Agenda

Configuration

EntityManagerFactory, EntityManager

Konfiguration

Mapping

Typen

Hibernate

Java Persistence API in einem Schichtenmodell

Configuration

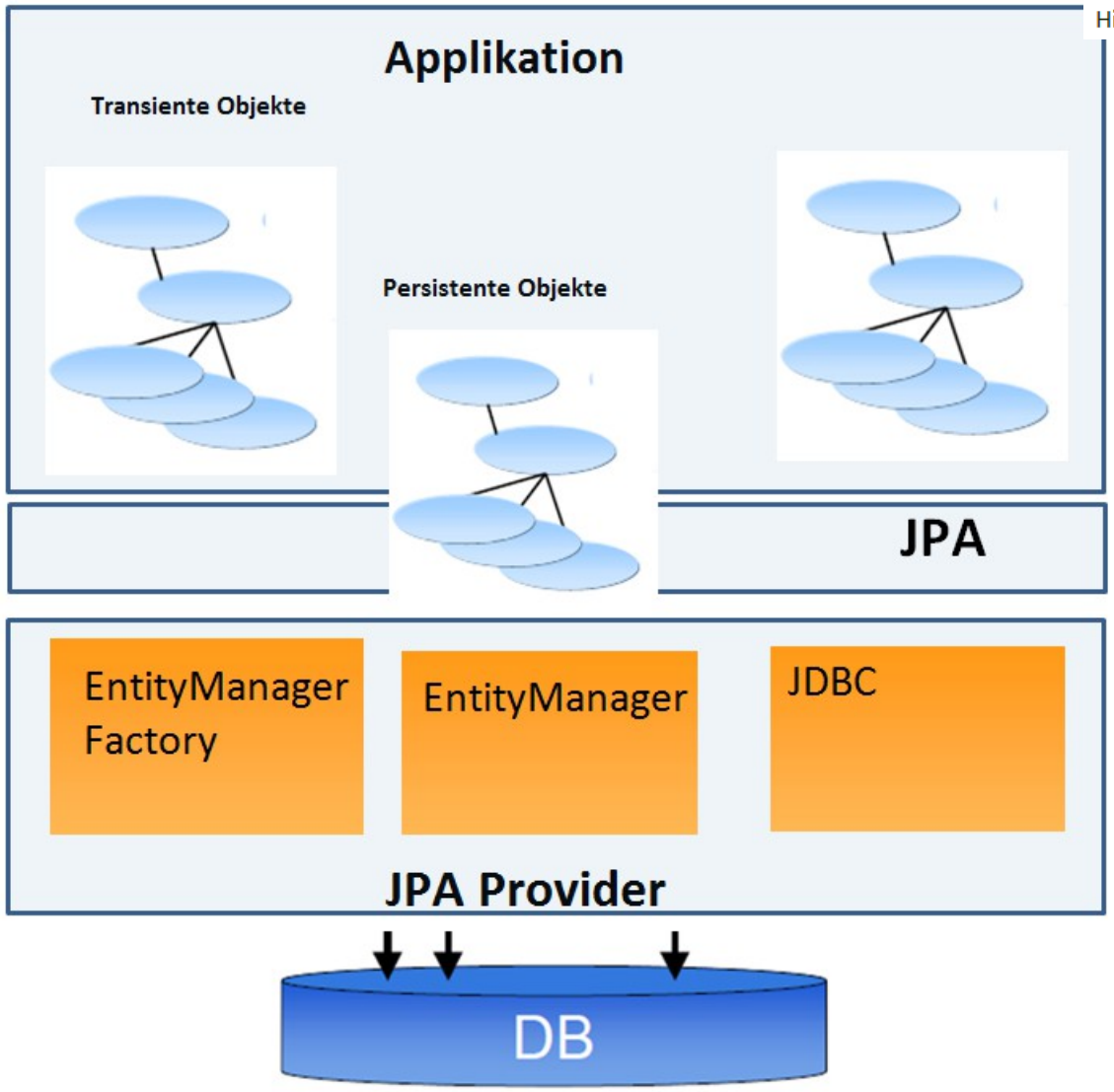
EntityManagerFactory, EntityManager

Konfiguration

Mapping

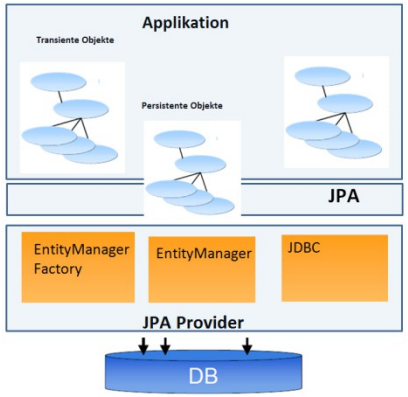
Typen

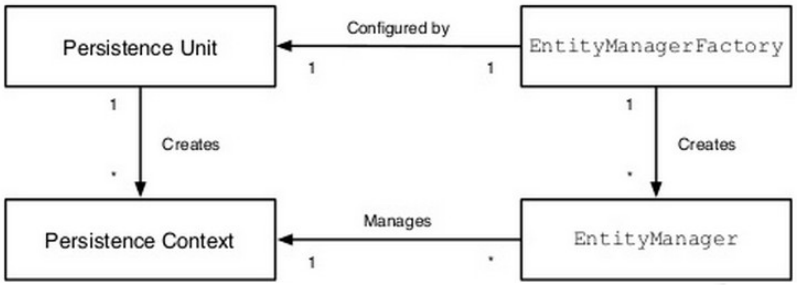
Hibernate



EntityManagerFactory

- createEntityManager() erzeugt EntityManager
- EntityManagerFactory wird threadübergreifend genutzt und ist threadsicher
- Default:
  - EntityManagerFactory pro Anwendung
  - Pro DB eine EntityManagerFactory (bei mehreren Datenbanken)
- Stellt Cache (providerabhängig) zur Verfügung
  - generierte SQL-Statements
  - 2nd Level Cache
- in Hibernate: **SessionFactory**





EntityManager

Configuration

EntityManagerFactory, EntityManager

Konfiguration

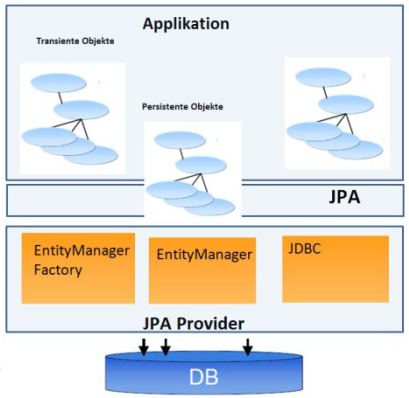
Mapping

Typen

Hibernate

- Leichtgewichtige Session zur Datenbank
- Definiert Methoden zum Datenzugriff
  - persist, merge, remove, createCriteria, createQuery, createSQLQuery
- First-Level Cache
