Objekt-relationales Mapping mit Hibernate

Michael Eckel

FH Gießen-Friedberg

3. Juni 2008

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Einführung
Um was geht es?

- Wie kann man Daten in einer Java-Anwendung dauerhaft speichern?

Einführung
Um was geht es?

- Wie kann man Daten in einer Java-Anwendung dauerhaft speichern?
- Objekte sind transient
- Sollen aber persistent sein

Einführung
Um was geht es?

- Wie kann man Daten in einer Java-Anwendung dauerhaft speichern?
- Objekte sind transient
- Sollen aber persistent sein
- Datenbanken haben sich bewährt

Einführung
Um was geht es?

2 Arten von Datenbanken:

1. Objektorientierte Datenbanken (OODB)
2. Relationale Datenbanken (rel. DB)

Einführung
Um was geht es?

2 Arten von Datenbanken:

1. Objektorientierte Datenbanken (OODB)
2. Relationale Datenbanken (rel. DB)
   - performanter
   - Basis: mathematische Theorie der Relationen

   $\Rightarrow$ sehr robust

Einführung
Um was geht es?

Wie speichert man Objekte in einer rel. DB?

Einführung
Um was geht es?

Wie speichert man Objekte in einer rel. DB?

- einfache Objekte sind kein Problem
  - Objekt-Attribut $\rightarrow$ Tabellenspalte

Einführung
Um was geht es?

Wie speichert man Objekte in einer rel. DB?

- einfache Objekte sind kein Problem
  - Objekt-Attribut $\rightarrow$ Tabellenspalte
- Wie sieht es aus mit ...
  - ... zusammengesetzten Objekten?
  - ... Objekt-Netzwerken?
  - ... Vererbung?

Einführung
Um was geht es?

Wie speichert man Objekte in einer rel. DB?

- einfache Objekte sind kein Problem
  - Objekt-Attribut $\rightarrow$ Tabellenspalte
- Wie sieht es aus mit ...
  - ... zusammengesetzten Objekten?
  - ... Objekt-Netzwerken?
  - ... Vererbung?
- keine Unterstützung durch rel. DBs

$\Rightarrow$ Objekt-relationale Paradigmen-Unvereinbarkeit

Einführung
Um was geht es?

Lösungen?

Einführung
Um was geht es?

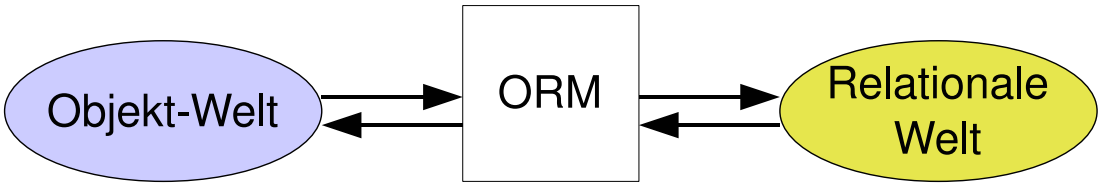
Lösungen?

1. handgeschriebene Persistenz-Schicht
   - schwer wartbar
   - fehleranfällig

Einführung
Um was geht es?

Lösungen?

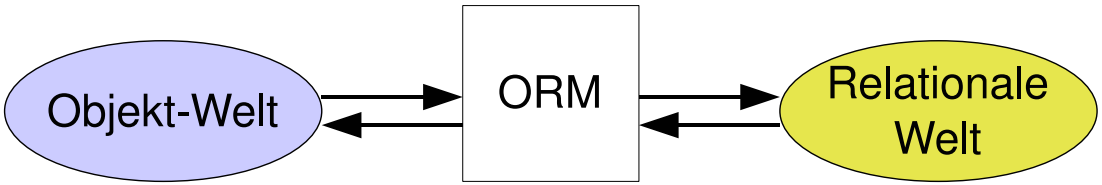
1. handgeschriebene Persistenz-Schicht
   - schwer wartbar
   - fehleranfällig
2. objekt-relationales Mapping (ORM)



Einführung
Um was geht es?

Lösungen?

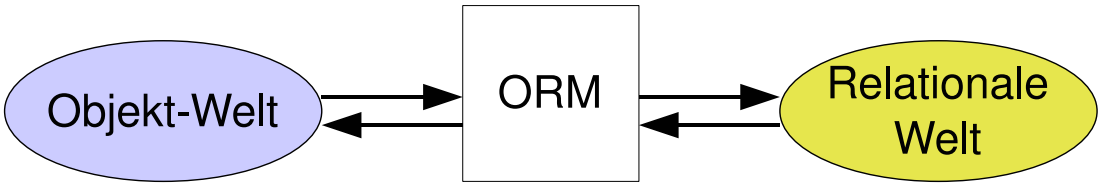
1. handgeschriebene Persistenz-Schicht
   - schwer wartbar
   - fehleranfällig
2. objekt-relationales Mapping (ORM)
   - komerzielle Tools



Einführung
Um was geht es?

Lösungen?

1. handgeschriebene Persistenz-Schicht
   - schwer wartbar
   - fehleranfällig
2. objekt-relationales Mapping (ORM)
   - komerzielle Tools
   - Open Source: Hibernate



Inhaltsverzeichnis

Persistenz-Mechanismen

Was ist Persistenz?

