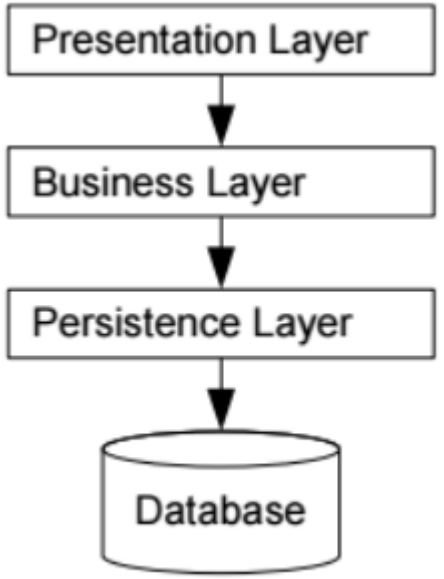
Hibernate

5.*

OAK 3
ACADEMY

Layerd architecture



Presentation layer: Visual presentation

Business layer: logic

Persistence layer: connection with database

ORM

= **O**bject **R**elational **M**apping

→ Hibernate framework provides an implementation of a persistence layer with ORM.

→ Hibernate uses POJOs.

→ JPA: Java Persisence Api = standardized ORM solution.

→ Hibernate = concrete implementation of JPA.

→ Version currently available: Hibernate 5.2 - JPA 2.1

OAK 3
ACADEMY

Dependencies

```
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>5.2.12.Final</version>
</dependency>
```

```
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>6.0.6</version>
</dependency>
```

OAK 3
ACADEMY

Configuration of JPA

- Database
- Username
- Password
- …

→ **persistence.xml** in META-INF directory.

OAK 3
ACADEMY

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

    <persistence-unit name="course" transaction-type="RESOURCE_LOCAL">

        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/spring"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="javax.persistence.schema-generation.database.action" value="create"/>

            <property name="hibernate.show_sql" value="true"/> <!--Show SQL in logging -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>

        </properties>
    </persistence-unit>

</persistence>
```

OAK 3
ACADEMY

Exercise 1, p15

Entity classes

- Default constructor
- Getters and setters
- Field with primary key (annotation **@Id**)
- Class and methods: not final!

→ place annontation **@Entity** above class declaration

OAK 3
ACADEMY

Primary Key

Natural primary key: is part of the object. Eg: National register number person.

Surrogate primary key: extra field with unique value. (Usually int or long)

Single primary key: property with the annotation @Id

Composite primary key: composed of multiple properties (separate primary-key class)

OAK 3
ACADEMY

Composite primary key class

- Serializable
- Public constructor without arguments
- Getters and setters
- Implement Equals and Hashcode.

@Id: place to every property

@IdClass: place above class declaration

→ **Example: p18 - 19**

OAK 3
ACADEMY

Autogenerated primary keys

@GenerateValue

Bv:

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
public long getId() {
    return id;
}
```

Strategy	*Description*
IDENTITY	An identity column is used in the database. This is a column of which the content is automatically incremented whenever a new record is created (auto-increment).
SEQUENCE	A sequence is used in the database.
TABLE	A separate table is used, in which the highest value of the primary key is stored.
AUTO	The JPA provider chooses the best strategy itself and takes the possibilities of the underlying database into account. This is the default value.

OAK 3 ACADEMY

FieldAccess vs PropertyAccess

JPA can handle both

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
public long getId() {
    return id;
}

