Objektorientierte Datenbanken

Ralf Möller, FH-Wedel

- Beim vorigen Mal:
  - Umsetzung in objektorientierte Modellierung auf Implementierungsebene am Beispiel Java
- Heute:
  - Umsetzung mit Hinblick auf persistente Objekte
- Lernziele:
  - Grundlagen der Programmierung persistenter Objekte
  - Java Data Objects

Wiederholung: UML -> Java

- UML-Klassen -> Java-Klassen
  - Zwei Sichtweisen:
    - Beschreibung von Instanzen (auch in Java)
    - Denotieren Menge von Instanzen (-> Extent)
- UML-Attribute -> Java-Instanzvariablen
- Werte -> Basistypen
- UML-Operationen -> Java-Methoden (+ Code)
- UML-Interfaces -> Java-Interfaces

Umsetzung von Assoziationen in Java

- Einführung von entsprechenden Attributen bei den beteiligten Klassen
  - Verwendung von Typen, mit den eine „Sammlung" von Objekten referenziert werden kann
  - (-> „lokaler" Ansatz)
- Einführung von Objekten zur Repräsentation der Relation
  - Ggf. Navigierbarkeit ausnutzen
  - (-> „globaler" Ansatz)

Vordefinierte Klassen für sog. Collections

- Spezialisierungen von java.util.Collection
  - java.util.Vector
  - java.util.List
  - java.util.Set
  - java.util.Map
- Entsprechende Operationen zum Erzeugen und zum Zugriff auf Elemente bzw. zum Ersetzen von Elementen definiert
- Vordefinierte Verknüpfungsoperationen

java.util.Vector

- Vector x = new Vector;
- x.setElementAt(Integer(42), 0);
- x.setElementAt(Integer(43), 1);
- x.elementAt(0);
- x.length();
- Vector y = new Vector(27);
- x.addElement(56);

Arrays am Beispiel

- int[] a = new int[100];
- a[0] = 1;
- for (int i = 1; i < b.length; i++) {
  a[i] = a[i-1]
  }

Arrays vs. Vektoren

- Bei Array muß die Maximallänge zur Erzeugungszeit (nicht Übersetzungszeit) bekannt sein
- Bei Vektoren (Instanzen der Klasse Vector) ist die Länge variabel
- Länge kann bei beiden erfragt werden (length)

java.util.List, java.util.LinkedList

- List l = new LinkedList;
- l.add(Integer(33));
- l.add(Integer(55));

java.util.Set, java.util.HashSet

- Set s = new HashSet;
- s.add(Integer(3));
- if (s.contains(Integer(3))) {

  ...

  }

java.util.Map, java.util.Hashtable

- Assoziation von „Elementen“ mit „Schlüsseln“
- Map d;
- d=new Hashtable;
- d.put(534958345, „Meyer“)
- d.get(534958345)
- d.remove(534958345)

Lineare Traversierung von Datenstrukturen

- Iterator
  - Objekt, das Traversierungszustand speichert
- Next-Operation zur Fortschreibung der Traversierung
- Operation zur Feststellung, ob Ende erreicht

Beispiel:

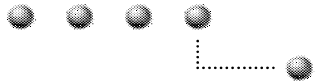
```
import java.util.*;
Map d = new HashTable;
d.put(Integer(3));
d.put(Integer(4));
Iterator iter = d.iterator();
while (iter.hasNext()) {
    Object o = iterator.next();
    int i = ((Integer)o).value();
    ....
}
```

Zusammenfassung: Assoziationen UML -> Java

- Lokaler Ansatz:
  - Wenn Obergrenzen bei Multiplizitäten, dann möglicherweise Arrays einsetzbar
  - Sonst: Vektoren (Vector), Listen (List), Mengen (Set)
- Globler Ansatz:
  - Abbildungen (Map)

Persistenz

- Java-Objekte flüchtig, d.h. nach Programmende nicht mehr vorhanden
- Wunsch nach Objekten, die auch nach Programmende für einen erneuten Programmstart oder für andere Programme noch vorhanden sind
- -> Persistente Objekte



