JPA und das Hibernate Framework

Einleitung:

In unseren bisherigen JDBC-Übungen haben Sie vom Zugriff auf die Datenbank über das Befüllen der Objekte mit Daten aus der Datenbank selber implementieren müssen.

Schön wäre es, wenn ein Framework einen Teil dieser Aufgaben übernehmen würde.

Und hier kommt das **Hibernate Framework** ins Spiel. Es übernimmt Teile des Zugriffs auf die Datenbank. Dabei spielen Transaktionen eine zentrale Rolle.
Dabei wird das Framework von **JPA** unterstützt. JPA dient als Schnittstelle für die Zuordnung von Objekten in der Applikation zu Entitäten in der Datenbank.

Es gibt verschiedene Möglichkeiten Hibernate, JPA und Transaktionen einzusetzen.

Beispiele:

- Java Projekt mit Maven, Verwendung von JPA, Hibernate Session und Transaktionen.
- Java Projekt mit Maven, Verwendung von JPA, Hibernate EntityManager.
- Spring Boot Projekt, Verwendung von JPA, Hibernate EntityManager, Transaktionen über Javax
- Spring Boot Projekt, Verwendung von JPA und eines CRUDRepositories

Ziele:

- ⇨ Sie können die Funktion des **Persistenz-Frameworks Hibernate** erläutern
- ⇨ Sie können die Funktion des **Interfaces JPA** erläutern.
- ⇨ Sie können die Funktion von **Transaktionen** erläutern.
- ⇨ Sie können Entitäten aus einer Datenbank über das Persistenz-Framework und das Interface JPA mit Objekten der Applikation verwalten.
- ⇨ Sie setzen **CRUD** für die Verwaltung der Daten um.
- ⇨ Sie setzten verschiedene Beziehungen zwischen Entitäten in der Applikation um.

Inhaltsverzeichnis

Aufgabe: Informieren:

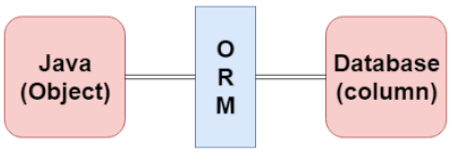
- Lesen Sie die folgenden Erläuterungen und lösen Sie die Aufgabe auf der letzten Seite.
- Erklären Sie zusammen mit Ihrem Lernpartner diese Begriffe:
  - Persistenz Framework
  - Hibernate
  - Session, Transaktion
  - Entity Manager
  - ORM
  - JPA
  - JPQL
  - Mapping (im Zusammenhang mit JPA)
- Erläutern Sie zusammen mit Ihrem Lernpartner
  - die Funktion des Persistenz-Frameworks Hibernate
  - die Funktion des Interfaces JPA
  - die Funktion von Transaktionen

Das Persistenz-Frameworks Hibernate:

(Quelle: https://de.wikipedia.org/wiki/Hibernate_(Framework))

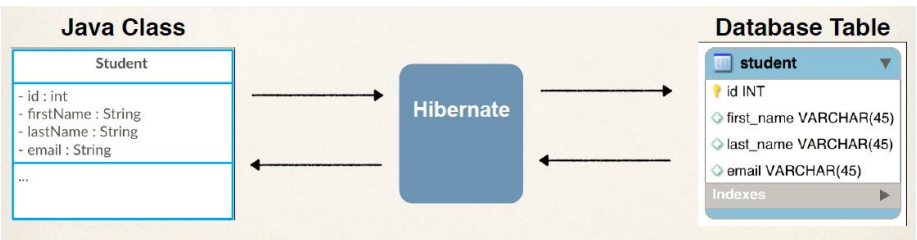
Hibernate (englisch für Winterschlaf halten) ist ein **Open-Source-Persistenz- und ORM-Framework für Java**.

Die Hauptaufgabe von Hibernate ist die objektrelationale Abbildung (englisch O-R-Mapping, kurz ORM).
Dabei werden **Daten-Objekte mit Attributen** und Methoden (Plain Old Java Objects oder POJOs genannt) in relationalen Datenbanken **gespeichert** und aus entsprechenden Datensätzen wiederum Objekte **erzeugt**.
Beziehungen zwischen Objekten werden auf entsprechende Datenbank-Relationen abgebildet.



In unseren bisherigen JDBC-Übungen haben Sie den ganzen Zugriff und das Abfüllen der Daten selbst implementieren müssen.

