

# ***Objektorientierte Datenbanken***

Vorlesung 4  
Sebastian Iwanowski  
FH Wedel

***Gewährleistung von Integritätsbedingungen  
durch objektorientierte Programmierung  
am Beispiel ODMG***

# Integritätsbedingungen

## Was ist eine Integritätsbedingung ?

Eine **Integritätsbedingung** ist eine beliebige logische Bedingung, die zwischen den betrachteten Daten gelten muss

### statische Integritätsbedingung

Zulässige Daten hängen von einer absoluten Bedingung ab.

### dynamische Integritätsbedingung

Zulässige Daten hängen vom Zustand der zuletzt angenommenen Datenbelegung ab.

*„weiche“ Bedingung*

Zulässige Daten hängen von der Entwicklung der in der Vergangenheit angenommenen Zustände ab.

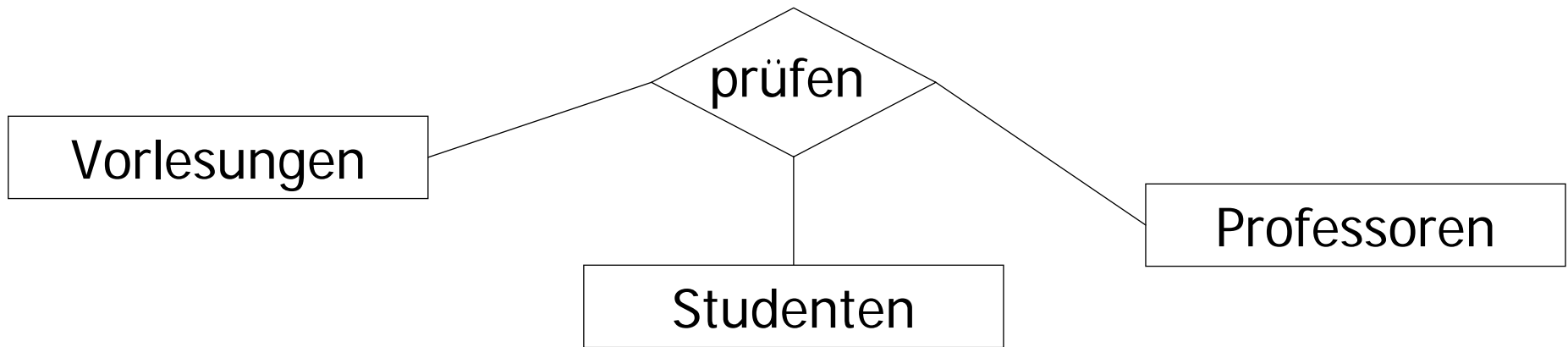
*„scharfe“ Bedingung*

**In OOP können Integritätsbedingungen immer gewährleistet werden durch:**

**a) Typvereinbarungen**

**oder b) Implementierung entsprechender Manipulationsmethoden**

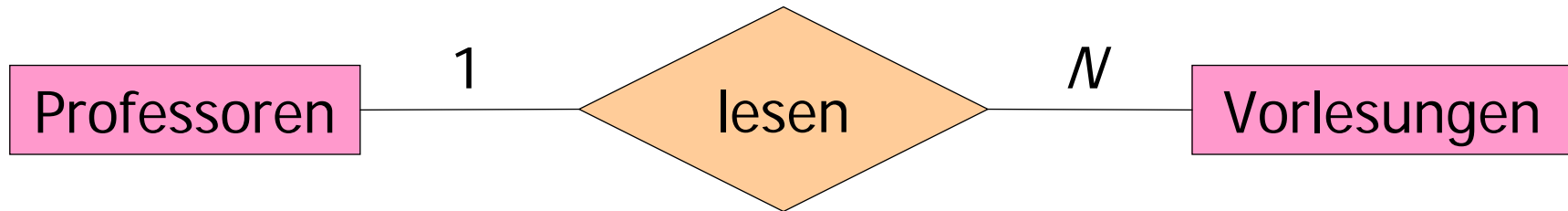
# 1. Beispiel für eine statische Integritätsbedingung:



Typvereinbarungen legen fest, ob Studenten von Professoren oder Assistenten geprüft werden:

```
class Prüfungen {
    attribute Datum Prüfdatum; // ist Klasse, aber dennoch Attribut
    attribute float Note;
    relationship Professoren Prüfer inverse Professoren::hatGeprüft;
    relationship Studenten Prüfling inverse Studenten::wurdeGeprüft;
    relationship Vorlesungen Inhalt inverse Vorlesungen::wurdeAbgeprüft;
};
```

## 2. Beispiel für eine statische Integritätsbedingung:



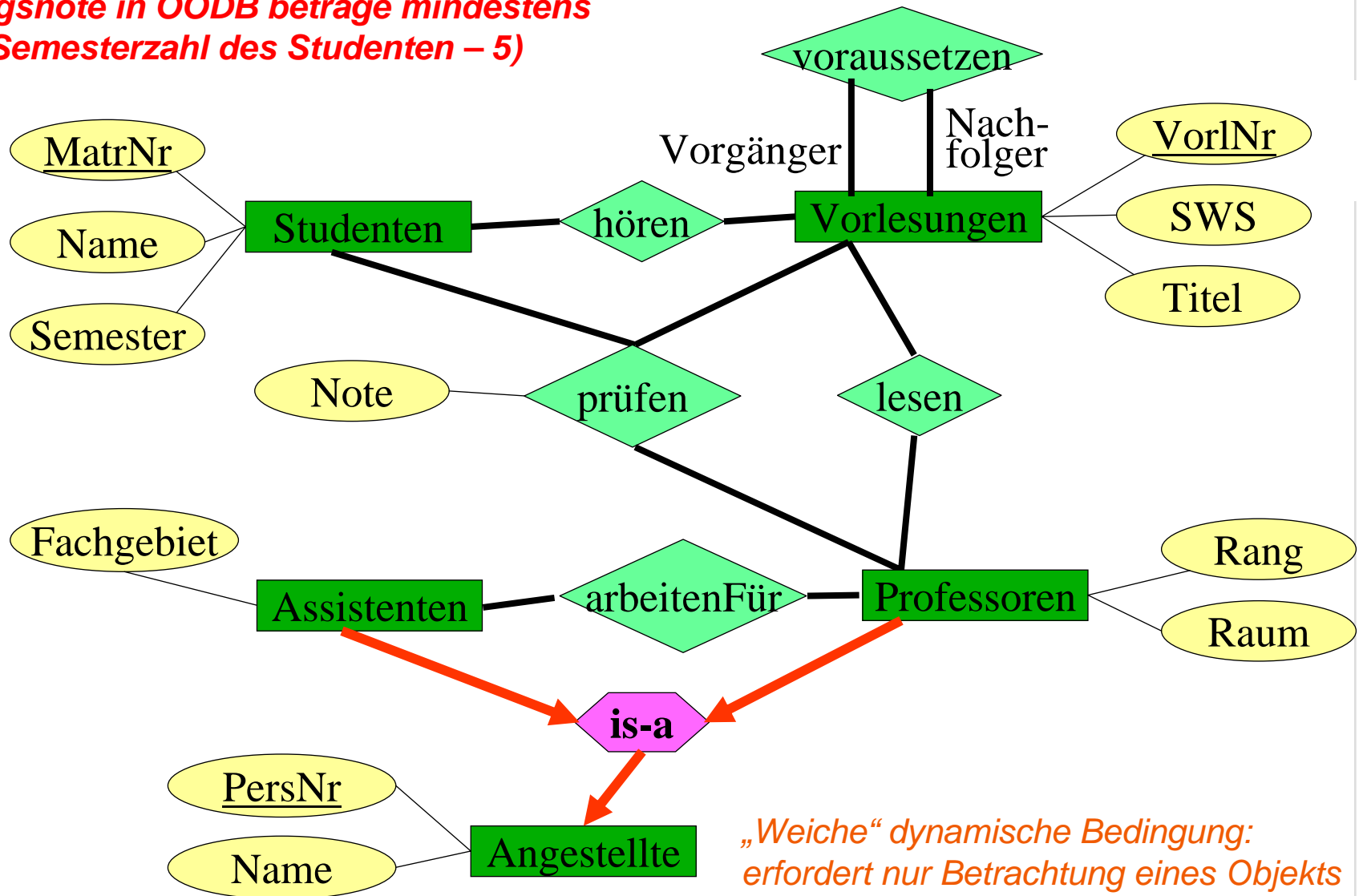
```
class Professoren {
    ...
    relationship set(Vorlesungen) liest inverse Vorlesungen::gelesenVon;
    boolean attachVorlesung (Vorlesung vorlesung)
    {
        if ((Vorlesungen.size() >= N)
            && !(Vorlesungen.contains (vorlesung)))
            return false;
        else
        {
            Vorlesungen.insert (vorlesung);
            vorlesung.wirdGelesenVon (this);
            return true;
        }
    }
    ...
};
```

Der Set-Typ sorgt dafür, dass ein Professor mehrere Vorlesungen halten kann.