Was ist Persistenz?

Persistenz-Mechanismen

Was ist Persistenz?

Was ist Persistenz?

Definition

Persistenz, von lat. *persistere*: „verharren“

Persistenz-Mechanismen

Was ist Persistenz?

Was ist Persistenz?

Definition

Persistenz, von lat. *persistere*: „verharren“

- Daten dauerhaft speichern
- in Java: Objekte dauerhaft speichern

Persistenz-Mechanismen

- Datenbanken
  - Vorteile sollten bekannt sein
  - gleichzeitiger Zugriff
  - Datenunabhängigkeit
  - ...

Persistenz-Mechanismen

- Datenbanken
  - Vorteile sollten bekannt sein
  - gleichzeitiger Zugriff
  - Datenunabhängigkeit
  - ...
- Serialisierung
  - aktuellen Applikationszustand speichern
  - ganze Netzwerke von Objekten speichern
  - Interface `Serializable`

Persistenz-Mechanismen

- Datenbanken
  - Vorteile sollten bekannt sein
  - gleichzeitiger Zugriff
  - Datenunabhängigkeit
  - ...
- Serialisierung
  - aktuellen Applikationszustand speichern
  - ganze Netzwerke von Objekten speichern
  - Interface `Serializable`

Nachteile:
- muss komplett deserialisiert werden
- selektiver Zugriff auf Objekte nicht möglich
- sehr unperformant in Datenbanken

Persistenz-Mechanismen

- XML
  - selektiver Zugriff möglich

Persistenz-Mechanismen

- XML
  - selektiver Zugriff möglich

  Nachteile:
  - unterstützt keine Vererbung
  - hierarchisch $\neq$ objektorientiert („objekt-hierarchische Paradigmen-Unvereinbarkeit“)

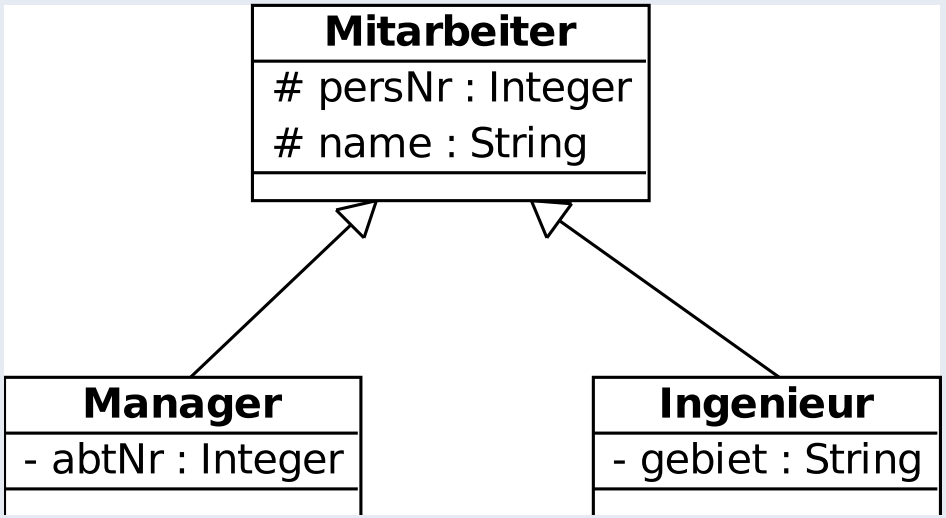
Probleme beim ORM

- Vererbung
- Identität
- Assoziationen
- Datennavigation

Das Problem der Vererbung

Beispiel

Wie kann man folgendes Klassendiagramm in einer rel. DB realisieren?



Das Problem der Vererbung

Ansatz

Eine Tabelle für alle Klassen plus zugehörige Views für die Superklasse und jede Subklasse.

Das Problem der Vererbung

```
create table M (
    persNr integer primary
           key,
    typ char(1) not null,
    name varchar(40),
    abtNr integer,
    gebiet varchar(30)
);

create view Mitarbeiter as
    select persNr, name
    from M;
```

```
create view Manager as
    select persNr, typ, name,
           abtNr
    from M
    where Typ = 'M';

create view Ingenieur as
    select persNr, typ, name,
           gebiet
    from M
    where Typ = 'I';
```

Das Problem der Identität

1. Java
   - Identität (prüfen mit ==)
   - Gleichheit (prüfen mit `equals()`)

Das Problem der Identität

1. Java
   - Identität (prüfen mit ==)
   - Gleichheit (prüfen mit `equals()`)
2. rel. DB
   - Identität = Wert des Primärschlüssels einer Tabelle

Das Problem der Identität

Beispiel (abgespecktes Online-Diskussions-Forum)

```
1  create table Benutzer (
2      loginname varchar(12) primary key,
3      vorname varchar(30),
4      nachname varchar(30),
5      email varchar(60)
6  );
7
8  create table Beitrag (
9      betreff varchar(20),
10     text varchar(100),
11     loginname varchar(12) references Benutzer (
           loginname)
12 );
```

Was tun, wenn der Login-Name sich ändert?

