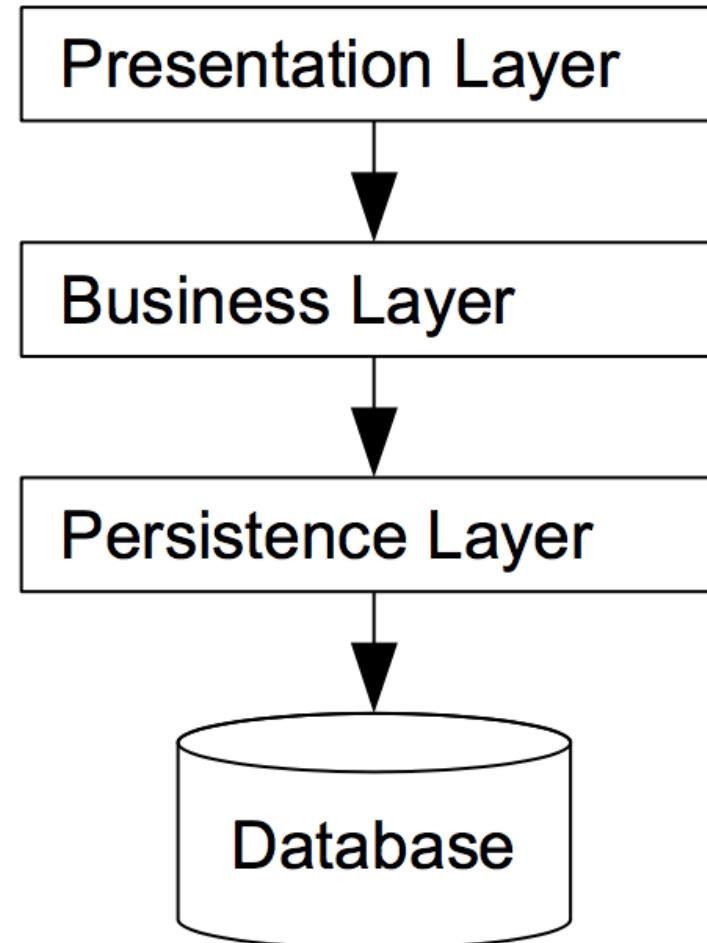


# Programming Advanced java

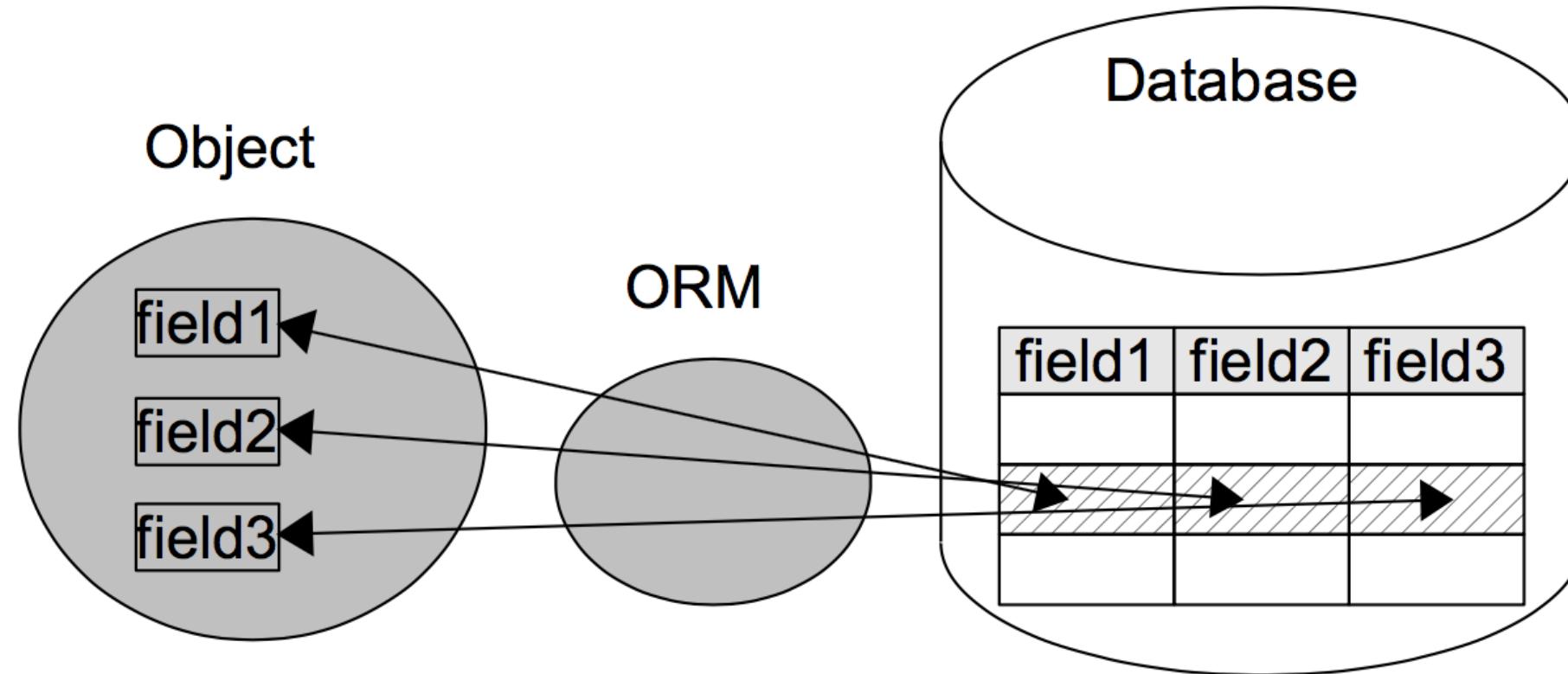
JPA



# Layer architecture



# Object Relation Mapping (ORM)



# Object Relation Mapping (ORM)

DAO:

- Veel boilerplate code
- Veel maintenance

→ Enterprise JavaBeans 2.0

- Entity Beans
- Runtime container nodig
- Application Server
- Log en omslachtig in gebruik

→ POJO → Hibernate / TopLink

→ Enterprise JavaBeans 3.0

- Java Persistence API (JPA)
- persistence layer gebruik makend van POJO's

# JPA

→ Enterprise JavaBeans 3.0

→ Java Persistence API (JPA)

- persistence layer gebruikmakend van POJO's
- specificatie
- implementaties
  - Hibernate
  - TopLink
  - EclipseLink
  - OpenJPA
  - ...

→ JPA 2.2 met Hibernate als implementatie

# JPA - Hibernate

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.4.12.Final</version>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.47</version>
</dependency>
```

# JPA – Hibernate - Configuratie

- **Database configuratie**

→ META-INF/persistence.xml

- databaseschema
- username
- password

- **Mapping configuratie** (relationele tabellen <-> java-objecten)

→ Deployment descriptor

- META-INF/orm.xml
- scheiding code en configuratie
- bestaande klassen gebruiken zonder code aan te passen

→ Annotations

- makkelijk en snel in gebruik
- java code

# persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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"
    version="2.1">

    <persistence-unit name="example">
        <properties>
            <!-- JPA standard -->
            <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/Example" />
            <property name="javax.persistence.jdbc.user" value="..." />
