
HIBERNATE

Was ist Hibernate

- Hibernate ist ein Open-Source-Java-Persistenz-Framework für relationale Datenbanken
- Persistenz-Frameworks: kapseln und abstrahiert den Zugriff auf eine Speicherschicht -> Zwischenschicht zwischen flüchtigen und nichtflüchtigen Speicher
- es existieren viele Persistenz-Frameworks für die unterschiedlichsten Einsatzzwecke und Speichermedien (z.B. XML, DB)

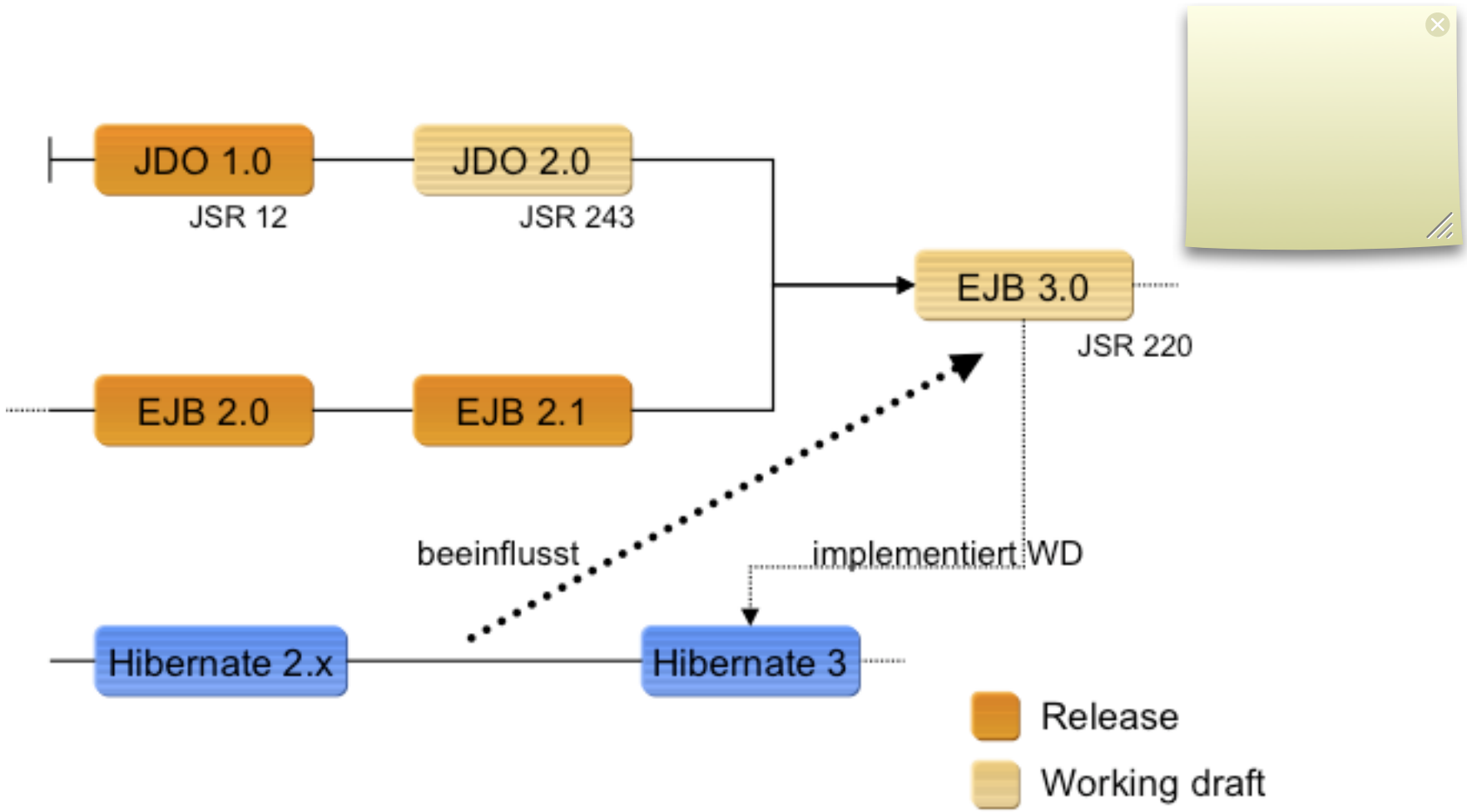
Hibernate - Features

- Open-Source
- Transparente Persistence (keine Interfaces, keine Ober-Klassen)
- Caching
  - auch in verteilten Systemen
- verzögerte Ausführung
  - INSERT und UPDATE werden erst bei Bedarf ausgeführt
  - Datenbank wird so z.B. durch Abbruch einer Transaktion erst gar nicht belastet
- keine unnötigen Updates
  - aus einer Liste werden z.B. nur geänderte Objekte wirklich in der Datenbank aktualisiert

Hibernate - Features 2

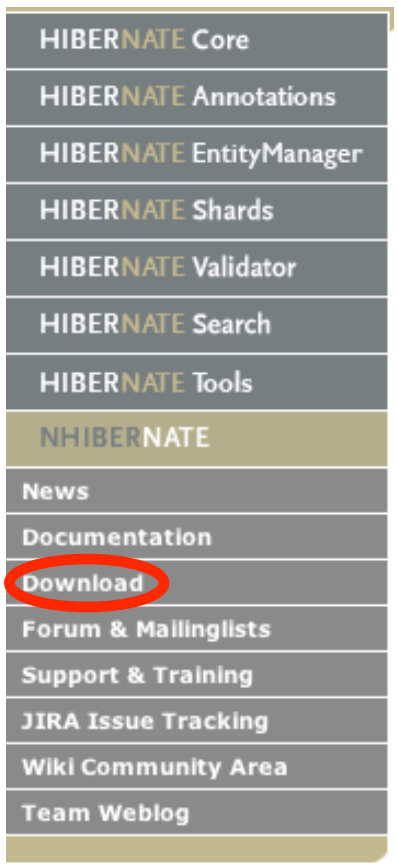
- Zusammenfassung von Update-Statements
- vorausschauendes Laden möglich z.B. in Schleifen
- dynamisches-Nachladen
- dynamische SQL-Abfragen (Criteria-Query)
- Nutzung von Java-Annotationen

JDO, Hibernate, javax.persistence, EJB 3.0



Hibernate - Download

- Download von www.hibernate.org
- Hibernate Core und Hibernate Annotations



Binary Releases

Package	Version	Release date	Category	
Hibernate Core	3.2.4.SP1	09.05.2007	Production	*Download*