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

OAK 3 ACADEMY

Exercise 1, p24

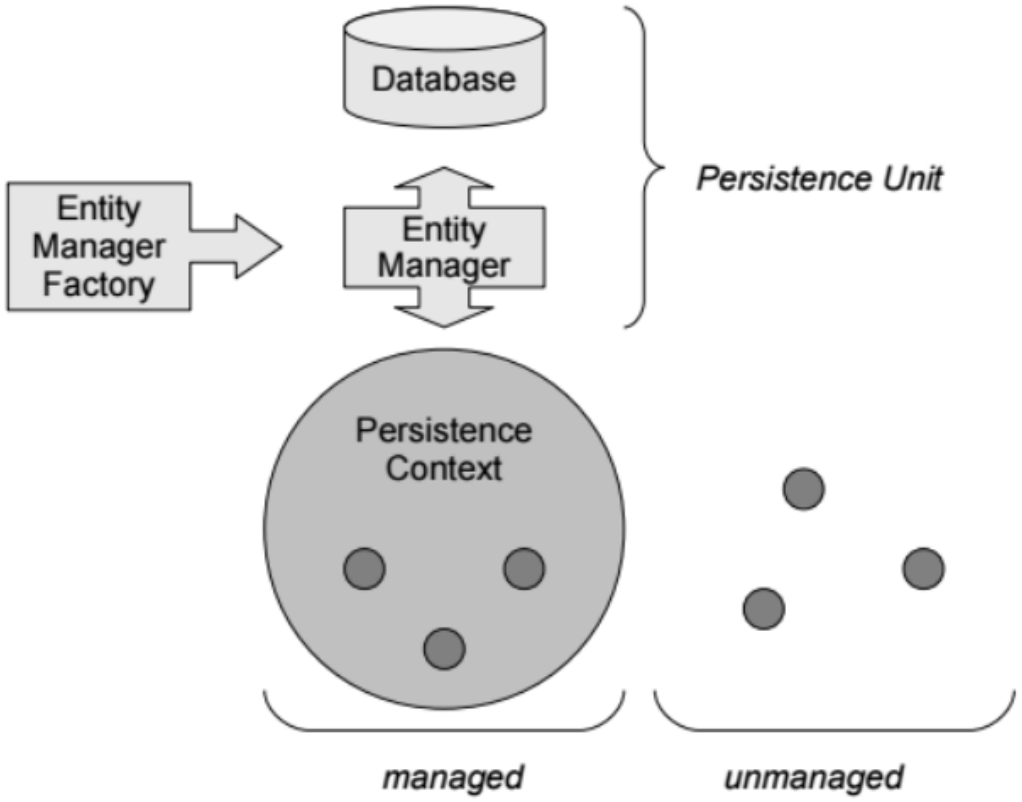
Entity objects

Entity object	Omschrijving
EntityManager	Communication with database
EntityManagerFactory	Object to create **EntityManager.**
Persistence context	Collection of objects, controlled by **EntityManager.**

→ Persistence unit: configured in **persistence.xml**

OAK 3
ACADEMY

Graphical design



Exercise 1, p28

Transactions

= ACID principle

Atomair: must either be fully executed or not

Consistent : data must be consistent after transaction

Isolated : isolated from other processes (not influenced by eg other transaction)

Durable: data must be securely stored in the database

OAK 3
ACADEMY

JPA transactions

```
EntityTransaction tx = em.getTransaction();
tx.begin();
// Do something with database
tx.commit();
```

OAK 3
ACADEMY

Local VS Managed transactions

Local transactions: start and complete transaction yourself.

→ Standalone applications.

Managed transactions: Use Java Transaction API. Only applicable in a managed environment with JEE application server. (Not in the scope of the course)

OAK 3
ACADEMY

EntityManager: Possibilities

Save objects: void persist(Object o)

Find objects: T find(Class<T> entityClass, Object primaryKey)

Modify objects: T merge(T entity)

Delete objects: void remove(Object o)

OAK 3 ACADEMY

Example: exercise 2, p35

OAK 3
ACADEMY

Domain model

= objects with their mutual relations

Mapping:

1. **From domain model:** generate tables by JPA (only possible for new applications)
2. **From relational model:** Create Entity classes that reflect existing structure
3. **From the middle:** Own domain model that corresponds in background with relational model.

OAK 3
ACADEMY

Configuration of tables and columns

Annotations:

1. **@Table** → Adjust at table level of certain configuration. Eg name of the table.
2. **@Column** → Adjust the configuration of a column. Eg: unique, name ...

```
@Column(name = "andere_naam", unique = true)
private String text;
```

OAK 3
ACADEMY

Secondary Tables

→ If data is spread over multiple tables

EG:

```
@Entity
@SecondaryTable (name="otherTable")
public class Person {
  …
  @Column(table="otherTable")
  private String address;
  …
}
```

person

Id	Firstname	Lastname

other_table

Id	Address

OAK 3 ACADEMY

Mapping of datatypes

1. **Primary keys:** @Id.
2. **Single types**: base field
   @Basic(optional): (**true***/ false) → @Column(nullabale=false)
   → Hibernate handles both annotations in the same way
   @Basic(fetch): immediate or delayed reading from database (lazy/ **eager***)
   → Hibernate loads all the fields in the eager way
3. **Primitive types(boolean, float, int …)**
   1. Date and time(Date, Calendar, TimeStamp), from **HIBERNATE 5**.: LocalDate and LocalDateTime)
   2. Enum (@Enumerated(EnumType.ORDINAL) of @Enumerated(EnumType.STRING)
   3. Big objects: (@Lob)
   4. Exclude fields: virtual fields(@Transient)

* Default value

OAK 3 ACADEMY

Exercise 1, p51

- Use LocalDate for the birthday.
- Skip the version and the homepage
- Just use the same column name as the name of the field!