Das Problem der Identität

- **Problem:** `loginname` ändern $\Rightarrow$ andere Tabellen auch ändern
- Problem nicht direkt lösbar
- Workaround/Lösung

Das Problem der Identität

- **Problem:** `loginname` ändern $\Rightarrow$ andere Tabellen auch ändern
- Problem nicht direkt lösbar
- Workaround/Lösung
  - Surrogatschlüssel (Ersatzschlüssel) einführen
  - auch in Java-Klassen mit aufnehmen
  - Getter- und Setter-Methoden definieren

Das Problem der Identität

Lösungsmöglichkeit:

```
1  create table Benutzer (
2      benutzer_id bigint not null primary key,
3      loginname varchar(12) unique,
4      vorname varchar(30),
5      nachname varchar(30),
6      email varchar(60)
7  );
8
9  create table Beitrag (
10     beitrag_id bigint not null primary key,
11     betreff varchar(20),
12     text varchar(100),
13     benutzer_id bigint references Benutzer(benutzer_id)
14 );
```

Mit Assoziationen verbundene Probleme

Java: Assoziation = Referenz (Zeiger) auf Objekt

rel. DB: Assoziation = Kopie von Fremdschlüssel

Mit Assoziationen verbundene Probleme

Java: Assoziation = Referenz (Zeiger) auf Objekt

rel. DB: Assoziation = Kopie von Fremdschlüssel

- *many-to-many*-Assoziation in Java

```
1 public class Buch {
2     private Set autoren;
3 }
4
5 public class Autor {
6     private Set buecher;
7 }
```

Mit Assoziationen verbundene Probleme

Java: Assoziation = Referenz (Zeiger) auf Objekt

rel. DB: Assoziation = Kopie von Fremdschlüssel

- *many-to-many*-Assoziation in Java

```
1 public class Buch {
2     private Set autoren;
3 }
4
5 public class Autor {
6     private Set buecher;
7 }
```

- *many-to-many*-Assoziation in rel. DB
  - Nur über Zwischentabelle möglich

Das Problem der Datennavigation

- Java
  - von Objekt zu Objekt navigieren.
  - Bsp: `telefonbuch.getPerson("Meier").getTelNr()`

Das Problem der Datennavigation

- Java
  - von Objekt zu Objekt navigieren.
  - Bsp: `telefonbuch.getPerson("Meier").getTelNr()`
- rel. DB
  - Navigation spielt keine Rolle
  - beliebige Tabellen-Joins möglich

Das Problem der Datennavigation

Lösungsversuch (Erster Gedanke)

Bei jedem Zugriff auf neues Objekt entsprechenden Datensatz aus der Datenbank holen.

Das Problem der Datennavigation

Lösungsversuch (Erster Gedanke)

Bei jedem Zugriff auf neues Objekt entsprechenden Datensatz aus der Datenbank holen.

- Probleme
  - macht Vorteile des rel. DB-Systems kaputt (Joins)
  - *n+1 selects*-Problem $\Rightarrow$ sehr ineffizient
- effizienter: Tabellen-Joins
- Man muss vorher wissen, auf was man zugreift

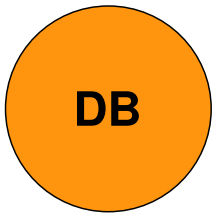
Inhaltsverzeichnis

Grundlagen von Hibernate

Wo genau ist Hibernate anzuordnen?

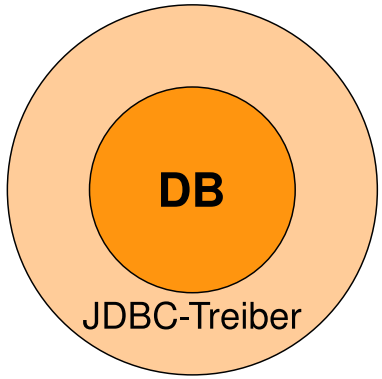
Grundlagen von Hibernate

Wo genau ist Hibernate anzuordnen?



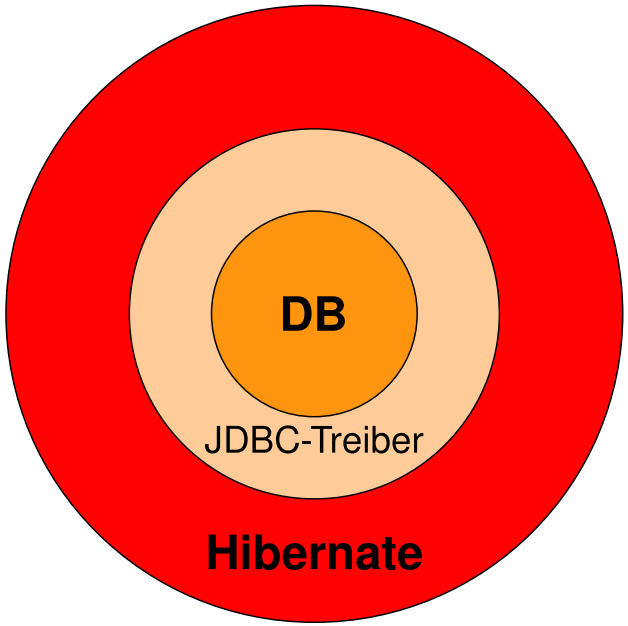
Grundlagen von Hibernate

Wo genau ist Hibernate anzuordnen?