Die Zugriffsmethode sorgt dafür, dass ein Professor nicht mehr als N Vorlesungen hält:

# Beispiel für eine dynamische Integritätsbedingung:

Die Prüfungsnote in OODB betrage mindestens den Wert (Semesterzahl des Studenten – 5)



# Dynamische Integritätsbedingungen: Weitere Beispiele

- Für die Mitarbeiter gilt, dass das Durchschnittsgehalt der Qualitätssicherer unter dem der Arbeitsplaner liegen muss

*„Weiche“ dynamische Bedingung:  
erfordert aber die Betrachtung vieler Objekte*

- Der Durchschnittsverkaufspreis eines Produktes bezogen auf die letzten zwölf Monate darf nicht mehr als fünf Prozent vom Durchschnittspreis der letzten beiden Jahre abweichen.

*„Scharfe“ dynamische Bedingung:  
erfordert die Abspeicherung vergangener Zustände*

# **JDO: Einführung**

**Entstehungsgeschichte und Ziele von JDO**

**Überblick über den Leistungsumfang von JDO**

**Grundlagen Persistenz / Transaktionen**

**Zusammenfassung: Erste Anwendung mit JDO**



# Entstehungsgeschichte von JDO

**OODB-Zugriffsstandard, exklusiv für Java**

**entwickelt von Sun, mit Implementierung**

**Förderer: Rick Cattell (Sun, Leiter der ODMG 1991)**

**maßgeblich beteiligt: David Jordan, Craig Russell (Buchautoren)**



**entstanden 1999, eingeführt 2000, seitdem sukzessiv weiterentwickelt**

# Ziele von JDO

**Vereinheitlichung und Verbesserung der folgenden älteren Persistenztechniken für Java:**

- **Serialisierung**

Kommunikation von Java mit anderer Software über Streams

- **JDBC (Java DataBase Connectivity)**

SQL-Anfragen direkt aus dem Java-Programm heraus

- **EJB CMP (Enterprise Java Beans Container Managed Persistence)**

Java in verteilten Anwendungen (sehr komplex)

# Leistungsumfang von 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

# Unterschied zum ODMG-Standard

- **Persistenzkonzept**

gleiches Prinzip wie Java-Anbindung der ODMG, aber genauere Beschreibungsvorgaben, in JDO wird die Zusatzinformation in XML festgelegt (entspricht ptj.opt-Datei bei ODMG).

- **Transaktionskonzept**

wesentlich umfangreicher als ODMG: ODMG kennt nur ein Konzept, legt Details nicht fest.

- **Anfragesprache (JDOQL)**

sehr verschieden von OQL: OQL ist eher SQL-basiert.  
Leistungsumfang von JDOQL mit OQL nicht vergleichbar.

- **Konzept zum Management von Datenveränderungen**

in ODMG nicht festgelegt

- **Datenidentitätskonzepte**

in ODMG nicht festgelegt

# Grundlagen Persistenz / Transaktionen

## In JDO vorgeschriebene Java-Interfaces:

(alle in Package javax.jdo)

- **PersistenceManagerFactory**

generiert Instanzen des Interfaces PersistenceManager,  
speichert die Verbindung zur zugehörigen Datenbank,  
legt die zu den Interfaces gehörenden Klassen des JDO-Anbieters fest.

- **PersistenceManager**

generiert Instanzen der Interfaces Transaction und Extent,  
stellt damit die Funktionalität für Transaktionen, Anfragen und Persistenzoperationen  
zur Datenbank der zugehörigen PersistenceManagerFactory bereit.

