

# Objektorientierte Programmierung

Hochschule Bochum

WS 19/20

Dr.-Ing. Darius Malysiak

# V. OOP Software Engineering

## Hibernate

**Ziel:** Speichern von Objekten inklusive transitiver Beziehungen in einer korrespondierenden relationalen Datenbank.

Vorteile:

- Zugriff auf Modelabstraktion von Daten in Form von sogenannten Hibernate Entities, keine weitere Datenaufbereitung nötig.
- Abstraktion von Datenbankabfragen durch Hibernate Query Language (HQL), keine Abhängigkeit zur Datenbank.
- Caching-Mechanik für beschleunigte Abfragen.
- Intelligentes Laden von transitiven Objekten.

# V. OOP Software Engineering

## Hibernate Entities:

Eine POJO, welches mit ‚Entity‘ annotiert wurde wird als Hibernate Entity bezeichnet.  
Instanzen dieser Klasse sind instanzierte Entities.

```
@RequiredArgsConstructor  
@Getter  
@Setter  
@EqualsAndHashCode
```



```
@Entity
```

```
public class Person {
```

```
    @NotNull
```

```
    String name;
```

```
    @EqualsAndHashCode.Exclude Integer age = 10;
```


```
}
```

Hierdurch wird eine Hibernate  
Entity deklariert.

# V. OOP Software Engineering


## Hibernate Entities:

Definition der Tabelle.




```
@Entity
@Table(name = "thing")
@Getter
@Setter
@EqualsAndHashCode
```

Definition des Primärschlüssels  
sowie der Sequenz.



```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
protected long id;
```

Definition einer Spalte, per Default sind  
alle Attribute Spalten.



```
@Column
protected String name;

@Override
public String toString()
{
    return "Id:" + id + " name:" + name;
}
}
```

# V. OOP Software Engineering

## Hibernate Entities:

Annotation bieten viele Möglichkeiten zur Parametrisierung: z.B. für ‚@Column‘

```
@Column(  
    String name() default Fieldname;  
    boolean unique() default false;  
    boolean nullable() default true;  
    boolean insertable() default true;  
    boolean updatable() default true;  
    String columnDefinition() default "";  
    String table() default "";  
    int length() default 255;  
    int precision() default 0; // decimal precision  
    int scale()) default 0;
```

```
@Column(updatable = false, name = "flight_name", nullable = false, length=50)
```

# V. OOP Software Engineering

## Hibernate Datenbankbindung:

Für kleine Experimente oder Tests eignet sich die Java-basierte relationale Datenbank ,H2‘

```
<dependency>  
  <groupId>com.h2database</groupId>  
  <artifactId>h2</artifactId>  
  <version>1.4.197</version>  
</dependency>
```

# V. OOP Software Engineering

## Hibernate Datenbankbindung:

Die Datenbankbindung wird über eine Datei namens ‚hibernate.cfg.xml‘ im Ordner ‚resources‘ gesteuert.

```
<?xml version="1.0" ?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">org.h2.Driver</property>
        <property name="hibernate.connection.url">jdbc:h2:D:\\testDBH2</property>
        <property name="dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="show_sql">>true</property>
        <property name="hbm2ddl.auto">create</property>
        <mapping class="solutions.exercisel1.Thing"/>
    </session-factory>
    <!-- a static mapping if one wants to avoid annotations -->
    <!-- <mapping resource="Things.hbm.xml"/> -->
</hibernate-configuration>
```

# V. OOP Software Engineering

## Hibernate Datenbankbindung:

Früher musste das Mapping von Feld zu Tabellenspalte manuell konfiguriert werden! Hier ein Beispiel für das Mapping von Thing, die Datei ‚Things.hbm.xml‘ sollte ebenfalls im Ordner ‚resources‘ liegen.

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="solutions.exercisel1.Thing" table="thing">
        <id name="id" type="int">
            <column name="id" />
            <generator class="increment" />
        </id>
        <property name="name" type="java.lang.String">
            <column name="name" />
        </property>
    </class>
</hibernate-mapping>
```



# V. OOP Software Engineering

## Hibernate Datenbankbindung:

Eine Hilfsklasse ‚Sessionhelper‘ zur Herstellung der Datenbankverbindung.

```
public class SessionHelper {  
    private static final SessionFactory sessionFactory;  
  
    static {  
        try {  
            Configuration config = new Configuration();  
            sessionFactory = config.configure().buildSessionFactory();  
        } catch (Throwable e) {  
            System.err.println("Error in creating SessionFactory object."  
                + e.getMessage());  
            throw new ExceptionInInitializerError(e);  
        }  
    }  
  
    public static SessionFactory getSessionFactory() {  
        return sessionFactory;  
    }  
}
```

# V. OOP Software Engineering


## Hibernate Datenbankankbindung:

Eine Hilfemethode ‚saveThing‘  
zum Persistieren einer Entity.