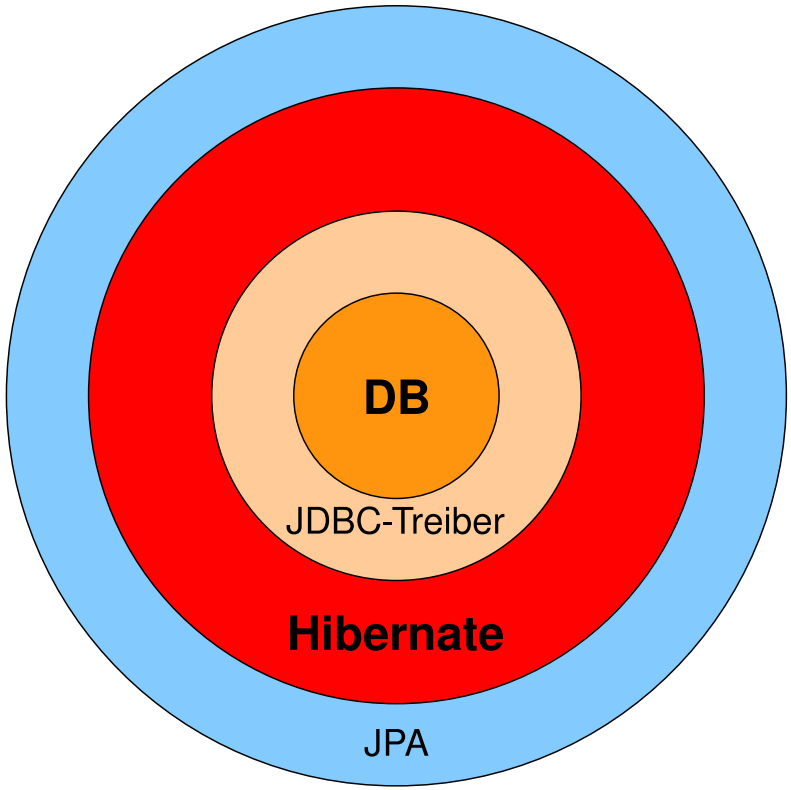
Grundlagen von Hibernate

Wo genau ist Hibernate anzuordnen?



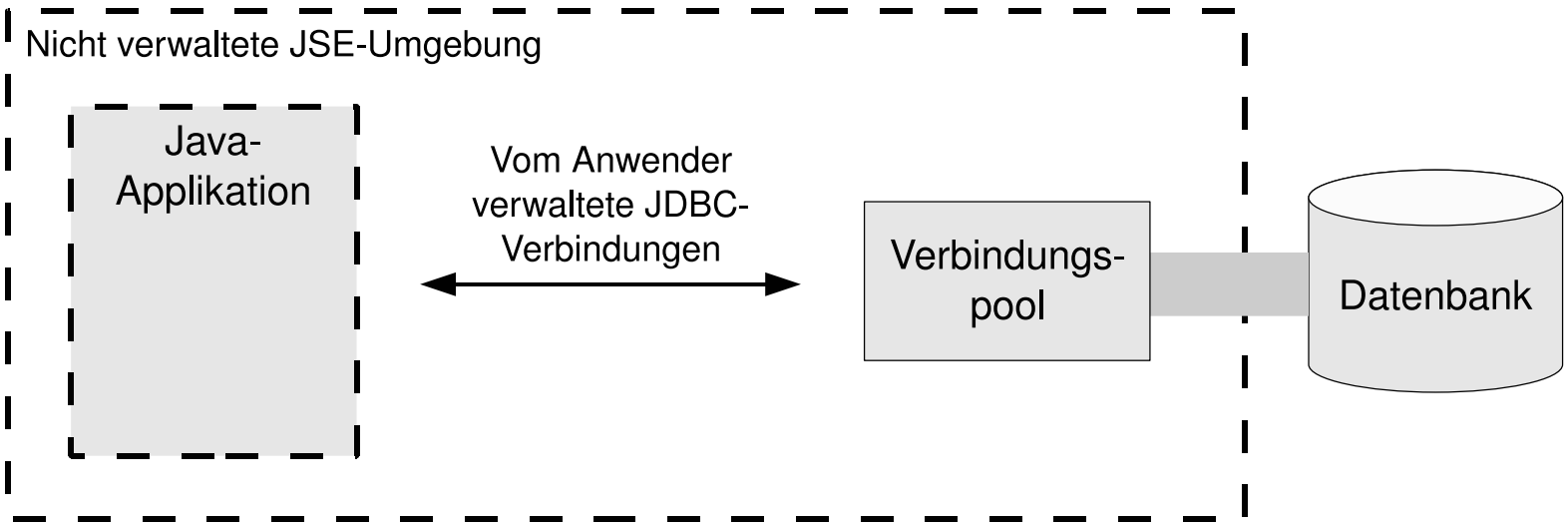
Grundlagen von Hibernate

Wo genau ist Hibernate anzuordnen?