- Nicht Threadsafe
- Fehler via Runtime Exceptions
  - Im Exception-Fall ist die session „dirty“! Dann ein Rollback.
- Hibernate: Session

```
Session session =
(Session)  em.getDelegate()
```



Create EntityManagerFactory EntityM

- **Selbst in der Anwendung erzeugen**

```
EntityManagerFactory emf = Persistence
.createEntityManagerFactory("jpaDatabase") ;
EntityManager em = entityManagerFactory
   .createEntityManager() ;
```

- **Erzeugt durch Container (Dependency Injection):**

```
@PersistenceUnit(unitName = "jpaDatabase")
private EntityManagerFactory emf;
@PersistenceContext
private EntityManager em;
```

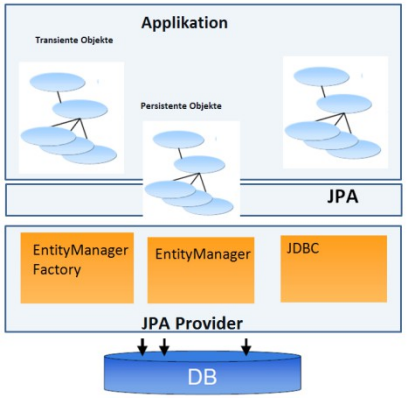
EntityManager API

- **Basisoperationen**
  - em.persist(object)
  - em.find(class, id) / em.getReference(class, id)
  - em.merge(object)
  - em.remove(object)
- **Zusatzoperationen**
  - em.contains(object)
  - em.refresh(object)
  - em.flush()
  - em.clear()
  - em.detach(object)
  - em.getDelegate()

EntityTransaction

- Einheitliche API oberhalb diverser Transaction-APIs
  - JDBC-Transaktionen
  - Java Transaction API