Hibernate Annotations	3.3.0 GA	20.03.2007	Production	*Download*
Hibernate EntityManager	3.3.1 GA	29.03.2007	Production	*Download*
Hibernate Validator	3.0.0 GA	20.03.2007	Production	*Download*
Hibernate Search	3.0.0 Beta2	31.05.2007	Development	*Download*
Hibernate Shards	3.0.0 Beta1	20.03.2007	Development	*Download*
Hibernate Tools	3.2.0 Beta9	13.01.2007	Development	*Download*
NHibernate	1.2.0.GA	03.05.2007	Production	*Download*
NHibernate Extensions	1.0.4	24.01.2007	Production	*Download*
JBoss Seam	1.2.0 Patch1	28.02.2007	Production	*Download*

Hibernate - Installation

- Pakete entpacken
- zum Classpath hinzufügen:
  - hibernate3.jar
  - hibernate-annotations.jar
  - und alle benötigten Pakete aus den jeweiligen lib-Ordnern (im Zweifelsfall alle)
  - eventuell JDBC-Treiber

Hibernate - Konfiguration

Freie Universität Berlin

- hibernate.cfg.xml im classpath

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>

    <!-- Database connection settings -->
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/db_name</property>
    <property name="connection.username">user</property>
    <property name="connection.password">password</property>
    <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>

    <!-- Enable the second-level cache  -->
    <property name="cache.provider_class">org.hibernate.cache.EhCacheProvider</property>

