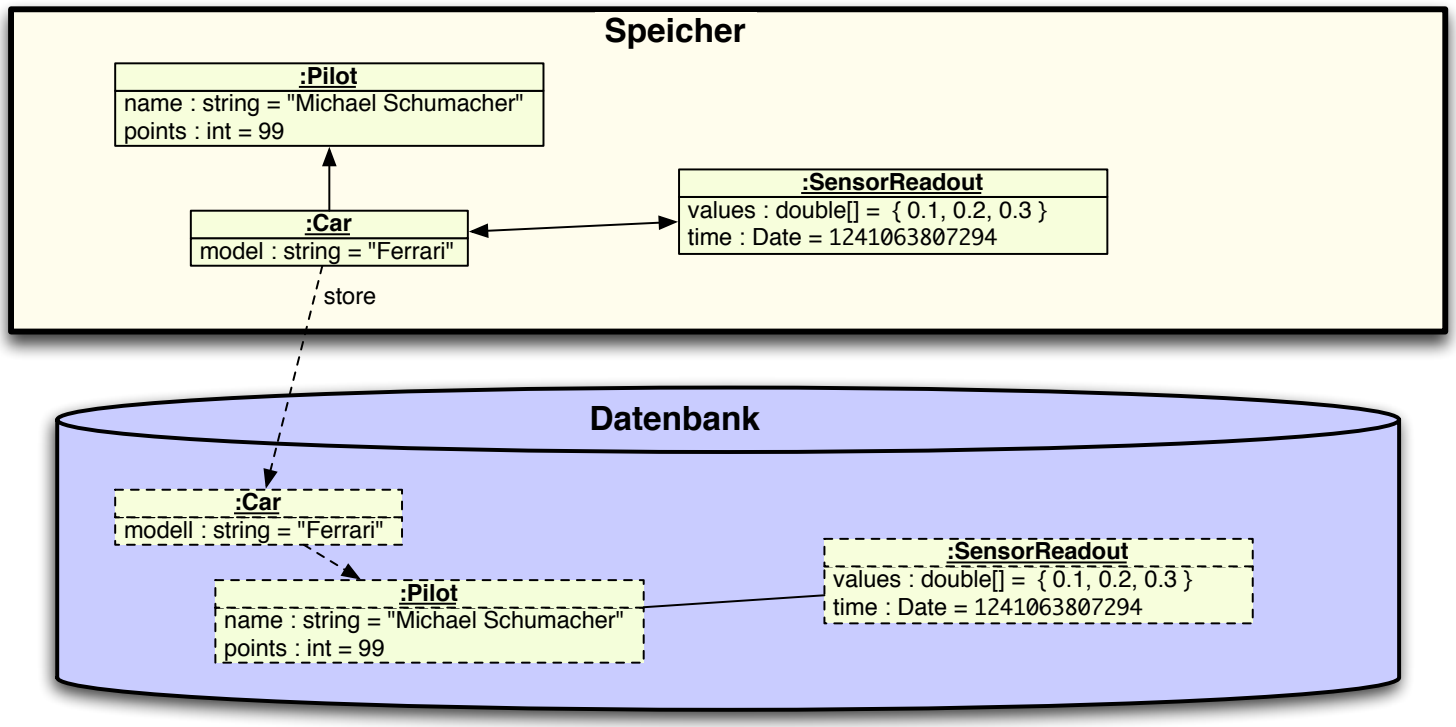


Beispiel (Erweiterung Car und SensorReadout)

```
public class Car {
    ...
    private List<SensorReadout> history
                    = new ArrayList<SensorReadout>();
    ...
}
public class SensorReadout {
    ...
    private Car car;
    ...
}
```