- **Transaction**

beginnt und beendet Transaktionen

- **Extent**

liefert Extents zu einer gegebenen Klasse zur weiteren Bearbeitung

# Grundlagen Persistenz / Transaktionen

## Generierung einer PersistenceManagerFactory:

über die Klasse **JDOHelper** (in javax.jdo):

```
PersistenceManagerFactory pmf =  
    JDOHelper.getPersistenceManagerFactory( properties );
```

- `properties` muss Instanz der Klasse `java.util.Properties` sein (ist Spezialisierung von `Hashtable` und damit von `Dictionary`).
- In `properties` wird die Zuordnung zur Datenbank und zu den Implementierungsklassen festgelegt.

**Beispiel** (für FastObjects):

```
java.util.Properties pmfProps = new java.util.Properties();  
  
pmfProps.put("javax.jdo.PersistenceManagerFactoryClass",  
            "com.poet.jdo.PersistenceManagerFactories" );  
    // names the class responsible for creating the  
    // FastObjects persistence manager factory implementation  
pmfProps.put("javax.jdo.option.ConnectionURL", "fastobjects://LOCAL/MyBase" );  
    // names a connection to the database specified by the URL  
PersistenceManagerFactory pmf =  
    JDOHelper.getPersistenceManagerFactory( pmfProps );
```

# Grundlagen Persistenz / Transaktionen

## Generierung der anderen benötigten Objekte:

- **PersistenceManager**

```
PersistenceManager pm = pmf.getPersistenceManager();
```

`pmf` muss das Interface `javax.jdo.PersistenceManagerFactory` implementieren

- **Transaction**

```
Transaction txn = pm.currentTransaction();
```

`pm` muss das Interface `javax.jdo.PersistenceManager` implementieren

- **Extent**

```
Extent ext = pm.getExtent( someClass.getClass(), true );
```

`pm` muss das Interface `javax.jdo.PersistenceManager` implementieren

Der 1. Parameter muss persistenzfähig sein. Der 2. Parameter legt fest, ob die Instanzen der Unterklassen im Extent enthalten sein sollen oder nicht.

# Grundlagen Persistenz / Transaktionen

Für den Anfang benötigte Methoden:

## Interface PersistenceManagerFactory

```
public PersistenceManager getPersistenceManager ();
```

## Interface PersistenceManager

```
public Transaction currentTransaction ();
```

```
public getExtent  
    (Class persistenceCapableClass,  
     boolean subclasses);
```

```
public void makePersistent (Object obj);
```

**Objektbindung durch Name geht nicht !**

```
public void deletePersistent (Object obj);
```

## Interface Transaction

```
public void begin();
```

```
public void commit();
```

```
public void rollback();
```

## Interface Extent

```
public Iterator iterator ();
```

## Interface java.util.Iterator

```
public boolean hasNext ();
```

```
public Object next ();
```

**weitere Methoden: in FastObjects-JavaAPI nachsehen !**



# Grundlagen Persistenz / Transaktionen

## Festlegung der persistenzfähigen Klassen in XML-Datei:

**Beispiel** (minimale XML-Datei, um someClass persistenzfähig zu machen):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name="">
    <class name="someClass">
      </class>
    </package>
  </jdo>
```

← *FastObjects-spezifische DTD*

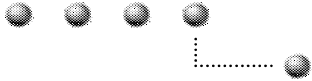
Original-DTD für JDO von Sun in  
<http://java.sun.com/dtd>

***! ACHTUNG: someClass muss Konstruktor someClass () haben !***

- Die Datei muss heißen: `someClass.jdo`  
Sie kann nur die Klasse `someClass` spezifizieren.
- Eine Datei zur Spezifikation mehrerer Klassen muss heißen: `package.jdo`

# Zusammenfassung: Erste Anwendung mit JDO

- **Persistenzfähige Klassen in XML-Datei beschreiben**
- **Datenbankanbindung über Properties an PersistenceManagerFactory**
- **Alle anderen Operationen über PersistenceManager erreichbar**
- **Vorerst können Objekte nur via Extent aus Datenbank geholt werden**
  - **Objektbindung durch Name geht nicht !**
- **Alle anderen Prinzipien wie bei ODMG:**
  - **Persistenz durch explizite Deklaration**
  - **Persistenz durch Erreichbarkeit**
  - **Datenbankoperationen nur in Transaktionen (Ausnahmen kommen später)**
  - **Persistenzfähige Klassen müssen nach jeder Änderung enhanced werden.**



**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>

# 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 Datenbank entfernt

# Extent

1. Der Extent ist ein *logisches* Gefäß für die Menge aller Objekte einer bestimmten Klassen, allenfalls auch ihrer Unterklassen.
2. Das Konstruieren des Extends beinhaltet noch nicht das Holen der Objekte aus der Datenbank.
3. Der Extent stellt einen Iterator zur Verfügung. Der Iterator bestimmt den Algorithmus für das Abholen der Objekte aus der Datenbank.
4. Queries können über einen Extent oder eine Collection durchgeführt werden. Queries über einen Extent werden *serverseitig* abgewickelt, Queries über eine Collection *clientseitig*.

# Extent, Beispiel

