1.Einleitung:

Bei der Arbeit mit objektorientierten Programmiersprachen stellt sich häufig die Frage wie Objekte persistiert werden können. Da vor allem relationale Datenbanken sich großer Beliebtheit erfreuen, lag der Schluss nahe Objekte auf Tabellenschemen abzubilden. Dieses Vorgehen führte zu objektrelationen Datenbanksystemen, die jedoch einen großen Nachteil mit sich bringen: das Mapping zwischen der objektorientierten und der relationalen Welt.
Um dieses Problem zu umgehen entstanden die Objektdatenbanken wie zum Beispiel db4o.

db4o wurde im Jahr 2000 durch Carl Rosenberger entwickelt und 2001 veröffentlicht. 2004 wurde es durch die Db4objects Inc. übernommen und kommerzialisiert. 2008 erhielt die Versant Corporation die Rechte an db4o und vertreibt seit dem das kostenpflichtige Produkt Versant Object Database und unter freier Lizenz db4o.
db4o liegt aktuell in Version 8.0 vor, die 2011 veröffentlicht wurde.

2.Basiskonzepte

Der wohl größte Unterschied zu objektrelationalen Datenbanksystemen besteht darin, dass die Objekte nicht in Tabellen abgelegt werden und somit kein Mapping zwischen den beiden Welten nötig ist. Die Speicherung erfolgt über die Serialisierung von Objekten (siehe Java ObjectOutputStream/ ObjectInputStream) was dazu führt, dass die Objekte im Binärformat abgelegt werden und somit nicht ohne Weiteres lesbar sind, wie es bei Tabellen der Fall ist.

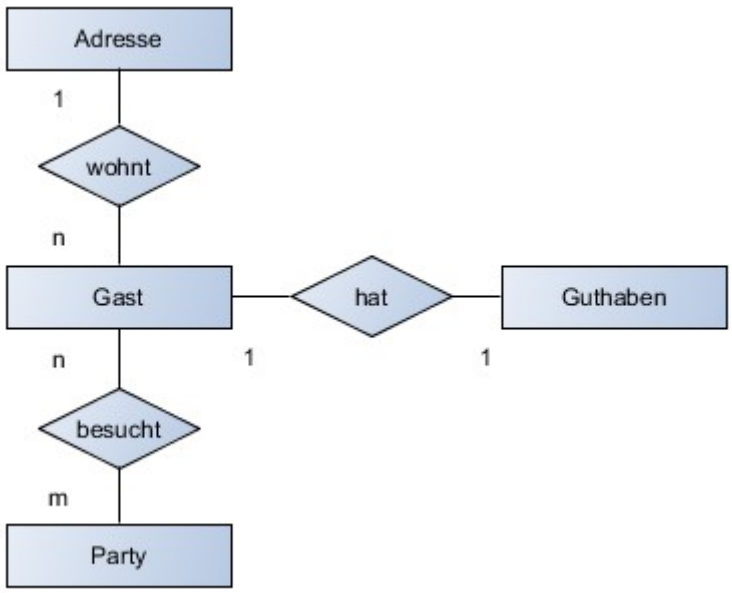
Die folgende Tabelle stellt die Grundbegriffe der objektorientierten und der relationalen „Welt“ gegenüber:

OODB	RDB	Gemeinsamkeiten	Unterschiede
Klasse	Tabelle	Definition einer Datenstruktur	- Klasse kann auch Methoden definieren - Tabelle nur statische Daten
Objektinstanz	Tupel	Repräsentation von Daten	- Objekt kann Daten mit verschiedenen Sichtbarkeiten und Referenzen zu anderen Objekten halten - Tupel nur Systemdatentypen
Attribut	Spalte	Festlegung eines Feldes der Datenstruktur	- RDB hat eine feste Menge von Datentypen - OODB kann beliebigen Datentypen enthalten
Methode	Prozedur	Festlegung eines bestimmten Verhaltens	- Methode gehört zu einer bestimmten Klasse - Prozedur ist ein eigenes Objekt
Identität	Schlüssel	Identifikation eines Objektes/Tupel	- Objektidentität (==) unabhängig vom Inhalt - Einzigartigkeit von Werten
Referenz	Relation	Stellt Verbindung zwischen verschieden Objekten/Tabellen dar	- OODB Objektreferenzen - RDB Fremdschlüssel