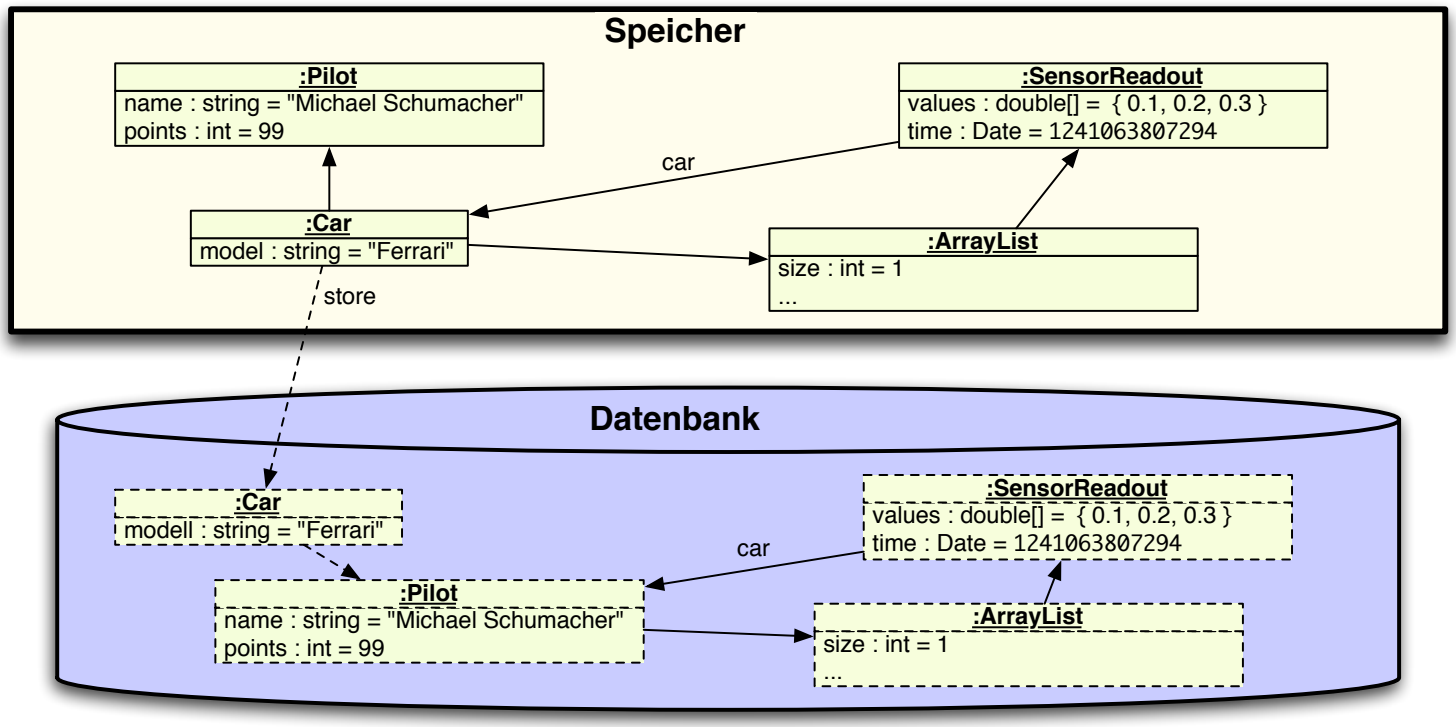
- Collection–Objekte sind *normale* Objekte:
- Aktivierung, Aktualisierung, Löschen, Anfragen:
  wie bei selbstdefinierten Klassen
- Tiefen–Einstellungen beachten!

Collections



- Repräsentation der 1:n–Beziehungen im Speicher und der Datenbank
- Zu abstrakt - wie ist die Navigierbarkeit realisiert?

Collections



- Repräsentation der 1:n–Beziehungen im Speicher und der Datenbank
- explizite Darstellung des Listen–Objekts

DB4O–Datenbankanfragen nach Collections

- Collection–Objekte sind *normale* Objekte

Beispiel (Anfrage nach `List`–Objekten)

```
List result = db.queryByExample(List.class);
```

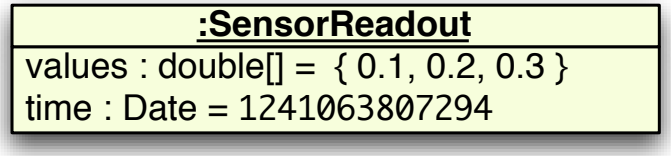
oder

```
List result = db.queryByExample(new ArrayList());
```