- Hibernate: Transaction



Agenda

Configuration

EntityManagerFactory, EntityManager

Konfiguration

Mapping

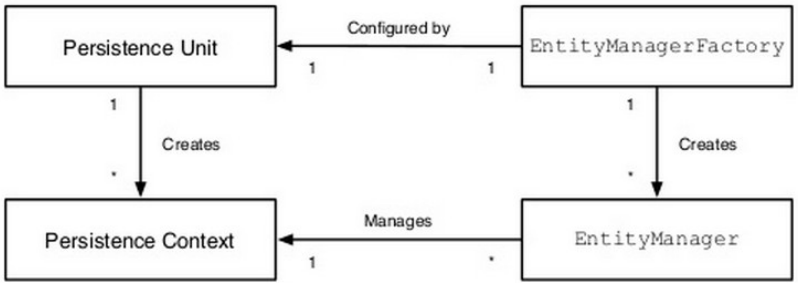
Typen

Hibernate

Konfiguration EntityManagerFactory

- **Factory erzeugen**

```
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myPU" ) ;
```



- **Überschreiben von Parametern**

```
Map myConf = new HashMap();
conf.put("hibernate.hbm2ddl.auto",  "create-drop");
EntityManagerFactory emf =
Persistence.createEntityManagerFactory("myPU",  myConf);
```

Konfiguration

- Pflicht: META-INF\persistence.xml
- proprietäre Hints an Provider erlaubt
- kann mehrere Persistenz-Units enthalten

```
<persistence>
        <persistence-unit name="myPU">
      …
        </persistence-unit>
</persistence>
```

persistence.xml mit Hibernate

- persistence.xml mit Hibernate als Provider

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">
   <persistence-unit name="manager1" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>java:/DefaultDS</jta-data-source>
      <mapping-file>ormap.xml</mapping-file>
      <jar-file>MyApp.jar</jar-file>
      <class>org.acme.Employee</class>
      <class>org.acme.Person</class>
      <class>org.acme.Address</class>
       <properties>
           <property name="hibernate.dialect"
   value="org.hibernate.dialect.HSQLDialect"/>
           <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      </properties>
   </persistence-unit>
</persistence>
```

persistence.xml: Weitere Einstellungen

- **Provider**: Spezifikation des Persistence Providers
- **JTA-DataSource**: Datasource aus JNDI
- **Mapping File**: externe Mappings-Datei

```
<persistence-unit name="myPU">
   <provider>org.xxx.HibernatePersistence</provider>
   <jta-data-source>java:/DefaultDS</jta-data-source>
   <mapping-file>orm.xml</mapping-file>
</persistence-unit>
```

orm.xml

- orm.xml kann Mappings aus Annotationen überschreiben

```
<?xml versions"1.0" encoding="UTF-8"?>
<entity-mappings version="2.0" ...>
  <persistence-unit-metadata>
  <xml-mapping-metadata-complete/>
  </persistence-unit-metadata>
  <entity class="de.example.domain.User" metadata-
   complete="true">
   <table name="BENUTZER"/>
   <attributes >
          <id name="id"/>
          <basic name="username"/>
   </attributes>
  </entity>
</entity-mappings >
```

Agenda

Configuration

EntityManagerFactory, EntityManager

Konfiguration

Mapping

Typen

Hibernate

Grundlagen Mapping - Felder und Pro...

- **@Id** ist Pflicht, restliche Felder per Konvention gemappt
- Ort der **@Id**-Annotation spezifiziert Property/Field Access
  - alternativ: **@Access**

```
@Entity
public class Spieler implements Serializable {
   @Id
   private Long spielerID;

   // Persistentes Feld
   private String position;
}
```

@Basic

- **@Basic** (optional) mappt:
  - primitiven Datentypen
  - Serialisierbaren Objekten

```
@Basic
int getLength()   {   ... }
@Basic
private String position;
@Basic
private String comment
```

Lazy/Eager Fetching

- Default-Werte
  - @Basic, X-1-Beziehungen: FetchType.EAGER
  - X-n-Beziehungen: FetchType.LAZY

```
@Basic(fetch = FetchType.EAGER)
private String   position;
@Basic (fetch = FetchType.LAZY)
private String  comment;
```

Transiente Felder

- Transiente Felder werden nicht persistiert

```
@Entity
public class Spieler implements Serializable {
  private transient BigDecimal gehalt;
  @Transient
  public int getAlter()   {   ... }
}
```

Mapping von Tabellen

- Expliziter Tabellenname erlaubt
- Default: Klassenname

```
@Entity
@Table(name="EXAMPLE_USER")
public class User implements Serializable{
…
Mehr Optionen:
@Table(catalog = "catalogName",
  schema = "schemaName",
  uniqueConstraints =
  @UniqueConstraint(columnNames={"month",
  "day"}))
```

Mapping von Spalten


```
@Entity
public class Flight implements Serializable {
  @Column(name = "flight_name",
  updatable = false,
  insertable = true,
  nullable = false,
  unique = true,
  length=50)
  public String name;
}
```

Constraints

- Ist Spalte Nullable?
- Unique-Index?