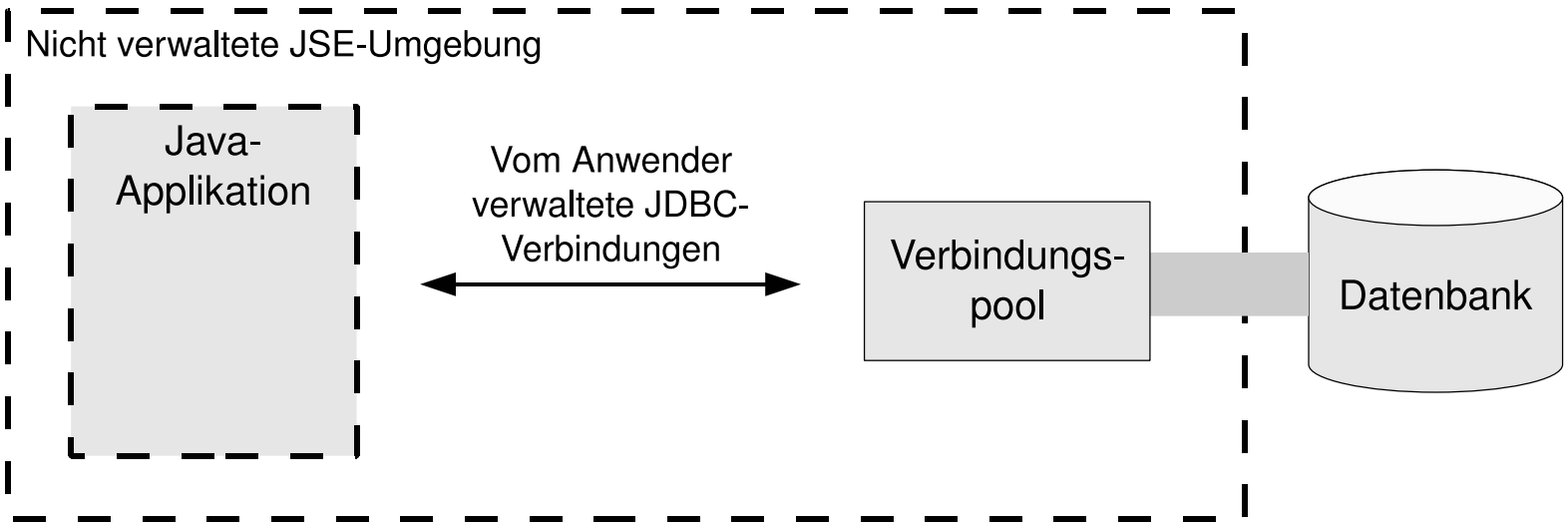
Verbindungspooling ohne Hibernate

- nicht jedes mal eine neue Verbindung aufbauen
- vorhandene Verbindungen nutzen
- Verbindungen nicht schließen, sondern an Pool zurückgeben

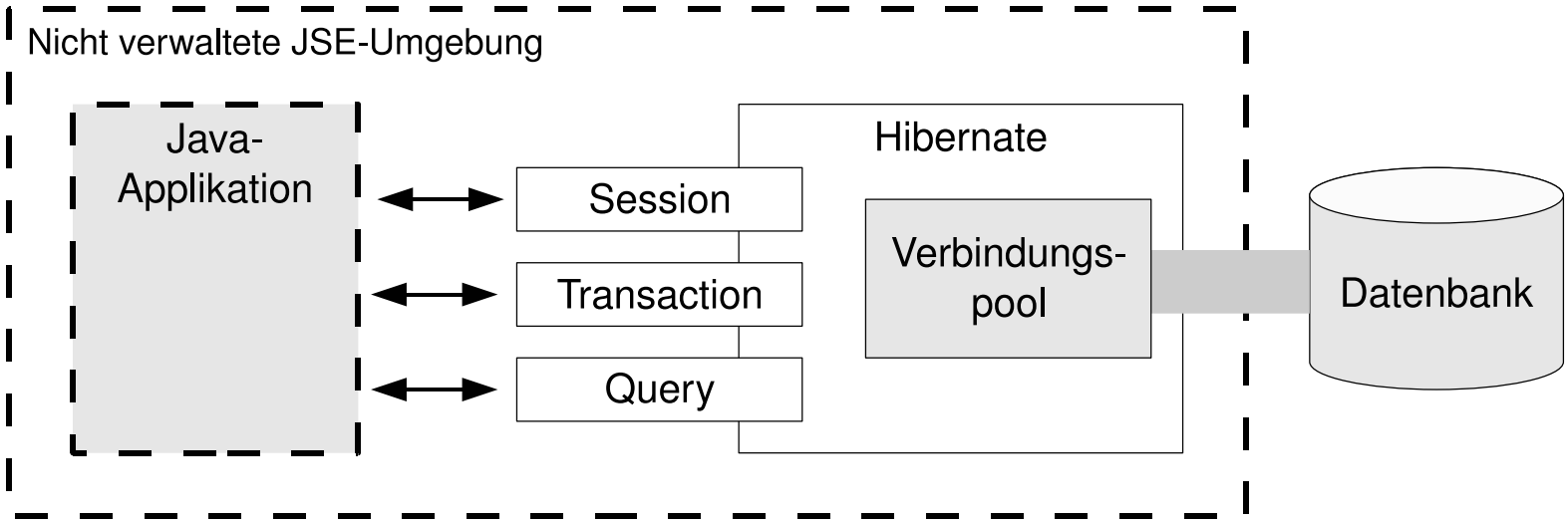


Verbindungspooling ohne Hibernate

- nicht jedes mal eine neue Verbindung aufbauen
- vorhandene Verbindungen nutzen
- Verbindungen nicht schließen, sondern an Pool zurückgeben
- Warum?
  - Verbindungsaufbau kostet Zeit und Ressourcen
  - aufwändig für DB-System