Berner Fachhochschule

Hochschule für Technik und Architektur Bern
Software-Schule Schweiz

Java Data Objects

Letzte Revision: Februar 2003

Dr. Arno Schmidhauser
Software-Schule Schweiz
Morgartenstr. 2c
3014 Bern
arno.schmidhauser@hta-be.bfh.ch
+41313355275

http://www.hta-be.bfh.ch/~schmd/jdo

FAST OBJECTS™
by Poet

Java Data Objects (JDO)
und
Implementierung in FastObjects

Was ist JDO

1. Eine Allzweck-Architektur um Java-Objekte in einer Datenbank abzulegen
   1. unabhängig vom Datenmodell (relational, objekt-orientiert, ...)
   2. unabhängig von der Umgebung (Client-Server, Applikationsserver)

2. Beliebig auswechsel- und kombinierbare "Driver" für verschiedene Datenbanktypen und Hersteller.

3. Durch verschiedenste Interessengruppen abgesegnete Spezifikation.

Ursprung

- Geschaffen unter dem *Java Community Process* von Sun
- **Leiter der Spezifikation**:
  - Craig Russell Sun Microsystems, Inc.
- **Expertengruppe, Mitglieder von:**

IBM	Informix Software	Ericsson Inc.
Oracle	Rational Software	SAP AG
Software AG	Sun Microsystems	**POET**

(und vielen anderen)

- akzeptiert als Standard Ende März 2002

JDO Anwendungskonzept

1. Typsystem von Java, keine speziellen Datentypen wegen Persistenz -> Arrays, Collections, Utility-Klassen, Benutzer-Klassen.

2. Modifikation von Objekten mit Java-Befehlen (Zuweisung), keine speziellen Operatoren wie in SQL.

3. Von der Applikation kontrolliert werden lediglich der Lebenszyklus von Objekten (persistent, transient) und die Transaktionsgrenzen (begin, rollback, commit).