```
...
PersistenceManager pm = pmf.getPersistenceManager();
tra.begin();
Extent e = pm.getExtent( PMQ.class, true );
Iterator it = e.iterator();
if ( it.hasNext() ) {
    PMQ pmq = (PMQ) it.next();
    if ( pmq.getName().equals( "myQueue" ) ) {
        SimpleMessage sm = new SimpleMessage( ... );
        pmq.append( sm );
    }
}
tra.commit();
...
```

# Grundsatz

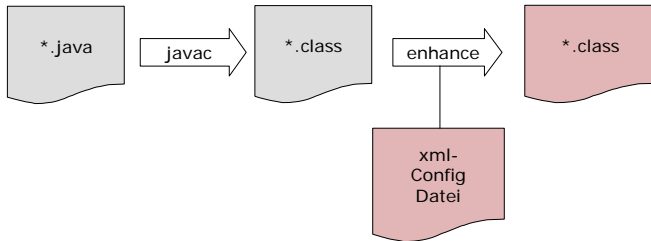
- Für den Entwickler sollen sich das Arbeiten mit der Datenbank auf gewisse *logische* Aufgaben beschränken:
  - Objekte als persistent markieren oder löschen
  - Transaktion starten, comitten oder rollbacken
  - Objekte gezielt aus der Datenbank holen via Query oder Objekt-ID
- Für den programmiersprachlichen Umgang mit Objekten gilt:

Arbeiten mit Datenbankobjekten = Arbeiten mit Java-Objekten



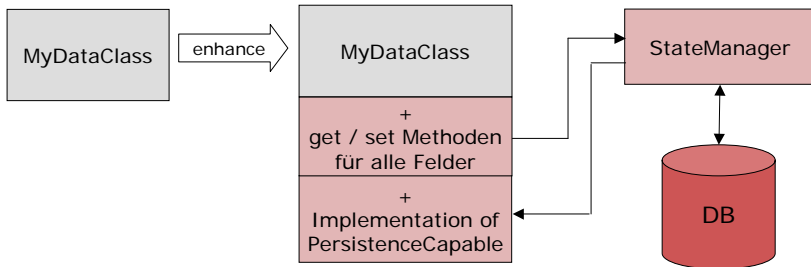
# JDO Enhancer

- Die Umprogrammierung des Java-Codes (Enhancement) findet auf Ebene Bytecode, also mit den Class-Files statt.



# JDO Enhancer

- In Java-Klassen, welche mit persistenten Objekten *arbeiten*, werden die Operatoren, welche diese Objekte verändern, in *Methodenaufrufe* umprogrammiert.
- In Java-Klassen, welche *selber persistente Objekte* repräsentierten, werden die entsprechenden Methoden durch den Enhancer *implementiert*.



# XML Konfigurationsdatei

- Das Konfigurationsfile beschreibt die persistenten Klassen.
- Wird für den Postprocessor (Enhancer) und zur Laufzeit der Applikation benötigt.

```
<jdo>
  <package name="pmq">
    <class name="PMQ"/>
    <class name="Message" identity-type="datastore"/>
    <class name="ImageMessage"
      persistence-capable-superclass="pmq.Message"/>
    <class name="SimpleMessage"
      persistence-capable-superclass="pmq.Message"/>
    <class name="CompositeMessage"
      persistence-capable-superclass="pmq.Message"/>
  </package>
</jdo>
```

Beispiel: Datei pmq.jdo

***beim nächsten Mal:***

***JDO:***

***Persistenzkonzept und Transaktionskonzepte***