Verbindungspooling mit Hibernate



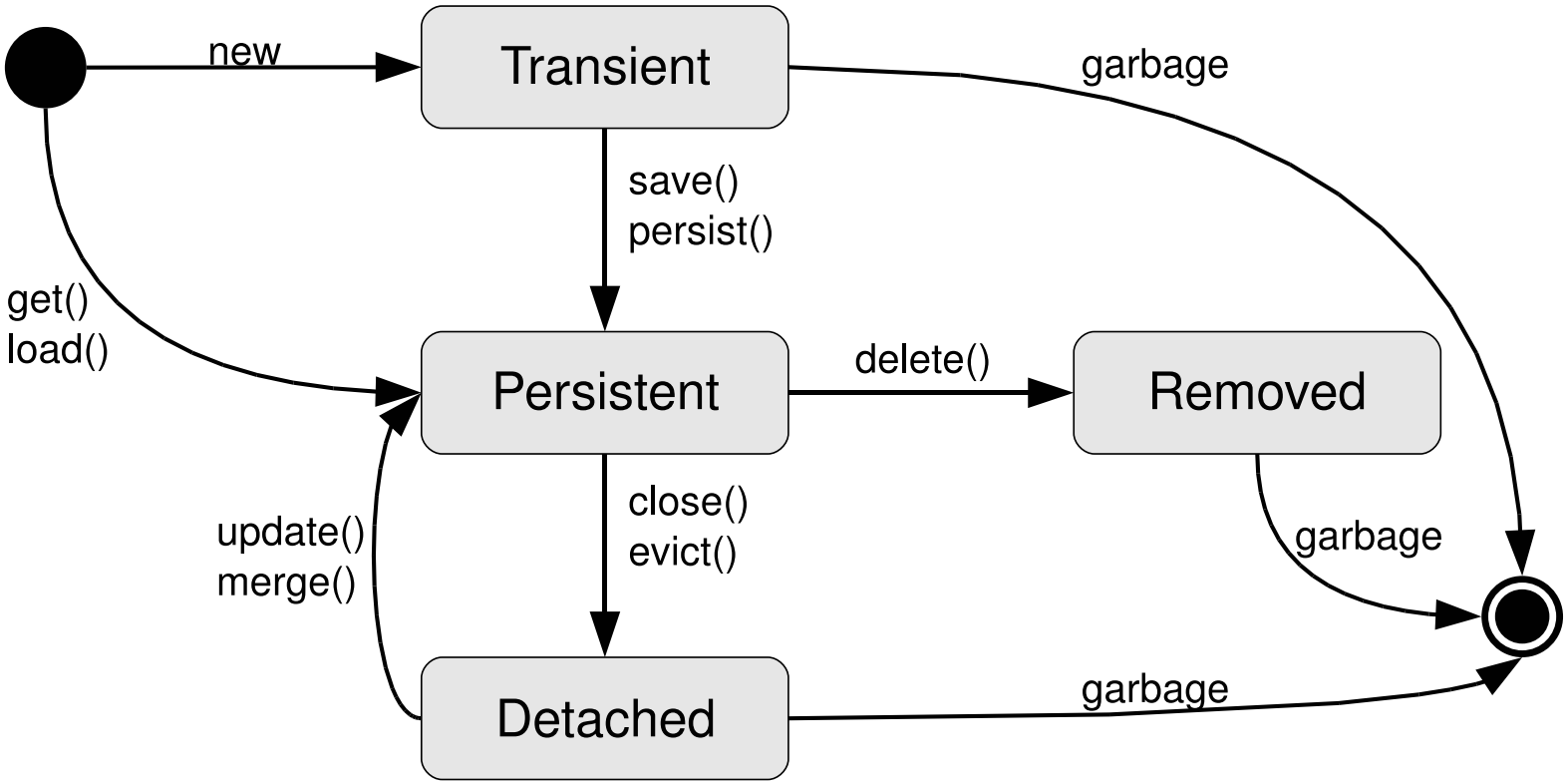
- Hibernate übernimmt die Aufgabe des Poolings
- Hibernate stellt verschiedene APIs zur Verfügung