4. Zugriff auf Objekte in der Datenbank via Extent einer Klasse, via bekannte Objekt-ID, via Abfragesprache.

FAST OBJECTS™

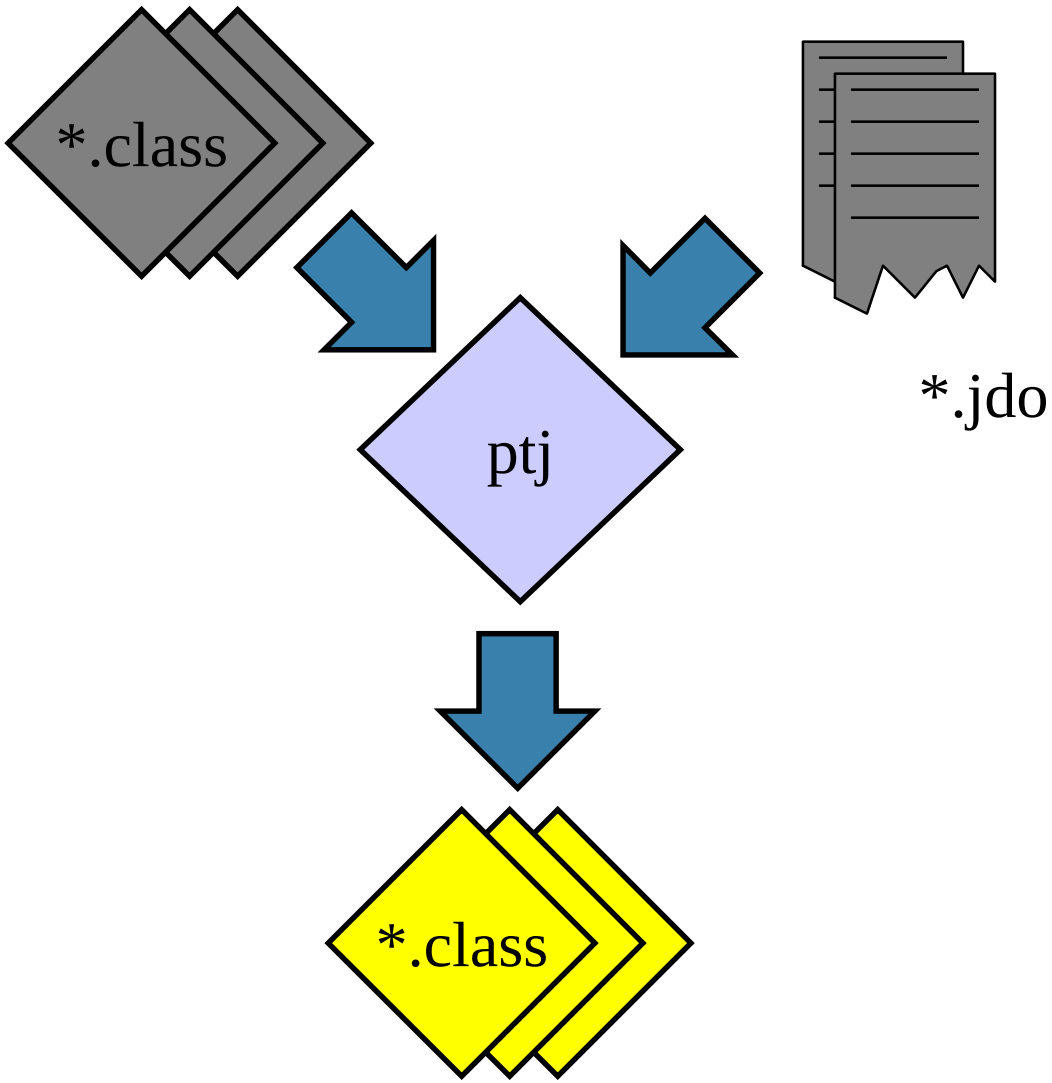
Enhancen

- Persistenzfähige Klassen müssen *javax.jdo.PersistenceCapable* implementieren
  - nicht beliebig, sondern kompatibel zur Referenzimplementation
- Forderung nach Transparenz: diese Implementierung sollte automatisch erzeugt werden
- oftmals mit einem Post-Prozessor, der direkt ByteCode in die .class-Dateien einfügt und/oder abändert
  - FastObjects Enhancer heißt ***ptj***

Enhancer (2)

- Methoden zum Laden und Speichern dieser Objekte werden eingefügt
  - werden intern von der JDO Implementation aufgerufen
- für alle zu speichernden Felder werden spezielle Zugriffs-Methoden (Getter und Setter) eingeführt
  - protokollieren den internen Zustand eines Objektes (clean, dirty, …)
  - jeder direkte Zugriff auf solche Felder wird durch Methodenaufruf ersetzt => auch nicht-persistente Klassen, die direkte Feldzugriffe bei persistenten Instanzen machen, müssen enhanced werden