```
public static void saveThing(Session session, Thing t)
{
    Transaction tx = null;
    Long id = null;
    try {
        tx = session.beginTransaction();

        id = (Long) session.save(t);
        System.out.println("Thing saved with id:"+id);
        session.save(t);

        tx.commit();
    } catch (HibernateException ex) {
        if (tx != null) {
            tx.rollback();
        }
        ex.printStackTrace();
    }
}
```



Hibernate unterstützt Transaktionen! ☺


# V. OOP Software Engineering

## Hibernate Datenbankbindung:

Eine Hilfemethode ‚getThing‘  
zum Abfragen einer Entity.

```
public static Thing getThing(Long id, Session session)
{
    Transaction tx = null;
    Thing thing = null;
    try{
        tx = session.beginTransaction();
        //HQL query, note: HQL uses class names instead
        //of table names, and property names instead of column names.
        Query q = session.createQuery("from Thing where id = :id");
        q.setParameter("id", id);
        List<Thing> list = q.list();
        if(list.size() == 1)
        {
            thing = list.get(0);
        }
        tx.commit();
    } catch (HibernateException e) {
        e.printStackTrace();
    }

    return thing;
}
```



HQL benutzt Klassennamen anstatt  
Tabellennamen und Feldnamen anstatt  
Spaltennamen.

# V. OOP Software Engineering

## Anti-Patterns:

**Def.:** Ein Design-Pattern / Codestyle wird als Anti-Pattern bezeichnet falls durch seine Anwendung eine Implementierung instabil, fehleranfällig oder schwer wartbar wird.

```
HashMap<String, Object> map = new HashMap<String, Object>() {{  
    put("test", new Object());  
    put("test2", new Object());  
}};
```

# V. OOP Software Engineering

## Anti-Patterns:

```
HashMap<String, Object> map = new HashMap<String, Object>() {  
    put("test", new Object());  
    put("test2", new Object());  
};
```

Anonyme Unterklasse

Initializer der Unterklasse



Verbesserung ? ⇔ Was ist hier eigentlich nicht OK??

- Variablenname prägnant wählen.
- Diamond-Operator nutzen.
- Initialisierungsmethode.
- Falls es ein Feld ist: Soll der Modifier wirklich Package Protected sein?
- Map-Keys als Konstanten deklarieren.
- Dokumentation?

# V. OOP Software Engineering

## Anti-Patterns:

Was fällt Ihnen auf?

- Reihenfolge hängt mit Modifier zusammen.
- AllCaps bei Konstanten.
- Camel-Case bei Methoden und nicht-konstanten Feldern.
- Einrückung.
- Sprechende Namen.
- JavaDoc der Klasse.
- Keine JavaDoc innerhalb der Klasse... .. ?

```
/**
 * A class which represents an example for good
 * coding. One should study its structure.
 * @since 1.0
 */
public class GoodPattern {
    public final static String KEY1="Object1";
    public final static String KEY2="Object2";

    private Map<String, Object> map = new HashMap<>();

    public GoodPattern() {
        initMembers();
    }

    private void initMembers() {
        map.put(KEY1, new Object());
        map.put(KEY2, new Object());
    }
}
```

# V. OOP Software Engineering

## Anti-Patterns:

### Principle of Least Surprise

*People are part of the system. The design should match the user's experience, expectations, and mental models.*

```
/**
 * A class which represents an example for good
 * coding. One should study its structure.
 * @since 1.0
 */
public class GoodPattern {
    public final static String KEY1="Object1";
    public final static String KEY2="Object2";

    private Map<String, Object> map = new HashMap<>();

    public GoodPattern() {
        initMembers();
    }

    private void initMembers() {
        map.put(KEY1, new Object());
        map.put(KEY2, new Object());
    }
}
```

# V. OOP Software Engineering

## Anti-Patterns:

### Principle of Least Surprise

*People are part of the system. The design should match the user's experience, expectations, and mental models.*

- Hilfsmethoden sind private und stehen unten!  
-> Details sollen unwichtig für Verständnis der Programmlogik sein!
- Methodennamen erklären die Businesslogic.
- Formatierung identifiziert Typen.
- Methoden sind kurz.
- Kommentare sind prägnant!

```
/**
 * A class which represents an example for good
 * coding. One should study its structure.
 * @since      1.0
 * */
public class GoodPattern {
    public final static String KEY1="Object1";
    public final static String KEY2="Object2";

    private Map<String, Object> map = new HashMap<>();

    public GoodPattern() {
        initMembers();
    }

    private void initMembers() {
        map.put(KEY1, new Object());
        map.put(KEY2, new Object());
    }
}
```



# V. OOP Software Engineering

Nächste Vorlesung:

***More Anti-Patterns***

***+***

***API Design***

***+***

***Tao of Programming***