    <!-- Update the database schema on startup create/validate/create-drop -->
    <property name="hbm2ddl.auto">update</property>

    </session-factory>
</hibernate-configuration>
```

Hibernate - Objekte

- SessionFactory
  - Erzeugung teuer->möglichst einmalig in Anwendung
  - threadsafe
  - optionalen Second-Level Cache
  - Cache kompilierte Mappings
- Session
  - Methoden zum laden, speichern, aktualisieren, löschen und suchen von Objekten
  - First-Level Cache
- Transaction
  - Abstrahiert JDBC, JTA, CORBA Tx
  - atomare Arbeitseinheit
  - jede DB-Operation muss innerhalb einer Transaction erfolgen (auch lesende)

speicherbare Objekte - POJO

Freie Universität Berlin

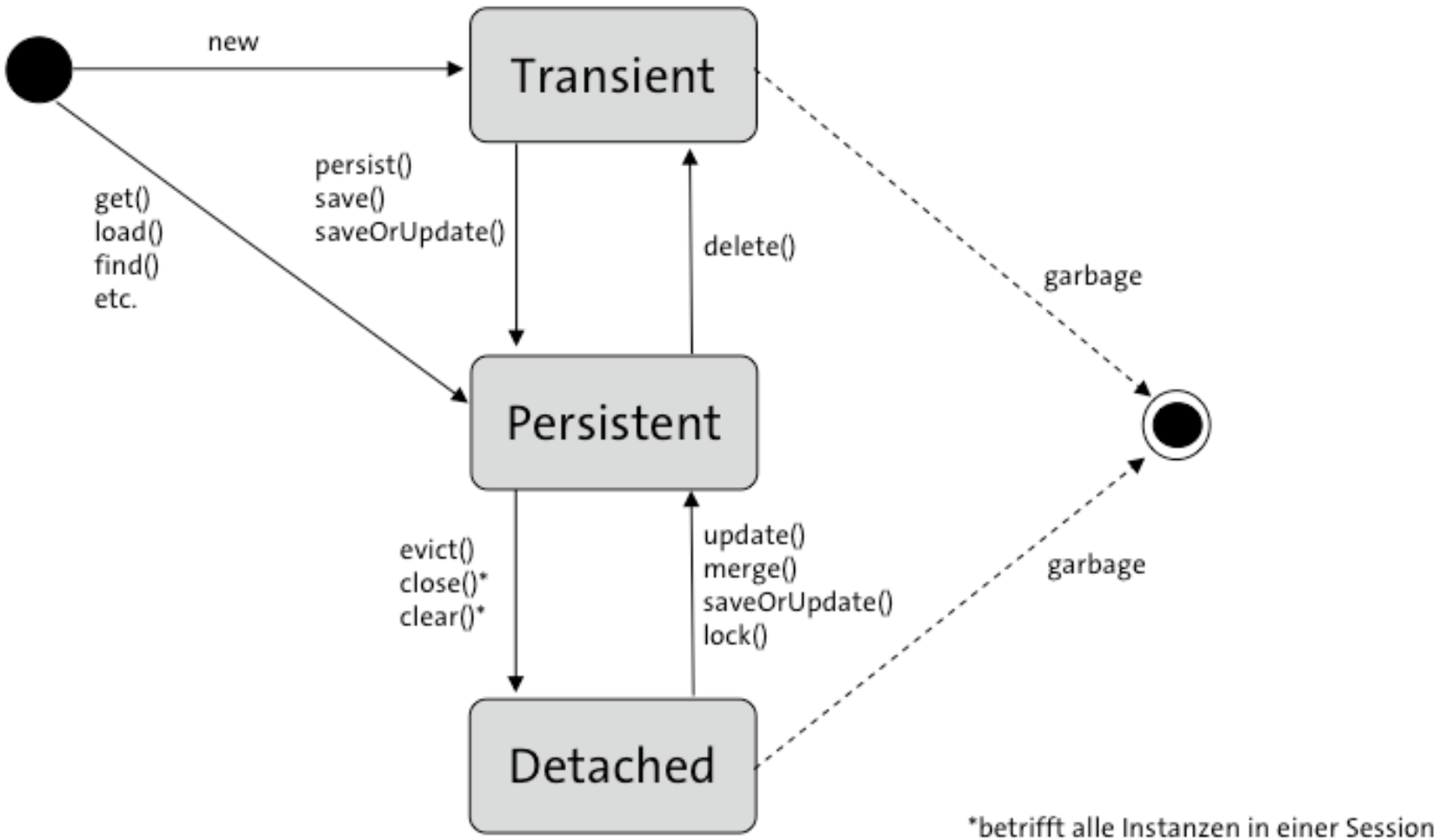
- Objektzustand kann gespeichert werden, wenn:
  1. eine Default-Konstruktor existiert (kann private sein)
  2. ein ID-Feld angegeben ist (optional)
  3. keine final Klassen deklaration (optional)
  4. getters/setters für Felder (optional)

Lebenszyklus

- Transient
  - Objekt befindet sich nicht in der DB und war es auch nie
- Persistent
  - Objekt befindet sich in der Datenbank
  - gehört zu einer Session
  - synchron mit der Datenbank
  - hat einen Primärschlüssel
- Detached
  - Objekt befindet sich in der Datenbank
  - hat einen Primärschlüssel
  - Benutzung außerhalb einer Session (getrennt von DB)
  - nicht synchron mit der Datenbank
  - kann aber synchronisiert werden
  - lazy-loading funktioniert in diesem Zustand nicht!!!

Lebenszyklus

Lebenszyklus



Beispiel

Hiberante Beispiel - Mapping

Freie Universität Berlin

- Mapping: Hibernate-Anweisungen zum Abbilden von Java-Objekten auf die Datenbank