```
@Entity
public class Person implements Serializable {
    @Column(nullable=false, unique=true)
    private String login;
    …
}
```

Agenda

Configuration

EntityManagerFactory, EntityManager

Konfiguration

Mapping

Typen

Hibernate

Weitere Mappings

- Konfigurationspflicht für Datumswerte

```
@Temporal(TemporalType.TIME)
private java.util.Date
  departureTime;
```

Enum Constant Summary

DATE	Map as java.sql.Date
TIME	Map as java.sql.Time
TIMESTAMP	Map as java.sql.Timestamp

- Enumerationen als String oder Ordinalszahl in DB

```
@Enumerated(EnumType.STRING)
private Grade spielerRating;
```

- java.sql.Clob und java.sql.Blob mit@Lob versehen

```
@Lob
private String longDescription;
```

Aufgabe

- Aufgabe 3:
  - persistence.xml
  - Enumerationen
  - Datumswerte
  - Large Objects

Agenda

Configuration

EntityManagerFactory, EntityManager

Konfiguration

Mapping

Typen

Hibernate

org.hibernate.cfg.Configuration

- Schnittstelle zur Konfiguration
- Spezifiziert Ort der Mapping-Dokumente
- Enthält Konfigurationshinweise für Hibernate

Programmatische Konfiguration


```
Configuration cfg = new Configuration().configure()
```

- new Configuration() lädt hibernate.properties Datei, falls sie existiert
- Andernfalls lädt configure() hibernate.cfg.xml

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <property name="connection.url">
          jdbc:mysql://localhost:3307/JH
        </property>
    <property name="connection.driver_class">
          com.mysql.jdbc.Driver
        </property>
    <property name="connection.username">
          mkonda
        </property>
    <property name="connection.password">
          password
        </property>
    <property name="dialect">
          org.hibernate.dialect.MySQL5Dialect
        </property>
    <mapping resource="Movie.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

hibernate.properties

```
#Derby Properties
hibernate.connection.driver_class = org.apache.derby.jdbc.EmbeddedDriver
hibernate.connection.url = jdbc:derby:memory:JH;create=true
hibernate.connection.username = myuser
hibernate.connection.password = mypassword
hibernate.dialect = org.hibernate.dialect.DerbyDialect
```

- Alternativ:
- new Configuration().configure("config/myconfig.cfg.xml")

Programmatische Konfiguration

- **Hinzufügen von Mapping-Dateien**

```
Configuration cfg = new Configuration()
.addResource("User.hbm.xml")
.addResource("Customer.hbm.xml");
```

- **Hinzufügen von persistenten Klassen & Properties**

```
Configuration cfg = new Configuration()
.addClass(de.mycomp.User.class)
.addClass(de.mycomp.Customer.class)
.setProperty("hibernate.dialect", "org.[..].MySQLInnoDBDialect")
.setProperty("hibernate.connection.datasource","java:comp/env/jdbc/db")
.setProperty("hibernate.order_updates", "true");
```

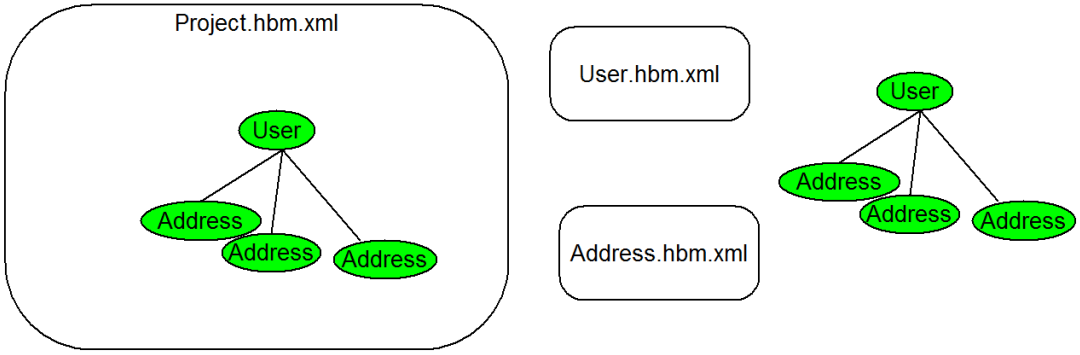
Hinzufügen von Mapping-Informationen

- **Programmatisch hinzufügen**

```
Configuration cfg = new Configuration()
cfg.addClass(de.example.User.class) ;
// oder
cfg.addRessource("de/example/User.hbm.xml");
```

- **In Konfiguration(hibernate.cfg.xml) hinzufügen**