            <property name="javax.persistence.jdbc.password" value="..." />

            <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />

            <!-- Hibernate specific -->
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.xxx" value="xxx"/>
        </properties>
    </persistence-unit>
</persistence>
```

# Entity class

**@Entity**

```
public class Message {
```

**@Id**

```
private long id;  
private String text;
```

**public Message() {  
}**

```
public Message(long id, String text) {  
    this.id = id;  
    this.text = text;  
}
```

```
// getters and setters
```

```
...
```

```
}
```

# Primary keys - datatypes

- primitive (byte, int, short, long, char, float, double)
- primitive wrapper-class
- String
- Big numeric type (BigInteger)
- Date

# Primary keys - type

- Natuurlijke primary key
  - Key met betekenis in de value
  - vb. rijksregisternummer
  - Indien mogelijk vermijden
- Surrogaat primary key
  - Oplopende nummer
  - Geen logische betekenis
  - 1. Enkelvoudig (@Id)
  - 2. Samengesteld (@IdClass)
  - 3. Autogenerated (@GeneratedValue)

# Primary keys - autogenerated

@GeneratedValue(strategy=GenerationType.AUTO)

Strategie	Omschrijving
IDENTITY	Er wordt gebruikgemaakt van een <i>identity column</i> in de databank. Dit is een kolom waarvan de inhoud automatisch ophoogt bij de creatie van een record ( <i>auto-increment</i> ).
SEQUENCE	Er wordt gebruikgemaakt van een <i>sequence</i> in de databank.
TABLE	Er wordt gebruikgemaakt van een aparte tabel waarin telkens de hoogste waarde van de <i>primary key</i> bewaard wordt.
AUTO	De JPA-provider kiest zelf de beste strategie, rekening houdend met de mogelijkheden van de onderliggende databank. Dit is tevens de standaardwaarde.