db4o speichert primitive Datentypen, deren Wrapperklassen, Strings- und Date-Objekte als Attribute direkt in einem Objekt, in denen sie referenziert werden. Alle anderen Datentypen werden als Objekt einzeln abgelegt und nur die Referenz auf diese direkt gespeichert. Intern werden die Objekte über eine eindeutige Objekt-ID unterschieden.

3. Modellierung relationaler Beziehungen

Die Modellierung relationaler Beziehungen in einer objektorientierten „Welt“ soll an folgendem Beispiel illustriert werden:



Die 1:1 Beziehung Gast/ Guthaben könnte als Attribut in beiden Klassen abgebildet werden:

```
public class Guest {
    private Credit credit;

    public Credit getCredit() {
        return credit;
    }

    public void setCredit(Credit credit) {
        this.credit = credit;
    }
}
 public class Credit {

     private Guest guest;

     public Guest getGuest() {
         return guest;
     }

     public void setGuest(Guest guest) {
         this.guest = guest;
     }
 }
```

Die 1:n Beziehung könnte als Liste auf 1-Seite und als Attribut auf der n-Seite abgebildet werden

```java
public class Guest {
    private Address address;

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
}
```

```java
public class Address {
    private List<Guest> guests;

    public List<Guest> getGuests() {
        return guests = guests == null ? new ArrayList<Guest>() : guests;
    }

    public void setGuests(List<Guest> guests) {
        this.guests = guests;
    }

    public void add(Guest guest) {
        if (!this.getGuests().contains(guest)) {
            this.getGuests().add(guest);
        }

    }

    public void remove(Guest guest) {
        if (this.getGuests().contains(guest)) {
            this.getGuests().remove(guest);
        }
    }
}
```

Die n:m Beziehung könnte als Liste auf beiden Seiten abgebildet werden:

```java
public class Guest {
    private List<Party> parties;

    public List<Party> getParties() {
        return parties = parties == null ? new ArrayList<Party>() : parties;
    }

    public void setParties(List<Party> parties) {
        this.parties = parties;
    }

    public void goesTo(Party party) {
        if (!this.getParties().contains(party)) {
            this.getParties().add(party);
        }
    }
}
```

```java
public class Party {
    private List<Guest> guests;

    public List<Guest> getGuests() {
        return guests = guests == null ? new ArrayList<Guest>() : guests;
    }

    public void setGuests(List<Guest> guests) {
        this.guests = guests;
    }

    public void register(Guest guest) {
        if (!this.getGuests().contains(guest)) {
            this.getGuests().add(guest);
        }
    }
}
```

4. CRUD-Operationen

Create, Read, Update, Delete sind Methoden um Objekte zu erstellen, zu lesen, zu manipulieren und zu löschen. Eine beispielhafte Verwendung dieser Methoden ist in dem Beispielprojekt **DbOs** unter /src/main/java/de/htwk/mhantel/examples/dbos/crud/CRUDOperations.java zu finden.