```
<mapping resource="de/example/User.hbm.xml"/>
```



hibernate.cfg.xml


```
<?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>
    <property name="hibernate.connection.driver_class">
      org.hsqldb.jdbcDriver
    </property>
    <property name="hibernate.connection.url">
      jdbc:hsqldb:file:./hSqlDbData/myDB;shutdown=true
    </property>
    <property name="hibernate.connection.username">sa</property>
    <property name="hibernate.connection.password"></property>
    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>
    <property name="show_sql">false</property>
    <property name="transaction.factory_class">
      org.hibernate.transaction.JDBCTransactionFactory
    </property>
    <property name="hibernate.cache.provider_class">
      org.hibernate.cache.HashtableCacheProvider
    </property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <mapping resource="Kunde.hbm.xml"/>
    <mapping resource="Speise.hbm.xml"/>
    <mapping resource="Kommentar.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Weitere Properties

- **hibernate.dialect**
  - Angabe des Datenbankdialektes über Klassenname
  - z. B.: org. hibernate.dialect.HSQLDialect
- **hibernate. show_sql**
  - Sollen SQL-Statements auf der Konsole ausgegeben werden?
- **hibernate.default_schema**
  - Datenbankschema / Tablespace
- **hibernate.default_catalog**
  - Datenbankkatalog

Weitere Properties

- **hibernate.session_factory_name**
  - An welche JNDI-Adresse soll SessionFactory automatisch gebunden werden?
  - z. B.: jndi/exampledemo
- **hibernate.hbm2ddl.auto**
  - soll DB Schema automatisch erzeugt werden?
  - Werte: **validate, update, create, create-drop**
  - Kommandozeilen-Tool: **org.hibernate.tool.hbm2ddl.SchemaExport**

*.hbm.xml: hibernate-mapping


```
<hibernate-mapping
        schema="schemaName"
        catalog="catalogName"
        default-cascade="cascade_style"
        default-access="field|property|ClassName"
        default-lazy="true|false"
        auto-import="true|false"
        package="package.name"
/>
```

- **schema** (optional): The name of a database schema.
- **catalog** (optional): The name of a database catalog.
- **default-cascade** (optional - defaults to none): A default cascade style.
- **default-access** (optional - defaults to property): The strategy Hibernate should use for accessing all properties. Can be a custom implementation of PropertyAccessor.

*.hbm.xml: hibernate-mapping


```
<hibernate-mapping
        schema="schemaName"
        catalog="catalogName"
        default-cascade="cascade_style"
        default-access="field|property|ClassName"
        default-lazy="true|false"
        auto-import="true|false"
        package="package.name"
/>
```

- **default-lazy** (optional - defaults to true): The default value for unspecifed lazy attributes of class and collection mappings.
- **auto-import** (optional - defaults to true): Specifies whether we can use unqualified class names (of classes in this mapping) in the query language.

*.hbm.xml: hibernate-mapping


```
<hibernate-mapping
        schema="schemaName"
        catalog="catalogName"
        default-cascade="cascade_style"
        default-access="field|property|ClassName"
        default-lazy="true|false"
        auto-import="true|false"
        package="package.name"
/>
```

- **package** (optional): Specifies a package prefix to assume for unqualified class names in the mapping document.
- If you have two persistent classes with the same (unqualified) name, you should set auto-import="false". Hibernate will throw an exception if you attempt to assign two classes to the same "imported" name.

"hbm.xml: class Mapping (Auszug)


```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
        select-before-update="true|false"
        polymorphism="implicit|explicit"
        where="arbitrary sql where condition"
        persister="PersisterClass"
        batch-size="N"
        optimistic-lock="none|version|dirty|all"
        lazy="true|false"
        entity-name="EntityName"
        check="arbitrary sql check condition"
        rowid="rowid"
        subselect="SQL expression"
        abstract="true|false"
        node="element-name"
/>
```

- **name** (optional): The fully qualified Java class name of the persistent class (or interface). If this attribute is missing, it is assumed that the mapping is for a non-POJO entity.
- **table** (optional - defaults to the unqualified class name): The name of its database table.
- **discriminator-value** (optional - defaults to the class name): A value that distiguishes individual subclasses, used for polymorphic behaviour. Acceptable values include null and not null

"hbm.xml: class Mapping (Auszug)


```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
        select-before-update="true|false"
        polymorphism="implicit|explicit"
        where="arbitrary sql where condition"
        persister="PersisterClass"
        batch-size="N"
        optimistic-lock="none|version|dirty|all"
        lazy="true|false"
        entity-name="EntityName"
        check="arbitrary sql check condition"
        rowid="rowid"
        subselect="SQL expression"
        abstract="true|false"
        node="element-name"
/>
```

- **mutable** (optional, defaults to true): Specifies that instances of the class are (not) mutable.
- **schema** (optional): Override the schema name specified by the root <hibernate-mapping>element.
- **catalog** (optional): Override the catalog name specified by the root <hibernate-mapping>element.