# Entity usage

```
EntityManagerFactory entityManagerFactory =  
Persistence.createEntityManagerFactory("example");
```

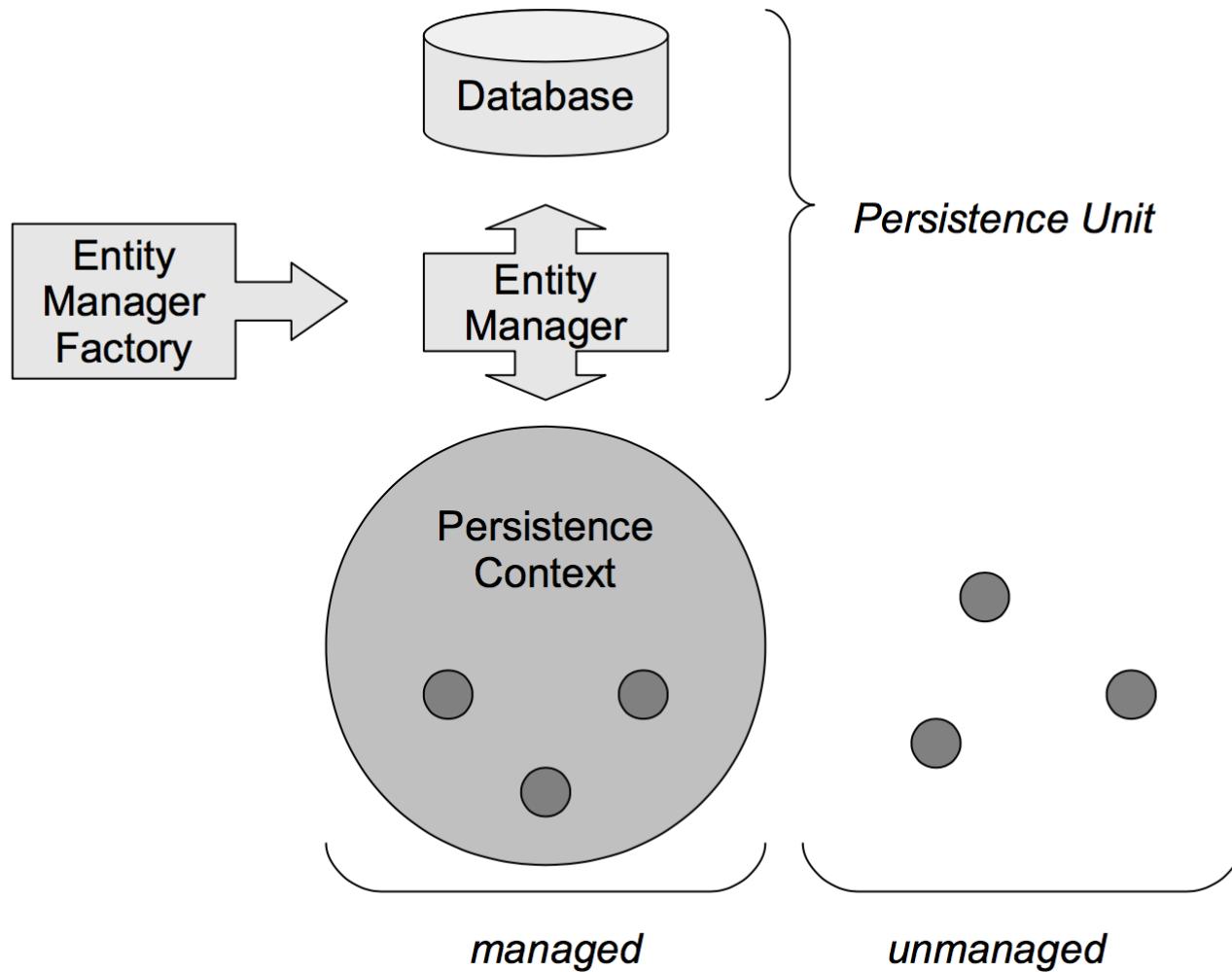
```
EntityManager entityManager =  
entityManagerFactory.createEntityManager();
```

```
EntityTransaction transaction =  
entityManager.getTransaction();
```

```
transaction.begin();  
Message message = new Message(1, "Hello Hibernate");  
entityManager.persist(message);  
transaction.commit();
```

```
entityManager.close();  
entityManagerFactory.close();
```

# Persistence Context



# Persistence Context

```
package messages;
import javax.persistence.*;

public class ChangeMessage {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("course");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();
        tx.begin();
        Message message = em.find(Message.class, 1L);
        message.setText("Hello Mars"); // managed
        tx.commit();
        em.close();
        message.setText("Hello Venus"); // unmanaged
        emf.close();
    }
}
```