```
@Entity
public class Book {

        @Id
        private String title;

        private String description;
```

Hiberante Beispiel - Mapping 2

- Ergänzung der Hibernate.cfg.xml um den Eintrag für das Mapping
- Im Falle der Annotationen: Klassenname

```
<mapping class="hibernate.Book" />
```

Hiberante Beispiel - Code

```
SessionFactory sessionFactory =
   new AnnotationConfiguration().configure().buildSessionFactory();

   ....
Session session = sessionFactory.openSession();
Transaction transaction = null;
try {
   transaction = session.beginTransaction();
   Book book = new Book();
   book.setName(„Hallo“);
   session.persist(book);
   transaction.commit();
} catch (Exception e) {
   if (transaction!=null) {transaction.rollback();}
} finally {
  session.close();
}
```

Mapping

Basic - Mapping

Freie Universität Berlin

```
@Entity
@Table(name="books")
public class Book {

        @Id
        @GeneratedValue
        private int id;

        @Column(length=1024, nullable=false)
        private String title;

        @Type(type = "text")
        private String description;

        @Transient
        private boolean locked;

        @Temporal(TemporalType.DATE)
        private Date published;

        @Lob //wird in dieser Konfiguration nicht Lazy geladen
        private byte[] cover;
```

Collection - Mapping

- z.B. OneToMany, cascadierend, kein verzögertes laden

```
@OneToMany(cascade=CascadeType.ALL, fetch=FetchType.EAGER)
private Collection<Author> author;
```

- z.B. OneToMany, Erzeugung cascadierend, verzögertes laden, geordnet nach Name des Autors

```
@OneToMany(cascade=CascadeType.PERSIST)
@OrderBy("name")
private Collection<Author> author;
```

- viele weitere Konfigurationsmöglichkeiten ... OneToOne, OneToMany, Bidirectional/Unidirectional

Vererbungs - Mapping

Freie Universität Berlin

- Angabe der Vererbungsstrategie als Annotation in der Oberklasse z.B. `@Inheritance(strategy = InheritanceType.JOINED)`
- Table per concrete class (`InheritanceType.TABLE_PER_CLASS`)
  - Klasse und Unterklassen jeweils in eigener Tabelle
- Table per class hierarchy (`InheritanceType.SINGLE_TABLE` - default, kann auch entfallen)
  - Klasse und Unterklassen jeweils in einer Tabelle
  - in zusätzlicher Spalte wird Klassenzugehörigkeit gespeichert
- Table per subclass (`InheritanceType.JOINED`)
  - Klasse und Unterklassen jeweils in eigener Tabelle
  - Tabelle der Unterklassen enthält nur nicht vererbte Werte

Vererbungs - Mapping 2

- Einfaches Beispiel