"hbm.xml: class Mapping (Auszug)


```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
        select-before-update="true|false"
        polymorphism="implicit|explicit"
        where="arbitrary sql where condition"
        persister="PersisterClass"
        batch-size="N"
        optimistic-lock="none|version|dirty|all"
        lazy="true|false"
        entity-name="EntityName"
        check="arbitrary sql check condition"
        rowid="rowid"
        subselect="SQL expression"
        abstract="true|false"
        node="element-name"
/>
```

- **proxy** (optional): Specifies an interface to use for lazy initializing proxies. You may specify the name of the class itself.
- **dynamic-update** (optional, defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed.
- **dynamic-insert** (optional, defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

"hbm.xml: class Mapping (Komplet


```
<class
        name="ClassName"
        table="tableName"
        discriminator-value="discriminator_value"
        mutable="true|false"
        schema="owner"
        catalog="catalog"
        proxy="ProxyInterface"
        dynamic-update="true|false"
        dynamic-insert="true|false"
        select-before-update="true|false"
        polymorphism="implicit|explicit"
        where="arbitrary sql where condition"
        persister="PersisterClass"
        batch-size="N"
        optimistic-lock="none|version|dirty|all"
        lazy="true|false"
        entity-name="EntityName"
        check="arbitrary sql check condition"
        rowid="rowid"
        subselect="SQL expression"
        abstract="true|false"
        node="element-name"
/>
```

- https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/4.2/html/Hibernate_Reference_Guide/Basic_OR_Mapping.html

*hbm.xml: property Mapping (Auszug)


```
<property
        name="propertyName"
        column="column_name"
        type="typename"
        update="true|false"
        insert="true|false"
        formula="arbitrary SQL expression"
        access="field|property|ClassName"
        lazy="true|false"
        unique="true|false"
        not-null="true|false"
        optimistic-lock="true|false"
        generated="never|insert|always"
        node="element-name|@attribute-name|element/@attribute|."
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

- **name**: the name of the property, with an initial lowercase letter.
- **column** (optional - defaults to the property name): the name of the mapped database table column. This may also be specified by nested <column> element(s).
- **type** (optional): a name that indicates the Hibernate type.

*hbm.xml: property Mapping (Auszug)


```
<property
        name="propertyName"
        column="column_name"
        type="typename"
        update="true|false"
        insert="true|false"
        formula="arbitrary SQL expression"
        access="field|property|ClassName"
        lazy="true|false"
        unique="true|false"
        not-null="true|false"
        optimistic-lock="true|false"
        generated="never|insert|always"
        node="element-name|@attribute-name|element/@attribute|."
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

- **update, insert** (optional - defaults to true) : specifies that the mapped columns should be included in SQL UPDATE and/or INSERT statements. Setting both to false allows a pure "derived" property whose value is initialized from some other property that maps to the same colum(s) or by a trigger or other application.
- **formula** (optional): an SQL expression that defines the value for a *computed* property. Computed properties do not have a column mapping of their own.
- **access** (optional - defaults to property): The strategy Hibernate should use for accessing the property value.