OAK 3
ACADEMY

Mapping datatypes

3. Value types: nested class

```
@Embeddable
public class Address { }
```

```
@Entity
public class Person {
  @embedded
  private Address = new Address();
}
```

OAK 3 ACADEMY

Exercise 2, p.54

OAK 3
ACADEMY

Mapping datatypes

4. Element Collections

```
@Entity
public class Person {
  @ElementCollection
  private List<String> hobbies = new ArrayList<>();
}
```

→ A separate table is created with a relationship between the two.

OAK 3 ACADEMY

Exercise

Add a list of Hobbies to the Person class.
Add a person to the database with some hobbies.

OAK 3
ACADEMY

Relations between entities

Domain model: references to other objects

Relational model: primary keys – foreign keys or relationship between tables.

→ JPA automatically ensures the translation between both

OAK 3
ACADEMY

Relations

One to one	Car → Engine
One to many	Work → Employees
Many to one	Shareholders → company
Many to many	Person→ club

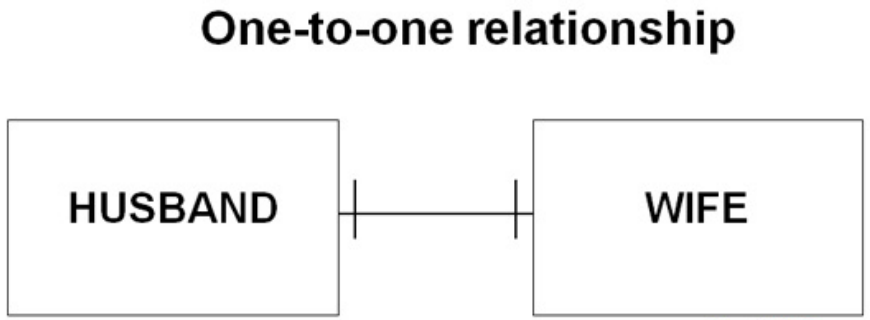
unidirectional

bidirectional

OAK 3
ACADEMY

One to One

= Unique relation between two objects



@OneToOne: the elements below can optionally be added

Element	*Description*
`cascade`	Indicates the cascade behaviour of the relationship.
`fetch`	Indicates the fetch type of the relationship.
`mappedBy`	Indicates the field that represents the relationship to the owner.
`optional`	Indicates if this relationship is optional. The default value is `true`.
`orphanRemoval`	Indicates if objects that are detached from the relationship have to be removed automatically. The default value is `false`.
`targetEntity`	Indicates the class of the related object. Only necessary if it can't be deducted from the variable or collection; e.g. a raw collection.

OAK 3 ACADEMY

Cascadetype

= Determine which entities of the Entity Manager must be executed on the related object.

→ Transmit entity changes from Parent to Child

Cascade type	***Omschrijving***
`CascadeType.ALL`	Alle operaties.
`CascadeType.MERGE`	Enkel *merge*-operaties.
`CascadeType.PERSIST`	Enkel *persist*-operaties.
`CascadeType.REFRESH`	Enkel *refresh*-operaties.
`CascadeType.REMOVE`	Enkel *remove*-operaties.
`CascadeType.DETACH`	Enkel *detach*-operaties.

OAK 3 ACADEMY

Fetch-type

= Determines how objects should be loaded

Fetch type	*Description*
`FetchType.LAZY`	The related object is loaded with a delay, i.e. when the object is requested.
`FetchType.EAGER`	The related object is loaded immediately. This is the default value of a one-to-one relationship.

Default one to one = EAGER

OAK 3
ACADEMY

Bidirectional mapping

mappedBy element

→ Place in the class that is not the owner.