- erfragt alle `List` bzw. `ArrayList`–Objekte in der Datenbank

- native und SODA-Anfragen entsprechend

DB4O–Datenbankanfragen mit Arrays



- Fragen nach SensorReadout–Objekten

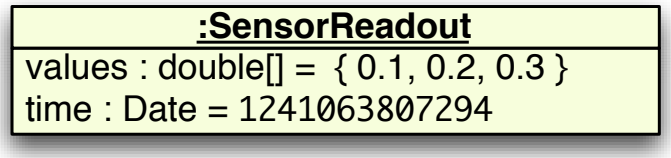
Beispiel (QBE mit Array)

```
SensorReadout proto = new SensorReadout(
        new double[]{0.3, 0.2},
        null,  // time
        null); // car

List<SensorReadout> readouts = db.queryByExample(proto);
```

- Es wird bei Array–Objekten *nicht* nach einem gleichen Array gesucht!
- Prototyp–Array enthält lediglich die Werte, die in den anzufragenden Arrays mindestens enthalten sein sollen.
- Reihenfolge der Elemente in Prototyp-Arrays ist irrelevant.

DB4O–Datenbankanfagen nach Arrays



- Fragen nach Array–Objekten

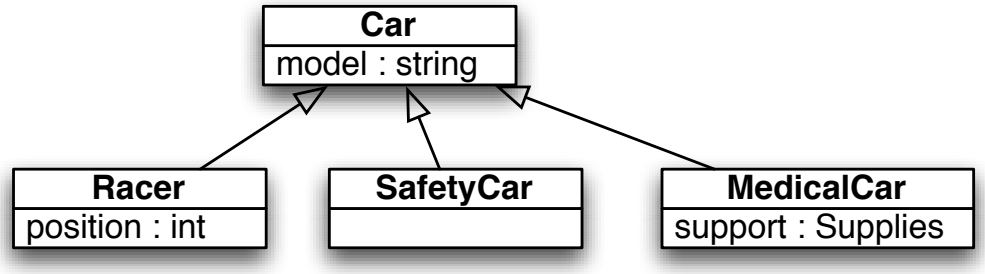
Beispiel (QBE nach Array)

```
List result = db.queryByExample(new double[]{0.1,0.2});

// result ist leer!
```

- Array–Objekten sind *keine normalen* Objekte
- Sie werden nicht eigenständig in der Datenbank abgelegt.

Vererbung



- Vererbungsstrukturen werden direkt in die Datenbank übernommen
- Anfragen nach Basis–Klasse liefert alle Objekte der Basis–Klasse und aller abgeleiteten Klassen.

Beispiel (SODA–Anfrage nach Rennwagen und Unterstützungsfahrzeugen)

```
Query query=db.query();
query.constrain(Car.class);
ObjectSet result=query.execute();
listResult(result);
```

- Anlegen, Aktualisieren, Löschen unabhängig von der Vererbung

Transparente Aktivierung

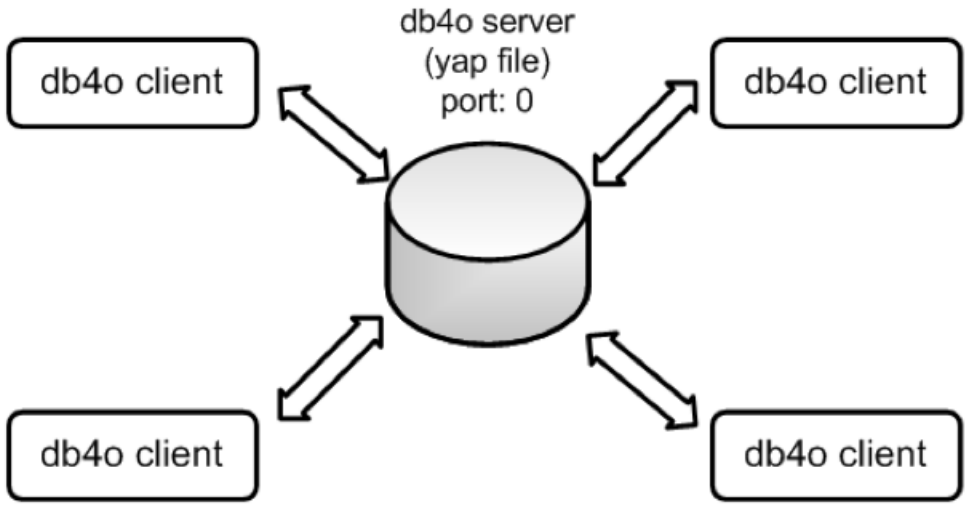
- DB4O aktiviert Objekte gemäß der Aktivierungstiefe
- vollständige Objekt–Aktivierung durch transparente Aktivierung
  - Nachteil: unnötige Objekte werden aktiviert
  - Lösung: Steuerung der Aktivierung bei Bedarf durch `Activatable`–Interface
  - Methoden `activate` und `bind`
  - Nachteil: Abhängigkeit von DB4O in Objekten
  - Lösung: Enhancement