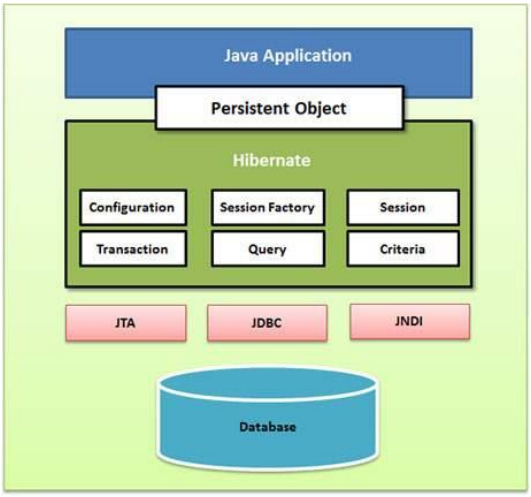
Hibernate übernimmt nun einige bis fast alle Schritte für Sie



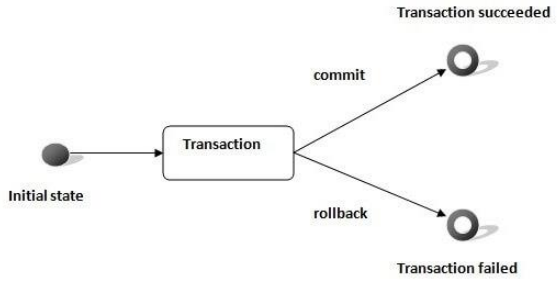
Darüber hinaus bietet Hibernate Mechanismen zur **Kompatibilität** mit verschiedenen Datenbanken. Die zum Datenbankzugriff erforderlichen SQL-Anweisungen werden nicht explizit in SQL programmiert, sondern von Hibernate in Abhängigkeit vom SQL-Dialekt der verwendeten Datenbank generiert.

Die zentralen Klassen von Hibernate sind:

- **SessionFactory**: Sie **lädt** die **Konfiguration** und die Abbildungen auf die Datenbank. Sie wird normalerweise nur einmal pro Anwendung erzeugt.
- **Session**: Sie ist das **Bindeglied** zwischen der **Java-Applikation** und den **Hibernate-Diensten** und bietet Methoden für Insert-, Update-, Delete- und Query-Operationen.
- **Transaction**: Sie bildet JDBC- und JTA-**Transaktionen** ab.
  Geschachtelte Transaktionen werden nicht unterstützt.



Die Transaktion



Datenübermittlung zwischen Applikation und Datenbank werden mittels Transaktionen ausgehandelt.

Ausgehend vom einem *Initial state* werden Daten mittels ***commit*** ausgetauscht.

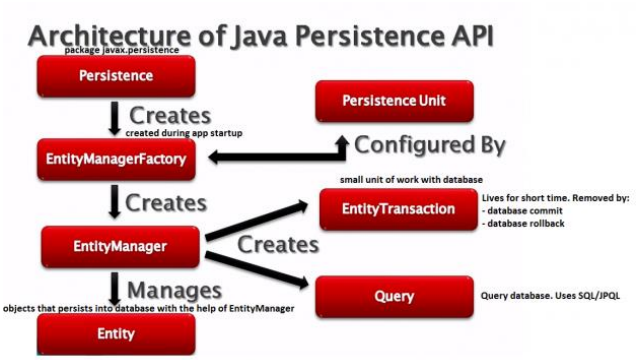
Falls die Transaktion **fehlschlägt**, so wird über ein ***rollback*** ein gültiger Zustand wiederhergestellt.

Auszug aus den Methoden der Session:

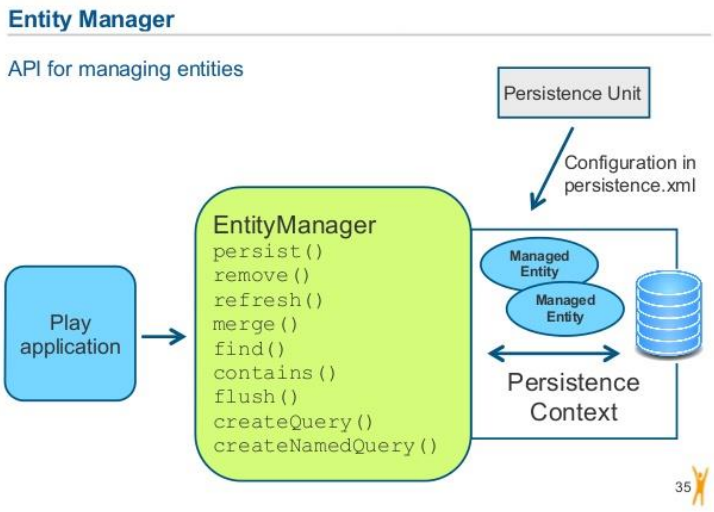
- **void begin()** starts a new transaction.
- **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
- **void rollback()** forces this transaction to rollback.
- **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
- **boolean isAlive()** checks if the transaction is still alive.
- **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
- **boolean wasCommited()** checks if the transaction is commited successfully.
- **boolean wasRolledBack()** checks if the transaction is rolledback successfully.

EntityManager:

Der Hibernate **EntityManager** bietet eine **Umsetzung** der Schnittstelle Java Persistence API (**JPA**) an.



Der EntityManager **kapselt** den Umgang mit der **Session und Transaction** in seinen Methoden wie *persists, remove, merge, find* usw.



Java Persistence API (JPA):

Die **Java Persistence API** (JPA) ist eine **Schnittstelle für Java-Anwendungen**, die die Zuordnung und die Übertragung von **Objekten zu Datenbankeinträgen vereinfacht**.

JPA erlaubt es eine **Java Klasse** mit den spezifischen Details einer **Datenbank, deren Entitäten** zu mappen (**verbinden**).

Zudem **definiert JPA eine Datenbanksprache die JPQL** um Queries auf einer Datenbank auszuführen. Der Vorteil von JPQL ist, dass diese **Sprache unabhängig von der Datenbank** selbst ist. Damit können Anwendung die JPA und JPQL verwenden leichter auf andere Datenbanken portiert werden.
JPQL lehnt sich stark an SQL an.

Das Mapping

Das **Mapping** zwischen der **Java Klasse und der Entität** in der Datenbank kann **entweder** über eine **Konfigurationsdatei oder** über **Annotierungen** in der Java Klasse geschehen.

Beispiel Mapping mit einer Konfigurationsdatei:

Employee.hbm.xml

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated Jun 18, 2016 1:29:07 PM by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="com.javacodegeeks.example.Employee" table="employee">
        <id name="id" type="int">
            <column name="ID" />
            <generator class="assigned" />
        </id>
        <property name="first_name" type="java.lang.String">
            <column name="FIRSTNAME" />
        </property>
        <property name="last_name" type="java.lang.String">
            <column name="LASTNAME" />
        </property>
        <property name="dept_id" type="int">
            <column name="DEPT_ID" />
        </property>
    </class>
</hibernate-mapping>
```

Beispiel Mapping mit annotieren in der Java Klasse:

```java
@Entity
@Table(name = "employee")
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id", unique = true)
    private int id;

    @Column(name = "firstname", unique = true)
    private String first_name;

    @Column(name = "lastname", unique = true)
    private String last_name;

    @Column(name = "dept_id", unique = true)
    private int dept_id;

    public Employee(){

    }
```

Quellen:

Informationen zu Hibernate:

http://hibernate.org/orm/

Transaktionen:

https://www.javatpoint.com/hibernate-transaction-management-example

Informationen zu JPA

https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

Informationen zu JPA Annotierungen

https://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html

https://dzone.com/articles/all-jpa-annotations-mapping-annotations

Weiteres Vorgehen in einer Übersicht

Sie haben ganz viele Informationen erhalten, nun ist es an der Zeit, das praktisch umzusetzen.

Sie werden:

1. Ein Beispiel Projekt "Employee" mit Hibernate, EntityManager und JPA in einer Maven basierten Java Applikation umsetzen
2. Das Beispiel um das vollständige CRUD erweitern

Zwei weitere Möglichkeiten anwenden, das Beispiel umzusetzen.

a) Variante mit EntityManager und Transaktion via Framework

b) Variante mit CRUDRepository

Danach realisieren Sie ein eigenes Projekt, diese Arbeit wird dann bewertet.

3. Ihre eigene, individuelle Datenbank entwerfen.
   Mehrere Entitäten mit unterschiedlichen Beziehungen
4. Java oder SpringBootMVC Applikation erstellen.
   CRUD umsetzen.