

# ***Objektorientierte Datenbanken***

Vorlesung 10 vom 24.06.2004

Dr. Sebastian Iwanowski

FH Wedel

# **Inhalt heute: JDO – letzter Teil**

**Wiederholung vom letzten Mal: Lebenszykluszustände**

**Vertiefung: Optionale Lebenszykluszustände**

**Zusammenfassung: JDO**

***Wiederholung vom letzten Mal:  
Lebenszykluszustände***

# Lebenszykluszustände eines Objekts

## Obligatorische Zustände:

- transient
- persistent-new
- hollow
- persistent-clean
- persistent-dirty
- persistent-deleted
- persistent-new-deleted

## Optionale Zustände:

- transient-clean
- transient-dirty
- persistent-nontransactional  $\longrightarrow$  *gibt es in FastObjects !*

# Der Zustand hollow

## Wann gilt der Zustand hollow ?

- durch Beendigung einer Transaktion (gilt für alle Objekte des PersistenceManagers)
- durch direkten Objektzugriff aus Datenbank (über Objektid bzw. Objektnamen)
- durch Iterieren eines Extents
- durch Ergebnis einer Query
- durch Navigieren von anderem persistenten Objekt aus

## Was bedeutet der Zustand hollow ?

- Verbindung zu konkretem Datenbankobjekt
- keine Speicherbelegung für Datenfelder (aus Effizienzgründen)
- keine feste Bindung vom PersistenceManager mehr

## Konsequenz ?

- Außerhalb einer Transaktion sind die Datenfelder nicht zugänglich !

# Lebenszykluszustände eines Objekts

Kann man die Zustände abfragen ?

**Methoden von JDOHelper:** transient hollow p.-clean p.-dirty p.-new p.-del. p.-new-del.

<code>isPersistent(Object)</code>	<i>F</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<code>isTransactional(Object)</code>	<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<code>isDirty(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
<code>isNew(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>
<code>isDeleted(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>

**für FastObjects:**

`PersistenceManagers.isHollow (PersistenceManager, Object)`

***Vertiefung:  
Optionale Lebenszykluszustände***

😊 **wird von FastObjects teilweise unterstützt** 😊

# Wiederholung: Persistenzkonzept bei Attributen

## Drei Typen für Attribute persistenter Objekte:

- **persistent**  
normale Persistenz durch Erreichbarkeit, `rollback()` bei Transaktionen wirksam
- **transactional**  
Daten sind immer transient, aber `rollback()` bei Transaktionen wirksam
- **none**  
Daten sind immer transient, `rollback()` bei Transaktionen setzt sie nicht zurück



# Analogon: Lebenszykluszustände der Objekte

## Drei Grundtypen für Objekte persistenzfähiger Klassen:

## Lebenszykluszustände:

- **persistent**  
Objekte sind in der Datenbank und in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen wirksam  
→ • `persistent-clean`  
→ • `persistent-dirty`  
☹ **FastObjects-Anomalien** ☹
- **transactional**  
Objekte sind nicht in der Datenbank, aber in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen wirksam  
→ • `transient-clean`  
→ • `transient-dirty`  
😊 **auch in FastObjects** 😊  
☹ **FastObjects-Anomalien** ☹
- **transient**  
Objekte sind weder in der Datenbank noch in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen unwirksam  
→ • `transient`  
***Defaultzustand***

# Analogon: Lebenszykluszustände der Objekte

## Drei Grundtypen für Objekte persistenzfähiger Klassen:

- **persistent**

Objekte sind in der Datenbank und in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen wirksam

- **transactional**

Objekte sind nicht in der Datenbank, aber in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen wirksam

- **transient**

Objekte sind weder in der Datenbank noch in der Cacheverwaltung des PersistenceManagers, `rollback()` bei Transaktionen unwirksam

## Methoden von PersistenceManager:

`makePersistent(obj)`

`makeTransactional(obj)`

😊 **auch in FastObjects** 😊

`makeNontransactional(obj)`

😞 **FastObjects-Anomalien** 😞

`makeTransient(obj)`

# Wiederholung: Transaktionskonzept

## 3 Transaktionsstrategien:

- **Normale (pessimistische) Transaktionen**

Bei Zugriff wird eine Sperre auf das betreffende Datenbankobjekt gelegt.

Erst bei Transaktionsende wird die Sperre aufgehoben.

- **Optimistische Transaktionen**

Es gibt während der Transaktion die meiste Zeit keine Sperre.

Es wird vor dem Commit nachgeprüft, ob die benutzten Daten sich während der Transaktion geändert haben (dabei gibt es eine Sperre).

Falls sich die Daten während der Transaktion geändert haben, erfolgt eine Nachricht an die Transaktion als Exception und kein Commit.

