Objektorientierte Datenbanken

Vorlesung 9 vom 10.06.2004

Dr. Sebastian Iwanowski
FH Wedel

JDO

- **Persistenzkonzept**

  Persistenzfähige Klassen, objektspezifische Persistenz, Persistenz durch Erreichbarkeit

- **Transaktionskonzept**

  mehrere Transaktionsmanagementstrategien

- **Anfragesprache (JDOQL)**

  mit objektorientiertem Fokus, Java-Syntax

- **Konzept zum Management von Datenveränderungen**

  über so genannte Lebenszykluszustände von Daten
  definiert Mechanismen für den Lebenszyklusübergang

- **Datenidentitätskonzepte**

  berücksichtigt unterschiedliche Anforderungen von Datenbank und Programm

Offene Frage vom letzten Mal:

Welche Datentypen sind als Parameter in einer JDOQL-Anfrage zulässig ?

Antwort:

Alle persistenzfähigen Klassen und alle einfachen Datentypen

Inhalt heute:

JDO-Datenidentitätskonzepte

FastObjects-Erweiterungen zur Datenidentität

Konzept zum Management von Datenveränderungen:

- **Lebenszykluszustände**

JDO-Datenidentitätskonzepte

JDO-Datenidentitätskonzepte

Wozu benötigt man die genaue Definition der Identität ?

- für die Suche nach einem bestimmten Objekt in der Datenbank
- für Verweise auf andere Objekte in der Datenbank
- für die Arbeit mit persistenten Objekten in der Java Virtual Machine
- für den Abgleich zwischen Objekten in der JVM und der Datenbank

JDO-Datenidentitätskonzepte

Wie bestimmt man im Allgemeinen die Identität eines Objekts ?

- **durch Gleichheit der Adresse**

- **durch Beantwortung bestimmter Äquivalenzmethoden mit `true`**

- **durch Verwendung eines eindeutigen Namens**

JDO-Datenidentitätskonzepte

JDO-Konzept:

- **Jedes First-Class-persistente Objekt hat eine JDO-spezifische ID**
- **Jede JDO-spezifische ID gehört zu einer ID-Klasse, die folgende Eigenschaften erfüllt:**
  - **Die ID-Klasse ist `public`**
  - **Die ID-Klasse implementiert das Interface `Serializable`**
  - **Die ID-Klasse hat einen Konstruktor ohne Parameter**
  - **Die ID-Klasse überschreibt `toString()` für Vergabe eines eindeutigen Namens.**
  - **Die ID-Klasse hat einen Konstruktor, der einen `String` als Parameter hat.**
- **Jede persistenzfähige Klasse hat eine eindeutige ID-Klasse**
- **Die ID-Klasse einer persistenzfähigen Klasse wird in den XML-Metadaten festgelegt:**

1) entweder automatisch 2) oder individuell gewählt

JDO-Datenidentitätskonzepte

1) Automatische Festlegung der ID-Klasse: Datastore-Identity

Konzept: Identität wird durch eindeutigen Namen festgelegt

- Der Identitätsname kann aber nicht selbst gewählt werden, sondern wird durch die Persistenzverwaltung automatisch festgelegt.
- Ebenso wird die Klasse und Instanz der ObjektId durch die Persistenzverwaltung automatisch festgelegt.
- Zugriff über Objektname (als String) oder ObjektId (als Object)

Vorteile ?

JDO-Datenidentitätskonzepte

2) Individuelle Festlegung der ID-Klasse: Application-Identity

Konzept: Identität wird durch Primärschlüsselfelder festgelegt

- **Klasse und Instanz der ObjektId wird durch die Applikation festgelegt.**
- **Ein eindeutiger Objektname, der sich aus den Primärschlüsseln zusammensetzen muss, kann ebenfalls durch die Applikation festgelegt werden.**
- **Zugriff über Objektname (als String) oder ObjektId (als Object)**

Jede Application-ID-Klasse muss folgende zusätzliche Eigenschaften erfüllen:

- **Die ID-Klasse hat für jedes Primärschlüsselfeld ein `public` Feld gleichen Namens**
- **Die Methoden `equals(obj)` und `hashCode()` müssen von jedem einzelnen Primärschlüsselfeld abhängen**
- **Die Datentypen der Primärschlüsselfelder müssen zu denen gehören, die per default persistenzfähig sind (**außer **`HashSet`**, siehe OODB 6, Folie 8**)**

Vorteile ?