Enhancement illustriert



FAST OBJECTS™

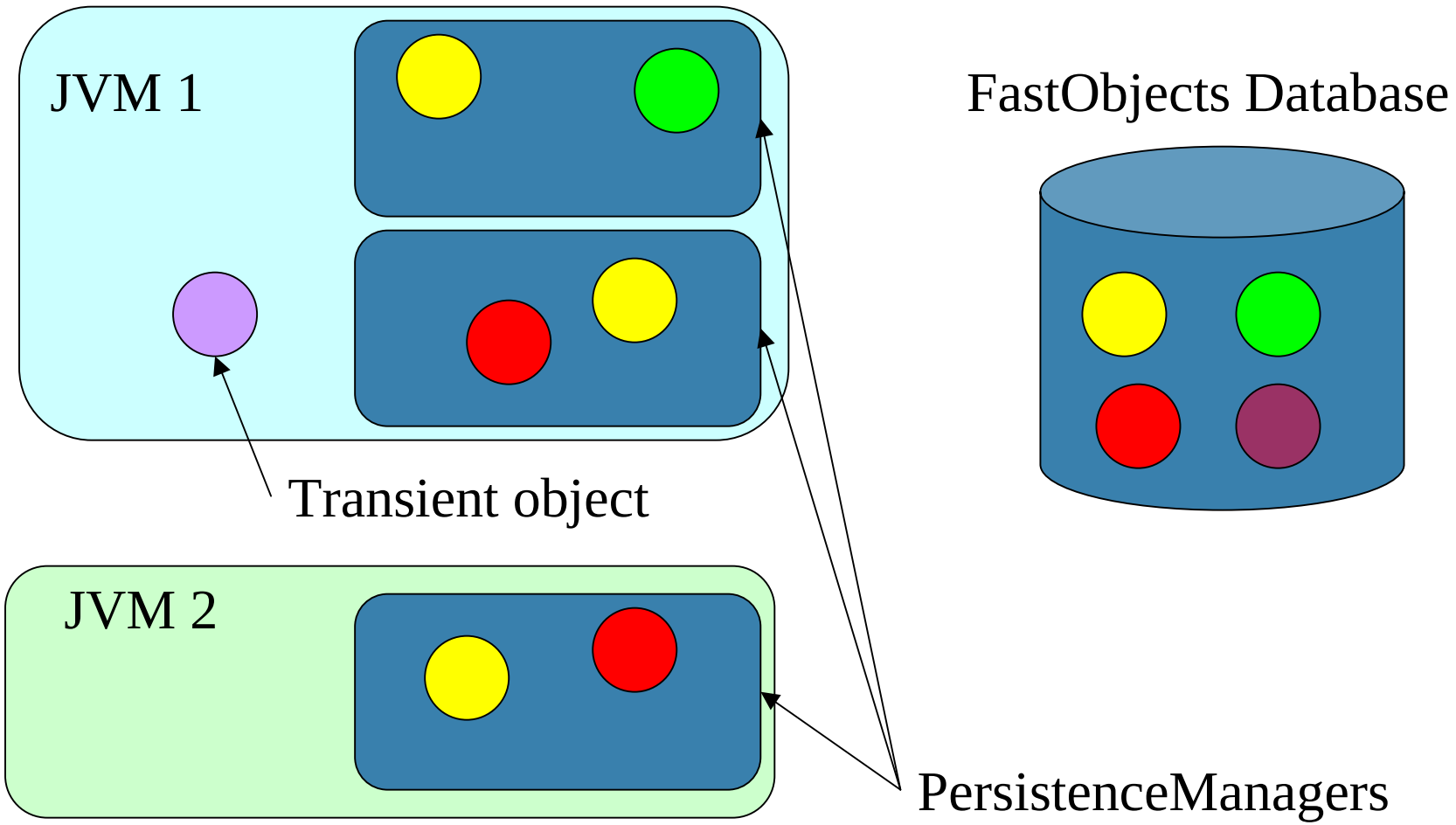
JDO aus Anwendersicht - Datenbanklogik

- Spezifikation definiert eine Menge von Interfaces
  - PersistenceManager, Transaction, Extent, Query, …
- Implementationen dieser Interfaces durch JDO Implementierer (z.B. Poet)
  - Anwender benutzt nur die Interfaces!
- Zentrales Interface: javax.jdo.PersistenceManager
- Persistente Java Objekte erhalten eine eindeutige ObjektId

javax.jdo.PersistenceManager

- Jeder Thread benutzt normalerweise seine eigene PersistenceManager-Instanz
- Jedes persistente Java-Objekt im Speicher gehört zu *genau einem* PersistenceManager
- Jedes persistente Java-Objekt korrespondiert zu *genau einem* Datenbank-Objekt, zwei persistente Java-Objekte des selben PersistenceManagers aber immer zu verschiedenen!

javax.jdo.PersistenceManager (2)