```
@Entity
public class EBook extends Book {

        String drmKey;

}

@Entity
public class Book {

    ....
}
```

Suche

Suche - HQL

- Suche alle Bücher mit gegebenen Titel

```
Query query = session.createQuery("from Book where title
like :title");
query.setString("title", "%"+title+"%");
List<Book> books = query.list();
```

- Seitenaufteilung

```
query.setFirstResult(10);
query.setMaxResults(10);
```

- Suche nach Büchern und Autor

```
Query query = session.createQuery("select b from Book as b
join b.authors as a where b.title like :query or a.name
like :query");
query.setString("query", "%" + queryString + "%");
```

Suche - Criteria Query

Freie Universität Berlin

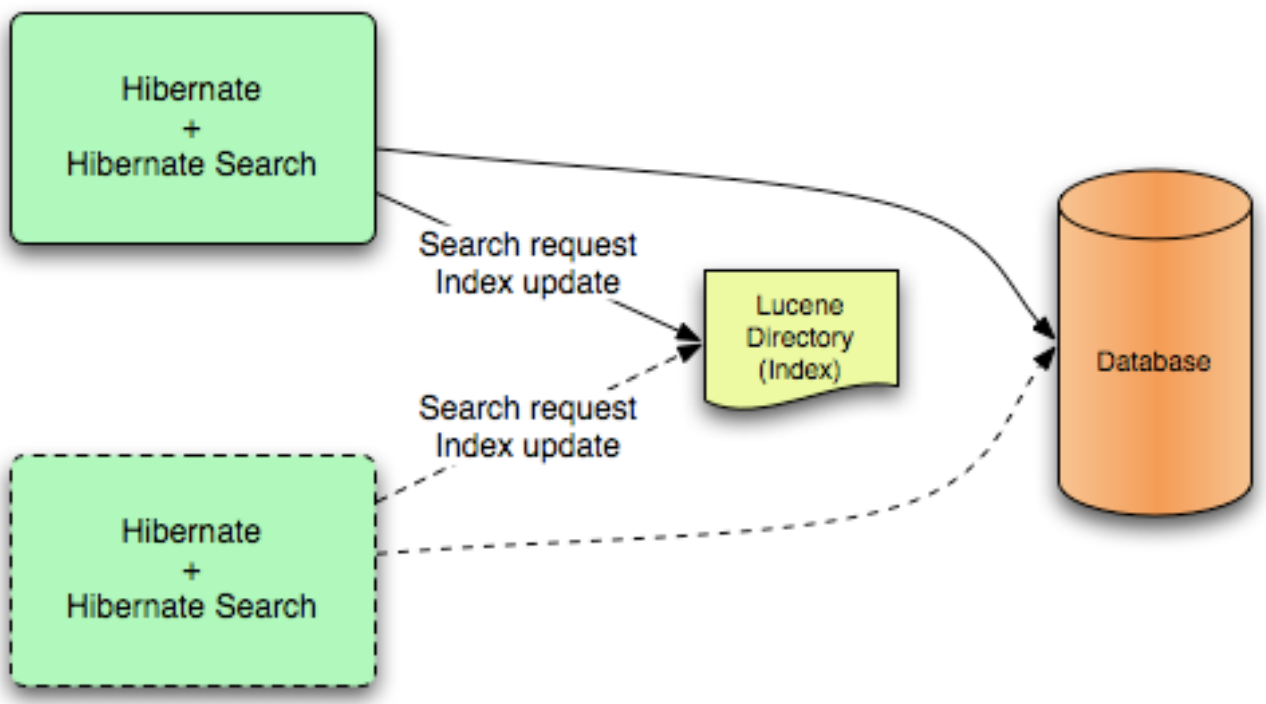
- Suche nach Buchtitel mit Criteria Query