JDO-Datenidentitätskonzepte

3) Spezialfall: nondurable Identity

- Für Datenbankobjekte, die in der Datenbank keine ID erhalten können
- Realisierung: produktabhängig

JDO-Datenidentitätskonzepte

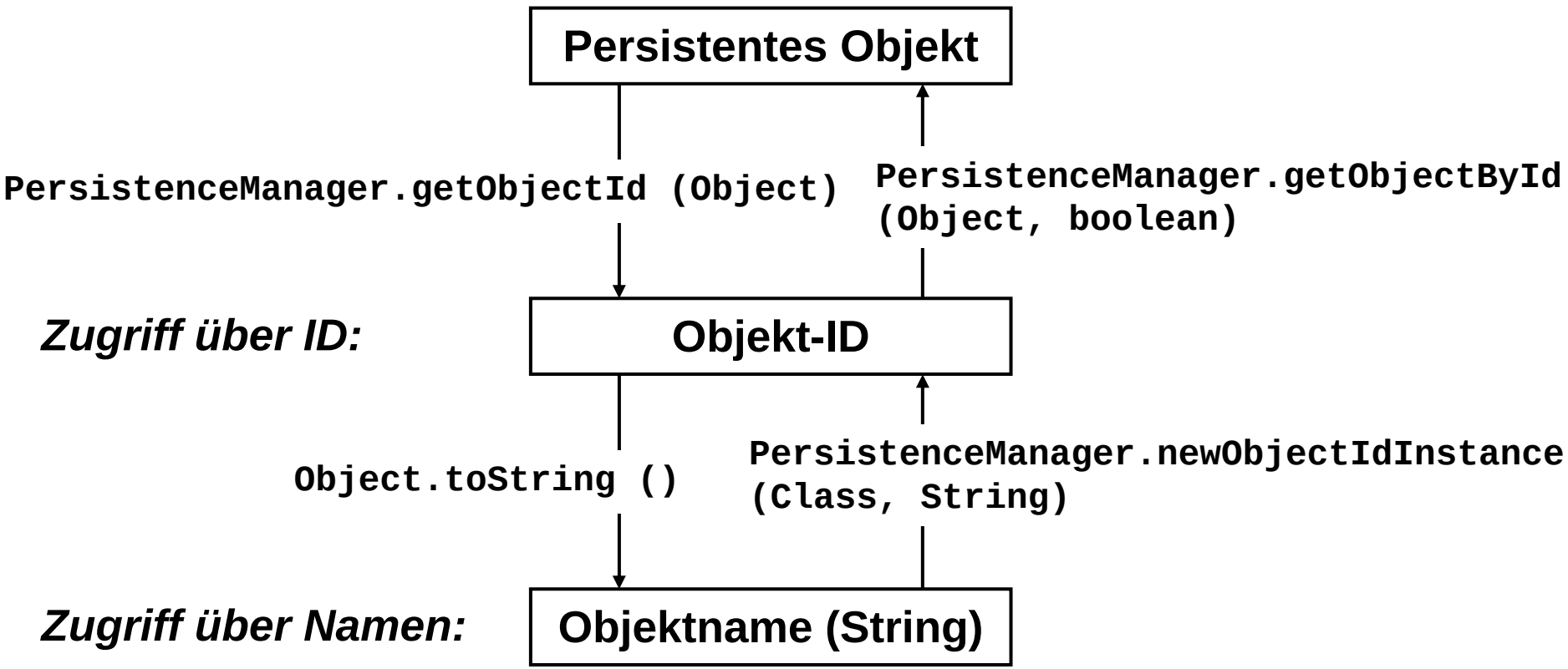
Zusammenfassung:

JDO kennt folgende Datenidentitätskonzepte:

- **Datastore identity** ☺ **wird von FastObjects voll unterstützt** ☺
- **Application identity** ☹ **wird von FastObjects nicht unterstützt** ☹
- **nondurable identity** ☹ **wird von FastObjects nicht unterstützt** ☹

JDO-Datenidentitätskonzepte

Möglichkeit des direkten Zugriffs auf ein Datenbankobjekt:



FastObjects-Erweiterungen zur Datenidentität

FastObjects-Erweiterungen zur Datenidentität

Zugriff über Namensbindung wie bei ODMG:

Methoden von `com.poet.jdo.PersistenceManager`s:

```
static void makePersistent (PersistenceManager, Object, String)

static Object findObject (PersistenceManager, String)

static void deletePersistentName (PersistenceManager, String)
```

FastObjects-Erweiterungen zur Datenidentität

Standard-JDO-Zugriffsmöglichkeiten auf Datenbankobjekte:

- via Extent
- via JDO-Query
- via Object-ID / Objektnamen

weitere FastObjects-Zugriffsmöglichkeit:

- via Objektwurzel(n) zu allen persistenten Objekten, die erreicht werden können

  Methode von `com.poet.jdo.NetWalker`:

  ```
  Collection compute (Transaction, Object)
  ```

Konzept zum Management von Datenveränderungen: Lebenszykluszustände

Lebenszykluszustände eines Objekts

- **Jedes Objekt einer persistenzfähigen Klasse hat Zustand, der die Persistenz„geschichte“ beschreibt**
- **Die Zustände werden intern als flags abgespeichert**
- **Zustände werden durch bestimmte Aktionen implizit gesetzt**
- **Einige Zustände können abgefragt oder explizit gesetzt werden**
- **Gegenwärtige Zustände beeinflussen die Ausführung anderer Aktionen**

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** ⟶ ***gibt es in FastObjects !***

Lebenszykluszustände eines Objekts

Zusammenhang zwischen den obligatorischen Zuständen:

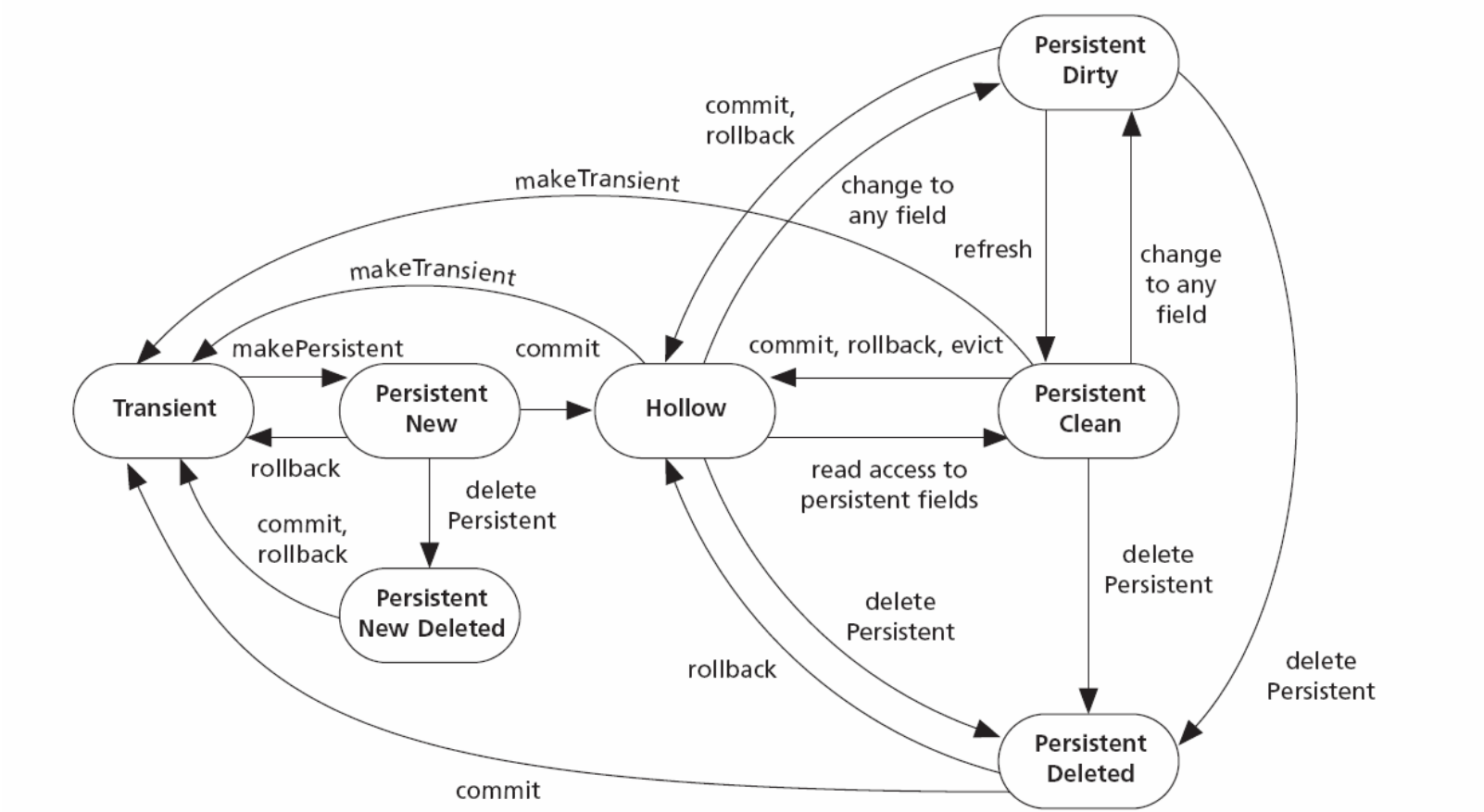


Figure 4.9 All required state transitions

Lebenszykluszustände eines Objekts

Welche Auswirkungen haben die Zustände ?

Transiente Objekte:

- **keine Verbindung zu irgendeinem Datenbankobjekt**
- **`makePersistent` erzeugt immer neues Datenbankobjekt**

„hohle“ (hollow) Objekte:

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

(könnte durch Garbage Collector aufgeräumt werden!)

Lebenszykluszustände eines Objekts

Erreichen des „Spezialzustands“ `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**

➔ `hollow` ist der „Normalzustand“ eines Objekts !

Lebenszykluszustände eines Objekts

Kann man die Zustände abfragen ?

Methoden von JDOHelper:

	transient	hollow	p.-clean	p.-dirty	p.-new	p.-del.	p.-new-del.
`isPersistent(Object)`	*F*	*T*	*T*	*T*	*T*	*T*	*T*
`isTransactional(Object)`	*F*	*F*	*T*	*T*	*T*	*T*	*T*
`isDirty(Object)`	*F*	*F*	*F*	*T*	*T*	*T*	*T*
`isNew(Object)`	*F*	*F*	*F*	*F*	*T*	*F*	*T*
`isDeleted(Object)`	*F*	*F*	*F*	*F*	*F*	*T*	*T*

für FastObjects:

```
PersistenceManagers.isHollow (PersistenceManager, Object)
```

Beim nächsten Mal:
JDO, letzter Teil (inklusive Zusammenfassung)