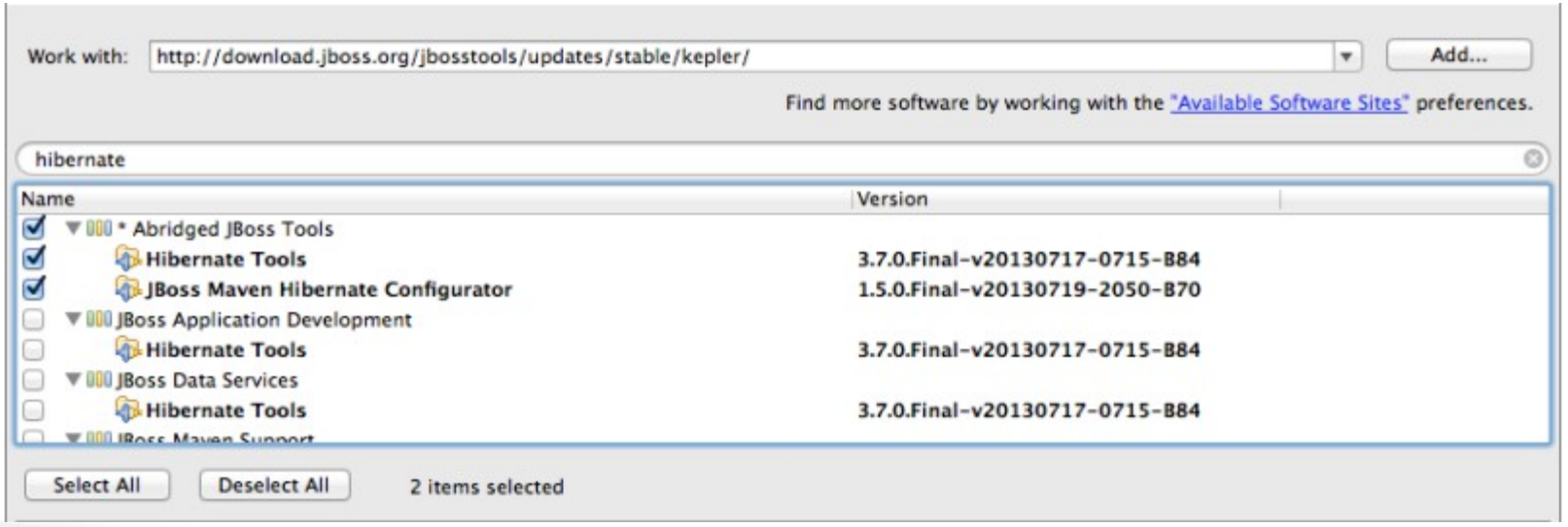
*hbm.xml: property Mapping (Komplett)


```
<property
        name="propertyName"
        column="column_name"
        type="typename"
        update="true|false"
        insert="true|false"
        formula="arbitrary SQL expression"
        access="field|property|ClassName"
        lazy="true|false"
        unique="true|false"
        not-null="true|false"
        optimistic-lock="true|false"
        generated="never|insert|always"
        node="element-name|@attribute-name|element/@attribute|."
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

- https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/4.2/html/Hibernate_Reference_Guide/Basic_OR_Mapping.html

Installation Hibernate Tools

- In Eclipse Help->Install new Software…
- Eingabe von http://download.jboss.org/jbosstools/updates/stable/kepler
- Filter auf Hibernate
- Auswahl Hibernate Tools und JBoss Maven Hibernate Configurator



Hibernate Tools

Configuration

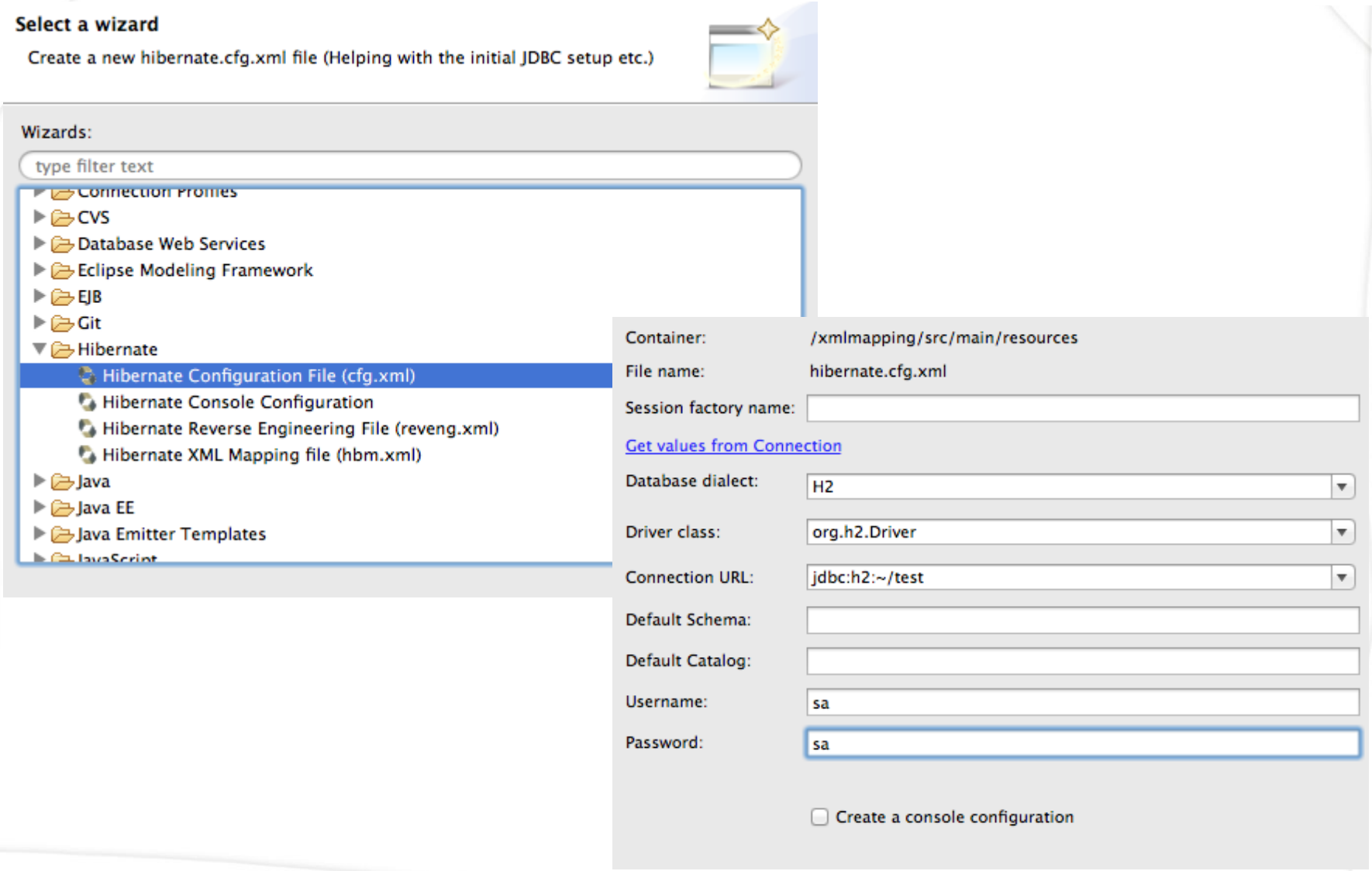
EntityManagerFactory, EntityManager

Konfiguration

Mapping

Typen

Hibernate



Hibernate Typen

- Hibernate Typen abstrahieren von den DB SQL Typen
- Entwickler mappt Java Typen auf Hibernate Typen.
- Hibernate mappt dann auf den DB SQL Typ

Hibernate Standard Typen

Java Type	Hibernate Type Name	SQL Type
Primitives		
Boolean or boolean	boolean	BIT
	true_false	CHAR(1)('T'or'F')
	yes_no	CHAR(1)('Y'or'N')
Byte or byte	byte	TINYINT
char or Character	character	CHAR
double or Double	double	DOUBLE
float or float	float	FLOAT
int or Integer	integer	INTEGER
long or Long	long	BIGINT
short or Short	short	SMALLINT
String		
java.lang.String	string	VARCHAR
	character	CHAR(1)
	text	CLOB
Arbitrary Precision `Numeric`		
java.math.BigDecimal	big_decimal	NUMERIC