```
Criteria criteria = session.createCriteria(Book.class);
criteria.add(Restrictions.ilike("title", queryString,
MatchMode.ANYWHERE)));

criteria.setFirstResult(10);
criteria.setMaxResults(10);

criteria.addOrder(Order.asc("title"));
List<Book> books = criteria.list();
```

Suche - Lucene

Lucene

- Volltext-Indexierer
  - AND, OR, NOT
  - Wildcards
  - Unscharfe Suche
  - Suche in einzelnen Feldern



Lucene - Installation

- Download von Hibernate Search

Binary Releases

Package	Version	Release date	Category	
Hibernate Core	3.2.4.SP1	09.05.2007	Production	*Download*
Hibernate Annotations	3.3.0 GA	20.03.2007	Production	*Download*
Hibernate EntityManager	3.3.1 GA	29.03.2007	Production	*Download*
Hibernate Validator	3.0.0 GA	20.03.2007	Production	*Download*
Hibernate Search	3.0.0 Beta2	31.05.2007	Development	*Download*
Hibernate Shards	3.0.0 Beta1	20.03.2007	Development	*Download*
Hibernate Tools	3.2.0 Beta9	13.01.2007	Development	*Download*
NHibernate	1.2.0.GA	03.05.2007	Production	*Download*
NHibernate Extensions	1.0.4	24.01.2007	Production	*Download*
JBoss Seam	1.2.0 Patch1	28.02.2007	Production	*Download*

Lucene - Konfiguration

- hibernate.cfg.xml

```
...
<property name="hibernate.search.default.directory_provider">
    org.hibernate.search.store.FSDirectoryProvider
</property>

<property name="hibernate.search.default.indexBase">
    /var/cache/index
</property>
...
```

Lucene - Mapping

- Annotationen (Indexed, DocumentId und Field)

```
@Entity
@Indexed
public class Pojo {

        @Id
        @DocumentId
        private String name;

        @Field(name="description", index=Index.TOKENIZED, store=Store.YES)
        private String description;
```

Lucene - Suche

Freie Universität Berlin

```
org.apache.lucene.queryParser.QueryParser
    qp = new MultiFieldQueryParser(new {"title", "description"},new StandardAnalyzer());

org.apache.lucene.search.Query
    luceneQuery = parser.parse( "title:Hibernate Or description:persistence" );

 org.hibernate.Query fullTextQuery = fullTextSession.createFullTextQuery( luceneQuery );
List result = fullTextQuery.list();
```

Gut zu wissen ...

Lazy Property fetching

1. Problem: dynamisches Nachladen einzelner Felder
2. Lösung:
   - java.sql.Blob
   - @Basic(fetch = FetchType.LAZY) ODER @Lob
     - erfordert Modifikation der Klassen
     - es gibt einen ANT-Task von Hibernate für diese Aufgabe
   - weitere Möglichkeiten z.B. @Proxy

Open-View-Pattern

1. Problem: Geschlossene Sessions und dynamisches nachladen, z.B. View hat keine Möglichkeit mehr Session zu schließen
2. Lösung:
   - auf dynamisches Nachladen verzichten
   - Hibernate.initialize(books.getAuthors)
   - bei Webserver schließen der Session in einem ServletFilter

Vielen Dank

Quellen

- www.hibernate.org
- http://www.doag.org/pub/docs/sig/development/2005-11/hibernate3.pdf
- http://www.oio.de/m/konf/jax2005/Persistenz%20mit%20Hibernate%20-%20final.pdf
- http://www.andrena.de/Entwicklertag/2005/Downloads/POJO-Persistenz.pdf
- http://www.slideshare.net/obelikan/workshop-zu-hibernate-322-ga/

- http://www.devarticles.com/c/a/Java/Managing-Transactions-with-Hibernate/
- http://www.javalobby.org/java/forums/t20533.html