OAK 3
ACADEMY

Example: Bidirectional mapping

```java
@Entity
public class Patient {
    @Id
    @GeneratedValue
    private long id;
    private String name;

    @OneToOne
    private MedicalFile medicalFile;
```

```java
@Entity
public class MedicalFile {
    @Id
    @GeneratedValue
    private long id;
    private float height;
    private float weight;

    @OneToOne(mappedBy="medicalFile")
    private Patient patient;
```

OAK 3 ACADEMY

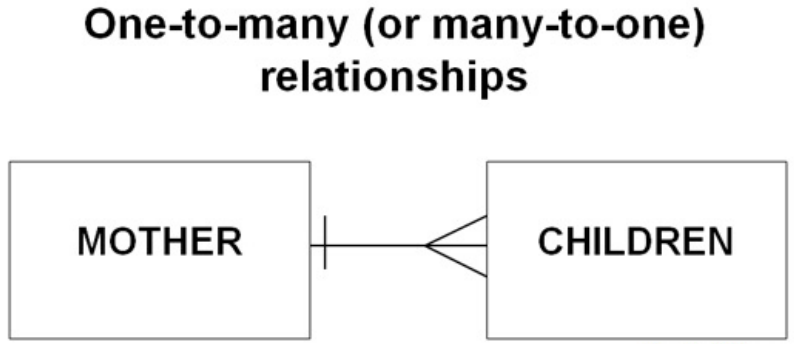
Exercise 4, p65

One to many and many to one

= object has relationship with several other objects

of the same type.

→ Collection, List, Set, Map



Bidirectional: @OneToMany in parent → @ManyToOne in child

One to many

Elements that can be optionally added in @OneToMany annotation.

@OneToMany

Element	***Description***
`cascade`	Indicates the cascade behaviour of the relationship. Not set by default.
`fetch`	Indicates the fetch type of the relationship. Default is `LAZY`.
`mappedBy`	Indicates the field that represents the relationship on the other side.
`orphanRemoval`	Indicates whether objects that are detached from the relationship have to be automatically removed. Default is `false`.
`targetEntity`	Indicates the class of the related object.

OAK 3
ACADEMY

Many to one

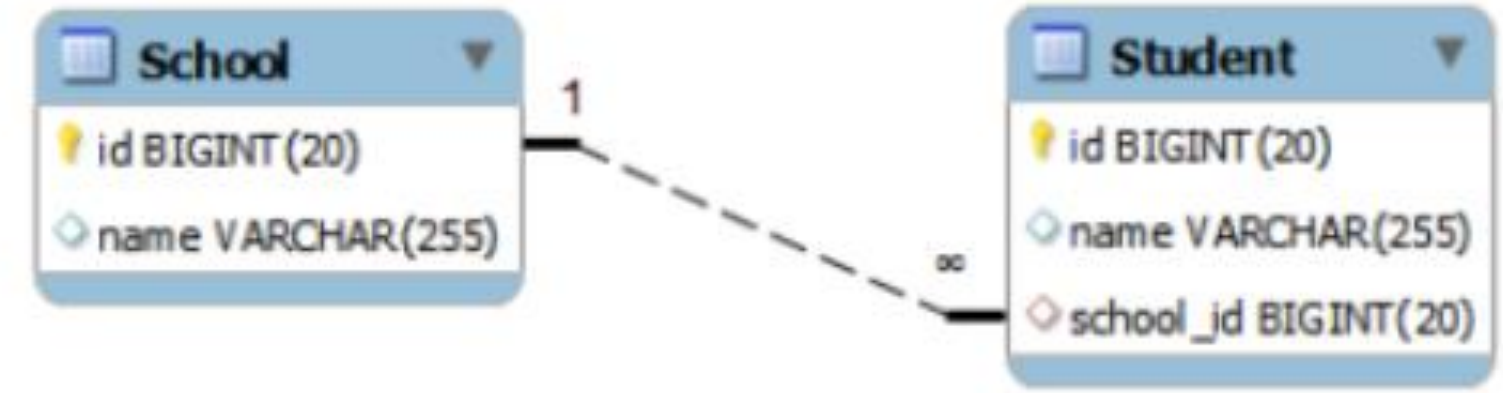
Elements that can optionally be added in @ManyToOne annotation.

@ManyToOne

Element	*Description*
`cascade`	Indicates the cascade behaviour of the relationship. Not configured by default.
`fetch`	Indicates the fetch type of the relationship. Default is `EAGER`.
`optional`	Indicates if this relationship is optional. Default is `true`.
`targetEntity`	Indicates the class of the related object.