Persistenz

- Objekte, die mit **new** angelegt wurden, sind transient
- Objekte, die aus der Datenbank geholt wurden, sind persistent
  - geholt mittels Query, Iterieren über Extent, Referenz von einem anderen persistenten Objekt
- PersistenceManager.makePersistent()
  - transientes Objekt wird persistent
  - Persistenz durch Erreichbarkeit
- PersistenceManager.deletePersistent()
  - löscht Objekt aus der Datenbank

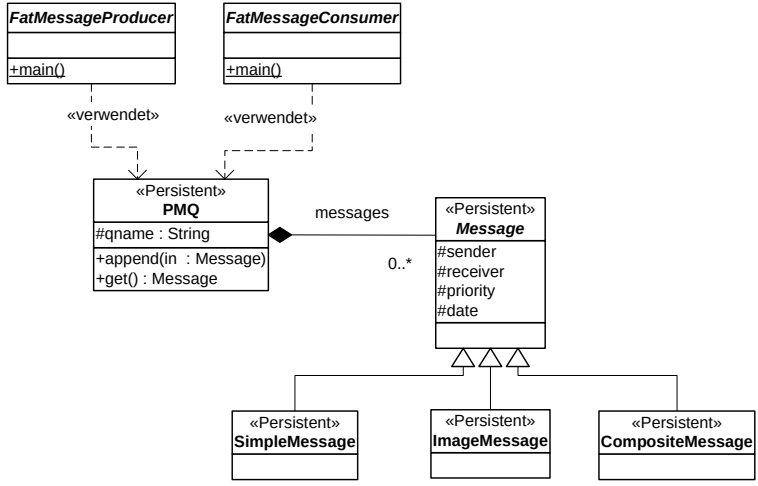
FAST OBJECTS™

Transaktionen

- Pro PersistenceManager maximal eine Transaktion offen
  - geschachtelte Transaktionen optional, von FastObjects unterstützt
  - Nacheinander i.d.R. mehrere Transaktionen pro PersistenceManager
  - Objektreferenzen aus alten Transaktionen bleiben gültig
  - PersistenceManager.currentTransaction();
- Interface javax.jdo.Transaction
  - Transaction.begin()
  - Transaction.commit()
  - Transaction.rollback()
- Transaktionen erfüllen ACID-Eigenschaften

Arbeiten mit JDO

Beispiel


FatMessageProducer
+main()
FatMessageConsumer
+main()
«verwendet»
«verwendet»
«Persistent»
PMQ
#qname : String
+append(in : Message)
+get() : Message
messages
0..*
«Persistent»
Message
#sender
#receiver
#priority
#date
«Persistent»
SimpleMessage
«Persistent»
ImageMessage
«Persistent»
CompositeMessage

Erzeugen persistenter Objekte

```
Properties p = new Properties();
...
PersistenceManagerFactory pmf;
pmf = JDOHelper.getPersistenceManagerFactory( p );
PersistenceManager pm = pmf.getPersistenceManager();
Transaction tra = pm.currentTransaction();
tra.begin();
PMQ pmq = new PMQ(  name );
pm.makePersistent( pmq );
tra.commit();
...
```

Automatische Persistenz

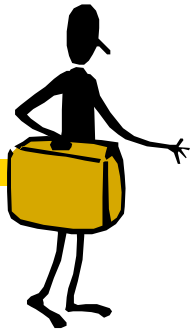
1. Jedes transiente Objekt, welches an ein persistentes Objekt angehängt wird, ist automatisch persistent.

```
...
PMQ pmq = new PMQ(  name );
pm.makePersistent( pmq );
Message m = new SimpleMessage( ... );
pmq.append( m );
...
```


m wird persistent !

2. Ein persistentes Objekt wird nur durch `PersistenceManager.deletePersistent()` wieder aus der Datebank entfernt

Zusammenfassung, Kernpunkte



- UML und Java
- Peristenz
- Einführung Java Data Objects

Was kommt beim nächsten Mal?

- Java Data Objects zweiter Teil