`Byte` Array		
byte[] or Byte[]	binary	VARBINARY

Hibernate Standard Typen

Time and Date		
java.util.Date	date	DATE
	time	TIME
	timestamp	TIMESTAMP
java.util.Calendar	calendar	TIMESTAMP
	calendar_date	DATE
java.sql.Date	date	DATE
java.sql.Time	time	TIME
java.sql.Timestamp	timestamp	TIMESTAMP

Hibernate Standard Typen

Localization		
java.util.Locale	locale	VARCHAR
java.util.TimeZone	timezone	
java.util.Currency	currency	
Class Names		
java.lang.Class	class	VARCHAR
Any Serializable Object		
java.io.Serializable	Serializable	VARBINARY
JDBC Large Objects		
java.sql.Blob	blob	BLOB
java.sql.Clob	clob	CLOB

Beispiel Hibernate Standard Typen


```
<hibernate-mapping>
  <class name="Person" table="PERSON" discriminator-
  value="PE">
     <id name="id" column="ID" type="long">
          <generator class="native"/>
     </id>
     <discriminator column="PERSON_TYPE" type="string"/>
     <property name="birthdate" column="BIRTHDATE"
    type="date"/>
     <list name="papers" table="STUDENT_PAPER">
        <key column="STUDENT_ID"/>
        <list-index column="POSITION"/>
          <element column="PAPER_PATH" type="string"/>
     </list> <!-- mapping of other fields -->
  </class>
</hibernate-mapping>
```

Benutzerdefinierte Typen

- **Wann im Einsatz?**
  - spezielle Mappings bei Legacy Datenbanken
  - Konstanten / Enumerationen auf Zahlen mappen
    - **Anrede, Familienstand, True/False, ...**
  - Property auf mehrere verschiedene Spalten mappen
    - **Kleine oft verwendete Objekttypen**
    - **Geldbetrag mit Währung / Betrag**

- **Wie realisiert?**
  - Implementierung von UserType oder CompositeUserType

```
<property name="salary" type="...MonetaryAmountCompositeUserType">
   <column name="gehalt"/>
   <column name="waehrung"/>
</property>
```