OAK 3
ACADEMY

Exercise: create bidirectional relation

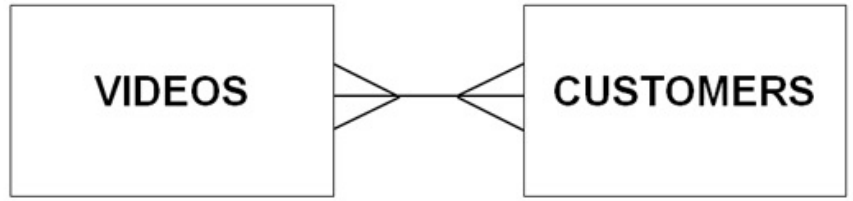


OAK 3 ACADEMY

Exercise 5, p71

Many to many

Multiple objects of the same type can have a relationship with type.



→ Additional relationship table required → @JoinTable

@JoinTable

Element	***Description***
`catalog`	The table catalog.
`inverseJoinColumns`	The column name of the foreign key in the table which is not the owner of the relationship.
`joinColumn`	The column name of the foreign key in the table which is the owner of the relationship.
`Name`	The table name.
`schema`	The table schema.
`uniqueConstraints`	Unique constraints to be forced upon the table.

OAK 3 ACADEMY

Inheritance

Domeinmodel: objects can be part of class hierarchy

Relationeel model: no inheritance possible

→ **Oplossing:** Special mapping techniques

OAK 3
ACADEMY

1. Single Table

= 1 table for the entire class hierarchy

Annotation @Inheritance (strategy = InheritanceType.SINGLE_TABLE)

→ **Advantage**: only one table needed

→ **Disadvantage**: columns must allow null values

OAK 3
ACADEMY

2. Table per subclass

Each concrete class is given a separate table (ie no abstract classes)

Annotation @Inheritance (strategy = InheritanceType.JOINED)

→ **Advantage**: tables fully normalized and do not contain empty values

→ **Disadvantage**: Many join operations are required when requesting data

OAK 3
ACADEMY

3. Table per concrete class

Each concrete class is given a separate table (ie no abstract classes)

Annotation @Inheritance (strategy = InheritanceType.TABLE_PER_CLASS)

→ **Advantage**: impose restrictions on the various columns

→ **Disadvantage**: Not fully normalized and also many join operations needed.

OAK 3
ACADEMY

Exercise

OAK 3
ACADEMY

Searches

1. Through Query API and JPQL
2. Through Criteria API
3. Through native SQL queries

OAK 3 ACADEMY

Query Api and JPQL

JPQL= Java Persistence Query Language

Bv:

```
Query query = em.createQuery("select p from Person as p");
List<Person> persons = (List<Person>) query.getResultList();
```

Without casting:

```
TypedQuery<Person> query =
        em.createQuery("select p from Person as p",Person.class);
List<Person> persons = query.getResultList();
```

OAK 3
ACADEMY

Named Queries

= Dynamic query → Enter string with content search

```
@NamedQuery(name="findAll", query="select p from Person as p")
@Entity
public class Person {
    ...
}
```

Op de plaats waar we de zoekoperatie willen uitvoeren, creëren we de *query* als volgt:

```
public class FindPerson {
    ...
       TypedQuery<Person> query =
                        em.createNamedQuery("findAll",Person.class);
       List<Person> persons = query.getResultList();
       return persons;
    ...
}
```

OAK 3 ACADEMY

JPQL

= **J**ava **P**ersistence **Q**uery **L**anguage

→ Focused on domain model with Java object instead of on the relational model.

```
select p from Person as p where p.lastName = 'Simpson'
```

projection *selection* *restriction*

OAK 3 ACADEMY

JPQL: selectie (FROM) en projectie (SELECT)

select e from entityName as e

→ Naam moet overeenkomen met het element name van de annotatie @Entity

select e1, e2 from entityName e1, entityName e2

→ Meervoudige selectie (**as** is optioneel)

select distinct e form entityName e

→ Uitsluiten van duplicaten

OAK 3
ACADEMY

JPQL: selectie (FROM) en projectie (SELECT)

select p.name, p.age from Person as p

→ Ipv volledig object, enkel een aantal attributen

select p.address.street from Person p

→ Navigeren naar attributen van gerelateerd object

select count(p) form Person p

→ Geeft het aantal personen terug

OAK 3
ACADEMY

JPQL: restrictie (WHERE)

select p from Person as p where p.age = 25

→ Criteria opleggen bij de zoekopdracht

```
Query query = entityManager.createQuery
              ("select p from Person as p where p.lastName=:last");
query.setParameter("last", "Simpson");
```