- **Datenbankzugriff außerhalb von Transaktionen**

für Objekte in bestimmten Zuständen (wird **jetzt** besprochen)

# Datenbankzugriff außerhalb von Transaktionen

Die in einer Transaktionen  
zugegriffenen Daten können  
von dieser Transaktion aus in  
folgende Zustände gebracht  
werden:

- **nontransactionalRead**

zum Lesen von Feldern  
außerhalb von Transaktionen

- **nontransactionalWrite**

zum Beschreiben von Feldern  
außerhalb von Transaktionen

## Methoden von Transaction:

`setNontransactionalRead(bool)`

☹ nicht in FastObjects ☹

`setRetainValues(bool)`

behält alle Werte im Cache des  
PersistenceManagers nach Ende der  
Transaktion

☺ auch in FastObjects ☺

`setRestoreValues(bool)`

stellt alte Werte im Cache des  
PersistenceManagers her nach rollback  
der Transaktion

☺ auch in FastObjects ☺

`setNontransactionalWrite(bool)`

☹ nicht in FastObjects ☹

# Datenbankzugriff außerhalb von Transaktionen

Zugriffe sind für alle Daten möglich, die sich noch im Cache vom `PersistenceManager` befinden:

- **transaktionale Objekte**  
nicht persistente Objekte unter der Kontrolle des Persistenzmanagers

- **persistente Objekte**

**Problem:**

Diese Objekte wechseln im Normalfall nach Ende einer Transaktion in den Zustand `hollow` und fallen aus der Kontrolle des Persistenzmanagers

**Lösung:**

Für nichttransaktionalen Zugriff wechseln diese Objekte in den neuen Zustand `persistent-nontransactional`

**Anmerkung:**

Im Zustand `persistent-nontransactional` befinden sich auch die persistenten Objekte von optimistischen Transaktionen, bevor `commit` gemacht wird.

# Optionale Lebenszykluszustände eines Objekts

Abfragemöglichkeiten für die optionalen Zustände:

Methoden von JDOHelper: transient hollow p.-nontransactional t.-clean t.-dirty

<code>isPersistent(Object)</code>	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>F</i>
<code>isTransactional(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>T</i>
<code>isDirty(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>
<code>isNew(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>
<code>isDeleted(Object)</code>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>

für FastObjects:

`PersistenceManagers.isHollow (PersistenceManager, Object)`

`PersistenceManagers.isPersistentNonTransactional (PersistenceManager, Object)` ☹️ **FastObjects-Anomalien** ☹️

# Praktische Erfahrungen mit FastObjects

- **Lesezugriff außerhalb von Transaktionen:**

nur nach `setRetainValues(true)` oder `setRestoreValues(true)` erlaubt

- **Schreibzugriff außerhalb von Transaktionen:**

nicht zulässig

# Praktische Erfahrungen mit FastObjects

- **transaktionale Objekte:**

Transaktionale Objekte wechseln nach `commit` der Transaktion, in der sie entstanden sind, in den Zustand `transient-clean`.

Wenn sie danach in einer Transaktion geändert werden, also in den Zustand `transient-dirty` wechseln, so darf diese Transaktion nur mit `rollback` abgeschlossen werden, d.h. **commit ist nicht zulässig**.

Im Lesezugriff wird also stets der Wert gelesen, der in der ersten oder in der gegenwärtigen Transaktion gesetzt wurde.

Ein **Wechsel in den Zustand transient** mit `makeNontransactional` **ist im Zustand transient-dirty nicht zulässig**.



# Praktische Erfahrungen mit FastObjects

- **persistente Objekte:**

Persistente Objekte wechseln nach `commit` der Transaktion, in der auf sie zugegriffen wurde und in der `setRetainValues (true)` oder `setRestoreValues (true)` ausgeführt wurde, in den Zustand `persistent-nontransactional` (nicht `hollow!`). Wenn sie danach in einer Transaktion geändert werden, die mit `rollback` abgeschlossen wird, **dann behalten sie im Cache dennoch die neuen Werte.**

Ein **Wechsel in den Zustand `persistent-nontransactional` mit `makeNontransactional` ist im Zustand `persistent-dirty` nicht zulässig.**

Ein **Wechsel in den Zustand `persistent-nontransactional` mit `makeNontransactional` ist im Zustand `persistent-clean` zwar zulässig, hat aber **keine Auswirkungen**: Innerhalb von Transaktionen wird bei Feldänderungen automatisch in den Zustand `persistent-dirty` zurückgekehrt.**

Außerhalb von Transaktionen ist Lesezugriff nur erlaubt, wenn `setRetainValues(true)` oder `setRestoreValues (true)` ausgeführt wurde. Anderenfalls geht Objekt nach Beendigung der Transaktion **in den Zustand `hollow`, auch wenn die `PersistenceManagers`-Methoden etwas anderes anzeigen !**

# ***Zusammenfassung: JDO***

# Zusammenfassung: JDO

## **Vorlesung 5: Der PersistenceManager als Dreh- und Angelpunkt**

alle Lese- und Schreibzugriffe, Transaktionsmanagement

## **Vorlesung 6: Persistenz- und Transaktionskonzept**

Attributmanagement via XML-Metadaten, transiente / transaktionale und persistente Attribute, First-Class- und Second-Class-Objekte

Transaktionseigenschaften, Isolationsniveaus, Transaktionsstrategien (pessimistisch und optimistisch)

## **Vorlesung 7 / 8: JDOQL**

Filter in Java-Syntax, Parameter- und Variablendeklarationen, Anfragen mit collection-wertigen Attributen, Vergleich zu OQL

# Zusammenfassung: JDO

## **Vorlesung 9: Datenidentitätskonzept und Lebenszykluszustände**

ID-Klassen, Datastore- und Application-Identity,  
verschiedene Zugriffsmöglichkeiten auf ein Datenbankobjekt

obligatorische Lebenszykluszustände, Wirkung und Zweck (besonders `hollow`),  
Zustandsabfrage

## **Vorlesung 10: Optionale Lebenszykluszustände**

transaktionale Objekte, nichttransaktionaler Datenzugriff

***Beim nächsten Mal:  
Weitere Datenbankbindungstechniken  
(über OODB hinaus)***