5. Anfragen an die Objektdatenbank

db4o kennt drei verschiedene Wege Anfragen an die bestehende Datenbank zu formulieren. Alle Beispiele befinden sich im Beispielprojekt DbOs unter /src/main/java/de/htwk/mhantel/examples/dbos/party/PartyExample.java

- Query by Example:
  - Erstellung eines Beispielobjektes mit Werten die gesucht werden
  - alle nicht Defaultwerte gehen in die Suche ein
  - Suche nach Defaultwerten nicht möglich
  - Beispiel PartyExample.inviteToAnotherParty()
- Native Query:
  - Anfragen vollständig in der Programmiersprache definierbar und ausführbar
  - alle Vorteile die die Programmiersprache mit sich bringt
  - siehe Predicate<>, Comparator<> Interface
  - Beispiel PartyExample.nativeQuery()
- Simple Object Database Access Query
  - benutzt Strings um Objektattribute zu identifizieren
  - keine Typsicherheit, keine Meldung bei Schreibfehlern etc.
  - bei dynamischen Queries vom Vorteil
  - auch private Attribute zugreifbar
  - arbeitet mit Reflection
  - Suche auf Attribut des Typs Collection führt zu Suche in jedem Element der Collection
  - Beispiel PartyExample.sodaQuery(), PartyExample.anotherSodaQuery()

6. Transaktionen

db4o startet implizit für jedes ObjectContainer-Objekt eine Transaktion. Eine Transaktion wird durch ein commit() oder ein close() beendet, wobei ein commit() sofort eine neue Transaktion startet. Das Abbrechen bzw. Zurücksetzen einer Transaktion erfolgt über rollback().
Transaktion sind in db4o Read-Committed, das heißt eine Änderung an der Datenbank wird nach einem commit() für alle sichtbar. Um Schreibkonflikte zu vermeiden ist ein eigenes Objekt-Locking nötig, wobei db4o Semaphore zur Verfügung stellt (siehe 7. für Beispielverwendung).

7. Client/Server-Modes

db4o kennt drei verschiedene Client/Server-Modes:

- Embedded-Modus:
  - Client/Server befinden sich innerhalb einer virtuellen Maschine
  - kein Login nötig
  - eignet sich für Webanwendungen, Datenbanken für mobile Geräte
- Client/Server-Modus:
  - Client und Server interagieren via TCP/IP
  - Login via Name/Passwort
  - Beispiel /src/main/java/de/htwk/mhantel/examples/dbos/clientserver
  - zu speichernde Klasse muss nicht bekannt sein, da über GenericReflection persistiert wird
- Out-of-Band-Signalling:
  - senden von Signalen vom Client an den Server
  - Beispielsignale close, defragment
  - siehe Interfaces: MessageSender/MessageRecipient

8. Features

Weitere interessante Features in db4o sind unter anderem:

- Callbacks
  - Event/Listener-System um auf Systemereignisse zu reagieren
- Replikationen
- Ladeverhalten
  - große Objektbäume werden nicht vollständig geladen
  - nur bis zu definierbarer Tiefe geladen
  - Objektbäume müssen über activate() nachgeladen werden, liefern sonst null
  - Activatable Interface und weitere Konfiguration bieten Möglichkeit Objekte „lazy“ nachzuladen

9. Zusammenfassung

db4o eignet sich vor allem für mobile Datenbanken wie zum Beispiel auf Smartphones oder Tabletts, da diese mit objektorientierten Sprachen arbeiten und somit ihre Objekte einfach persistieren können. Durch den Einsatz von db4o wird das Mapping zwischen Objekten und Tabellen überflüssig und produziert somit weniger Overhead bei der Speicherung von Daten. Darüber hinaus bieten objektorientierte Datenbanken den Vorteil, dass Entwickler sich nicht mit mehreren Paradigmen auseinandersetzen müssen, wie es zum Beispiel bei SQL der Fall ist. Die db4o Datenbanken ist einfach einzurichten und über einbinden von wenigen Java-Bibliotheken (Jar-Dateien) nutzbar. db4o Datenbanken sind nur von „innen“ durch Code konfigurierbar und sollten eine Größe von 10 Gigabytes nicht überschreiten (siehe db4o Versant: Reference Documentation/Evaluation Guide/Scalability).
Da viele Entwickler nicht auf die Vorteile von relationalen Datenbanken verzichten wollen, bietet db4o die Möglichkeit eine relationale Datenbanken als Backend über Hibernate zu verwenden.