→ named parameter gebruiken

```
Query query = entityManager.createQuery
              ("select p from Person as p where p.lastName=?1");
query.setParameter(1, "Simpson");
```

→ index parameter gebruiken

OAK 3 ACADEMY

JPQL: restrictie (WHERE)

select p from Person as p where p.gender = persons.GenderType.Female

→ Criteria met een enum

select p from Person as p where p.birthday = '2011-03-12'

→ Criteria met een datum

select p from Person as p where p.name is not empty

→ Criteria met een operator

OAK 3 ACADEMY

JPQL

select concat(p.firstName, p.lastName) from Person p

→ select met een functie

select p from Person as p order by p.lastName asc

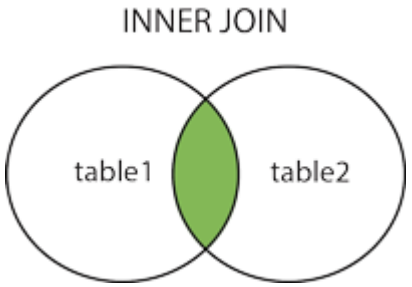
→ sorteren van verzamelingen

OAK 3
ACADEMY

JPQL: joins

select p from Person p inner join p.address a where p.age > 25

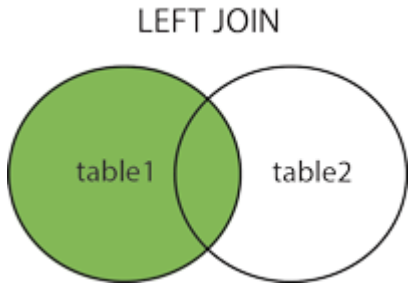
→ inner join voorbeeld (objecten waarvoor relatie bestaat)



select s, st from School s left join s.students st on st.age < 30

→ left join voorbeeld (objecten selecteren waarvoor geen relatie bestaat)

→ null waarden mogelijk



OAK 3 ACADEMY

JPQL

select value(s) from Company c, join c.staff s where key(s.staff) = StaffRole.CEO

→ value: selecteert de waarde

→ key: selecteert de sleutel

→ Specifiek voor een Map

select p from Person p where p.birthDay= (select min(p.birthDay) from Person p)

→ voorbeeld met een subquery

OAK 3
ACADEMY

Opdracht

OAK 3
ACADEMY

Native Query

= query uitvoeren in pure SQL.

```
Query query  =
    em.createNativeQuery("select * from Message",Message.class);
List<Message> result = (List<Message>)query.getResultList();
for(Message m: result) {
   System.out.println(m.getText());
}
```

OAK 3
ACADEMY

Opdracht

OAK 3
ACADEMY

Criteria API

JPQL

```
select p from Person p
where p.age > 25
```

CRITERIA

= zoekopdracht volledig programmatisch samenstellen.

```
CriteriaBuilder cb = em.getCriteriaBuilder();

CriterialQuery<Person> q = cb.createQuery(Person.class);
Root<Person> p = q.from(Person.class); //SELECTION
q.select(p); // PROJECTION
q.where(cb.greaterThan(p.get("age"), 25); // RESTRICTION
```

Opdracht

OAK 3
ACADEMY

Callbacks en listeners

Object op de hoogte brengen van een wijziging → Callback methoden

Voorwaarden:

1) Void
2) Geen checked exceptions
3) Geen parameters
4) Gemarkeerd met specifieke annotatie

Annotatie	***Moment***
`@PrePersist`	Net voordat het object gecreëerd wordt in de databank.
`@PostPersist`	Net nadat het object gecreëerd werd in de databank.
`@PostLoad`	Net nadat het object geladen wordt uit de databank.
`@PreUpdate`	Net voordat het object wordt weggeschreven naar de databank.
`@PostUpdate`	Net nadat het object wordt weggeschreven naar de databank.
`@PreRemove`	Net voordat het object verwijderd wordt uit de databank.
`@PostRemove`	Net nadat het object verwijderd werd uit de databank.

OAK 3 ACADEMY

Exercise

OAK 3
ACADEMY