Beispiel (transparente Aktivierung)

```
public class Car implements Activatable {
private transient Activator _activator;
...
public void activate(ActivationPurpose purpose) {
  if (_activator != null) _activator.activate(purpose);
}
public void bind(Activator activator) {
  if (_activator == activator) return;
  if (activator != null && _activator != null) {
     throw new IllegalStateException(); }
  _activator = activator;
}   }
```

Client–Server–Betrieb

- bisher: Betrieb im Solo–Modus:
  nur ein Programm greift auf die Datenbank zu
- Client–Server–Betrieb:
  mehrere Clients greifen auf die gleiche Datenbank zu
  - Embedded–Server:
    Server und alle Clients in einem Prozess
    DB4O startet Server „auf Port 0“



  - Verteilter Betrieb im Netzwerk–Modus:
    Server und Client sind in unterschiedlichen Prozessen

Quelle: Römer/Visengeriyeva: *db4o schnell+kompakt*

Client–Server–Betrieb

- Embedded–Server:
  Server und alle Clients in einem Prozess
  DB4O startet Server „auf Port 0“

Beispiel (embedded–Server/Client)

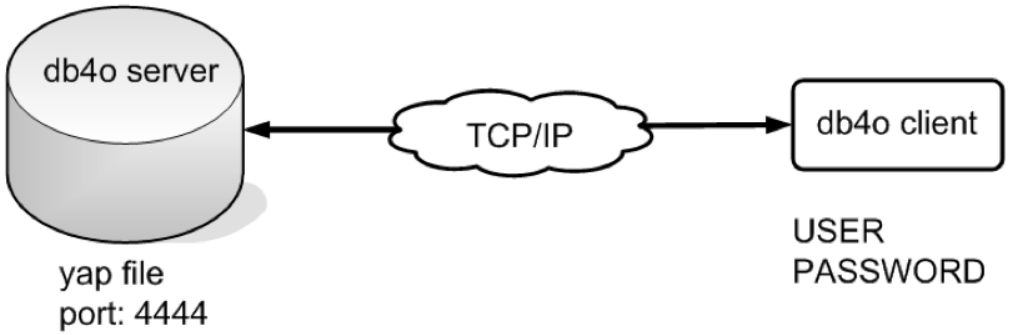
```
Student student1= new Student(12345,"John","Doe");
Student student2= new Student(12346,"Jill","Doe");
ObjectServer server= Db4o.openServer("trans.yap",0);
ObjectContainer client1=server.openClient();
ObjectContainer client2=server.openClient();

client1.store(student1); // client1:1/client2:0
client2.store(student2); // 1/1
client1.commit();        // 1/2
client2.rollback();      // 1/1
client2.store(student2); // 1/2
client2.commit();        // 2/2
```

Kommentar gibt die Anzahl der Studentenobjekte in der Datenbank aus Sicht der Transaktion 1 bzw. 2.

Verteilter Client–Server–Betrieb

- Verteilter Betrieb im Netzwerk–Modus:
  Server und Client sind in unterschiedlichen Prozessen



- Authentifizierung bei Netzwerk–Zugriffen zwingend.

Quelle: Römer/Visengeriyeva: *db4o schnell+kompakt*

Verteilter Client–Server–Betrieb

- Verteilter Betrieb im Netzwerk–Modus:
  Server und Client sind in unterschiedlichen Prozessen

Beispiel (Netzwerk–Server)

```
ObjectServer server = Db4o.openServer("db.yap",0xdb40);
server.grantAccess("user", "password");
...
server.close();
```

Beispiel (Netzwerk–Client)

```
ObjectContainer client = Db4o.openClient(
    "localhost",0xdb40, "user", "password");
    // mit DB arbeiten
client.close();
```

Ausblick

Weitere DB4O–Konzepte (teilweise in den Übungen)

- Callbacks: Informationen über DB4O-Aktionen
- Enhancement: Automatische Transparente Aktivierung
- Replikation: Abgleich zwischen Datenbanken

- Fragen?