  Session: repräsentiert eine Arbeitseinheit der Datenbank

  Transaction: Transaktionsgrenzen setzen

  Query: Abfragen formulieren

Der Persistenz-Kontext



Inhaltsverzeichnis

Ein Hibernate-Projekt durchführen

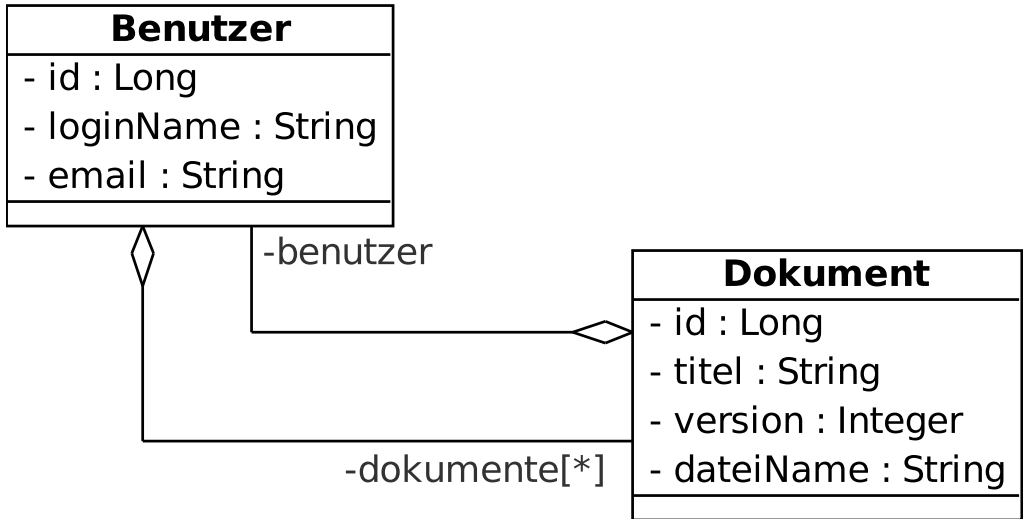
- Entwicklungsumgebung:
  - Eclipse 3.3 Europa
  - PostgreSQL 8.3
  - Hibernate Core 3.2.6

Ein Hibernate-Projekt durchführen

- Datenbank einrichten
- Neues Projekt in Eclipse erstellen
- relevante Hibernate-JAR-Dateien $\rightarrow$ `lib`-Verzeichnis
- PostgreSQL-JDBC-Treiber (JAR) $\rightarrow$ `lib`-Verzeichnis
- JAR-Dateien dem *Java Build Path* hinzufügen

Das Klassenmodell

- Folgendes Klassen-Modell („Dokumentenverwaltung“) liegt zugrunde:



- Zu den Klassen:
  - einfache POJOs
  - Getter- und Setter-Methoden
  - Default-Konstruktor

Das Klassenmodell

Die Klasse `Benutzer`

```
1  public class Benutzer {
2      private Long id;
3      private String loginName;
4      private String email;
5      private Set dokumente = new HashSet();
6
7      public Benutzer() {
8      }
9
10     // Getter und Setter, weitere Konstruktoren
11
12 }
```

Das Klassenmodell

Die Klasse `Dokument`

```
1  public class Dokument {
2      private Long id;
3      private String titel;
4      private Integer version;
5      private String dateiName;
6      private Benutzer benutzer;
7
8      public Dokument() {
9      }
10
11     // Getter und Setter, weitere Konstruktoren
12
13 }
```

Die Mapping-Dateien

Die Datei `Benutzer.hbm.xml` (1)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3      "-//Hibernate/Hibernate Mapping DTD//EN"
4      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5  <hibernate-mapping package="de.fhgiessen.dbs">
6      <class name="Benutzer" table="benutzer">
7          <id name="id" column="benutzer_id">
8              <generator class="increment"/>
9          </id>
10
11         <property name="loginName" column="loginname"/>
12         <property name="email" column="email"/>
```

Die Mapping-Dateien

Die Datei `Benutzer.hbm.xml` (2)

```
1
2           <set
3               name="dokumente">
4               <key column="benutzer_id"/>
5               <one-to-many class="Dokument"/>
6           </set>
7       </class>
8   </hibernate-mapping>
```

Die Mapping-Dateien

Die Datei `Dokument.hbm.xml`

```
1  <!-- Header -->
2  <hibernate-mapping package="de.fhgiessen.dbs">
3      <class
4          name="Dokument" table="dokument">
5          <id name="id" column="dokument_id">
6              <generator class="increment"/>
7          </id>
8
9          <property name="titel" column="titel"/>
10         <property name="version" column="version"/>
11         <property name="dateiName" column="dateiname"/>
12
13         <many-to-one name="benutzer" column="benutzer_id"
14             insert="false" update="false"/>
15     </class>
16 </hibernate-mapping>
```

Hibernate konfigurieren

Die Datei `hibernate.cfg.xml` (1)

```
1  <!DOCTYPE hibernate-configuration SYSTEM "http://hibernate.
       sourceforge.net/hibernate-configuration-3.0.dtd">
2  <hibernate-configuration>
3      <session-factory>
4          <property name="hibernate.connection.driver_class">
5              org.postgresql.Driver
6          </property>
7          <property name="hibernate.connection.url">
8              jdbc:postgresql://localhost/dokuverw
9          </property>
10         <property name="hibernate.connection.username">
11             postgres
12         </property>
13         <property name="hibernate.connection.password">
14             postgres
15         </property>
```

Hibernate konfigurieren

Die Datei `hibernate.cfg.xml` (2)

```
1         <property name="hibernate.dialect">
2             org.hibernate.dialect.PostgreSQLDialect
3         </property>
4
5         <!-- Tabellen automatisch erzeugen lassen -->
6         <property name="hibernate.hbm2ddl.auto">update</property>
7
8         <!-- Verwenden des C3P0 Connection Pool Providers -->
9         <property name="hibernate.c3p0.min_size">5</property>
10        <property name="hibernate.c3p0.max_size">20</property>
11        <property name="hibernate.c3p0.timeout">300</property>
12        <property name="hibernate.c3p0.max_statements">50</
              property>
13        <property name="hibernate.c3p0.idle_test_period">3000</
              property>
```

Hibernate konfigurieren

Die Datei `hibernate.cfg.xml` (3)

```
1
2           <!-- Schoenes, sauberes SQL bei stdout ausgeben -->
3           <property name="show_sql">true</property>
4           <property name="format_sql">true</property>
5
6           <!-- Liste der XML-Mapping-Dateien -->
7           <mapping resource="de/fhgiessen/dbs/Benutzer.hbm.xml" />
8           <mapping resource="de/fhgiessen/dbs/Dokument.hbm.xml" />
9       </session-factory>
10  </hibernate-configuration>
```

Das Hauptprogramm

Die Klasse `MainKlasse` (1)

```
1  public class MainKlasse {
2      public static void main(String[] args) {
3          // 1. Unit of Work beginnen
4          Session session = HibernateUtil.getSessionFactory().
               openSession();
5          Transaction tx = session.beginTransaction();
6
7          // Dokumente anlegen und in Persistenz-Kontext aufnehmen
8          Dokument dok1 = new Dokument("Reise", new Integer(1), "
               Reise.xml");
9          Dokument dok2 = new Dokument("Test", new Integer(6), "
               test_doc.doc");
10         Dokument dok3 = new Dokument("Abrechnung", new Integer(2),
                "abr.pdf");
11         session.save(dok1);
12         session.save(dok2);
13         session.save(dok3);
```

Das Hauptprogramm

Die Klasse `MainKlasse` (2)

```
1          // Benutzer anlegen mit Dokument(en)
2          // und in Persistenz-Kontext aufnehmen
3          Benutzer ben1 = new Benutzer("peter", "peter@web.de");
4          session.save(ben1);
5          ben1.getDokumente().add(dok1);
6
7          Benutzer ben2 = new Benutzer("hilde", "hilde@gmx.at");
8          session.save(ben2);
9          ben2.getDokumente().add(dok2);
10         ben2.getDokumente().add(dok3);
11
12         // Dokument aendern
13         dok1.setTitel("Geaendert");
14
15         // Transaktion bestaetigen und Session schliessen
16         tx.commit();
17         session.close();
```

Das Hauptprogramm

Die Klasse `MainKlasse` (3)

```
1     // 2. Unit of Work beginnen
2     session = HibernateUtil.getSessionFactory().openSession();
3     tx = session.beginTransaction();
4
5     // Benutzer ausgeben
6     System.out.println("--- Benutzer ------------------------");
7     List benutzer = session.createQuery("from Benutzer").list();
8     for (Benutzer b : (List<Benutzer>) benutzer) {
9         // Benutzername, E-Mail-Adresse und ID ausgeben
10        System.out.println("  * " + b.getLoginName() +
11                "  (" + b.getEmail() + ") [" + b.getId() + "]");
12        // Dokumente des Benutzers ausgeben
13        System.out.println("     * Dokumente:");
14        for (Dokument d : (Set<Dokument>)b.getDokumente()) {
15            // ...
16        }
17    }
```

Das Hauptprogramm

Die Klasse `MainKlasse` (4)

```
1      // Dokumente ausgeben
2      System.out.println();
3      System.out.println("--- Dokumente ----------------------");
4      List dokumente = session.createQuery("from Dokument").list();
5      for (Dokument d : (List<Dokument>)dokumente) {
6          // Dokumenttitel, -dateiname und ID ausgeben
7          System.out.println("  * " + d.getTitel() +
8              " (" + d.getDateiName() + ") [" + d.getId() + "] {" +
9              d.getBenutzer().getLoginName() + "}");
10     }
11     // Transaktion bestaetigen und Session schliessen
12     tx.commit();
13     session.close();
14     // Programm beenden
15     HibernateUtil.shutdown();
16 }
17 }
```

Erzeugtes Datenbankschema

Folgendes Datenbankschema wurde von Hibernate erzeugt:

```
1  create table benutzer (
2      benutzer_id bigint not null primary key,
3      loginname varchar(255),
4      email varchar(255)
5  );
6
7  create table dokument (
8      dokument_id bigint not null primary key,
9      titel varchar(255),
10     version integer,
11     dateiname varchar(255),
12     benutzer_id bigint references benutzer (benutzer_id)
13 );
```

Ausgabe des Hauptprogramms

Ausgabe von `MainKlasse`:

```
--- Benutzer -------------------------
  * peter (peter@web.de) [37]
     * Dokumente:
        * Geaendert (Reise.xml)
  * hilde (hilde@gmx.at) [38]
     * Dokumente:
        * Test (test_doc.doc)
        * Abrechnung (abr.pdf)

--- Dokumente ------------------------
  * Geaendert (Reise.xml) [1] {peter}
  * Test (test_doc.doc) [2] {hilde}
  * Abrechnung (abr.pdf) [3] {hilde}
```

Zusammenfassung

Zusammenfassung

Zusammenfassung

Zusammenfassung

Zusammenfassung

Literaturangaben

BAUER, Christian ; KING, Gavin.

Java Persistence mit Hibernate.

Hanser Fachbuch, 2007 – ISBN 3-44640-9416.

Fragen?

Danke für die Aufmerksamkeit!