# JPA Transactions

- Alle wijzigingen aan een object binnen een persistence context kunnen enkel via een transactie naar de databank weggeschreven worden
  - transaction.begin()
  - transaction.commit()
  - transaction.rollback()
- Wijzigingen binnen een persistence context maar buiten een transactie worden pas in rekening gebracht bij de eerstvolgende voltooiing van een transactie

# JPA Transactions - flush

- Wijzigingen doorvoeren binnen een transactie alvorens de commit
- FlushMode
  - FlushModeType.AUTO
    - Wegschrijven na commit
    - Wegschrijven net voor zoekoperatie
    - Default
  - FlushModeType.COMMIT
    - Wegschrijven enkel na commit
    - Onnodige databaseoperaties worden vermeden

# Entity Manager

persist	Toevoegen van een object aan de persistence context Wegschrijven naar de database is afhankelijk van de implementatie. → Rekening houden met automatisch gegenereerde primary keys
find	Opzoeken record en toevoegen aan de persistence context
merge	Een unmanaged object samenvoegen met een managed object
remove	Verwijderen van een object uit de persistence context en database
refresh	Gegevens van een managed object opnieuw uitlezen uit de database
clear	Leegmaken van de persistence context.
detach	Een managed object unmanaged maken
contains	Nagaan of een object zicht in de persistence context bevindt

# JPA – extra configuratie

## Tabel configuratie

```
@Entity  
@Table(name="PERSONS")  
public class Person {  
    ...  
}
```

<b>Element</b>	<b>Omschrijving</b>
catalog	De catalog van de tabel.
name	De naam van de tabel.
schema	Het schema van de tabel
uniqueConstraints	Reeks van <i>unique constraints</i> .
indexes	De lijst van indexen die voor deze tabel gemaakt moeten worden.

# JPA – extra configuratie

## Tabel configuratie

```
@Entity  
@Table(name = "PERSONS" ,  
       indexes={@Index(name="LAST_NAME_INDEX",  
                         columnList="LAST_NAME") })
```

```
public class Person {  
    ...  
}
```

```
@Entity  
@Table(name="PERSONS",  
       uniqueConstraints = {  
           @UniqueConstraint(columnNames="firstName")  
       })
```

# JPA – extra configuratie

## Kolom configuratie

```
@Id  
@GeneratedValue  
private long id;  
@Column (name="FIRST_NAME")  
private String firstName;  
@Column (name="LAST_NAME")  
private String lastName;
```

Element	Omschrijving
columnDefinition	SQL-fragment dat gebruikt wordt bij de DDL voor de kolom.
insertable	Geeft aan of dit veld wordt opgegeven bij het invoegen van een nieuw record. Standaard is true.
length	De lengte van de kolom.
name	De naam van de kolom.
nullable	Geeft aan of de kolom NULL-waarden mag bevatten. Standaard is true.
precision	De nauwkeurigheid van getallen in geval van een decimale waarde.
scale	De schaal van getallen in geval van een decimale waarde.
table	De naam van de (secundaire) tabel die dit veld bevat.
unique	Geeft aan of dit veld uniek moet zijn. Standaard false.
updatable	Geeft aan of dit veld aangepast moet worden bij een update. Standaard true.

# JPA – mapping van datatypes

## Datatypes

→ Aanpassen gegenereerd datatype

- Primitieve datatypes  
→ `@Column(columnDefinition=sql-data-type)`
- Datum en tijden  
→ `@Temporal(TemporalType.DATE)`
- Het opsommingstype  
→ `@Enumerated(EnumType.STRING)`
- Grote objecten  
→ `@Lob`
- Speciale datatypes  
→ `@Transient` : uitsluiten van persistentie

# JPA – mapping van datatypes

- Datum en tijden  
→ @Temporal(TemporalType.DATE)

<b>Java-type</b>
java.util.Date
java.util.Calendar
java.sql.Timestamp

<b>Annotatie</b>	<b>Omschrijving</b>
@Temporal(TemporalType.DATE)	Enkel de datum.
@Temporal(TemporalType.TIME)	Enkel de tijd.
@Temporal(TemporalType.TIMESTAMP)	Datum en tijd.

# JPA – mapping van datatypes

- Het opsommingstype  
→ @Enumerated(EnumType.STRING)

```
package persons;  
  
public enum GenderType {  
    MALE, FEMALE  
}
```



```
@Enumerated(EnumType.STRING)  
private GenderType gender;
```

Annotatie	Omschrijving
@Enumerated(EnumType.ORDINAL)	Het volgnummer. Standaardwaarde.
@Enumerated(EnumType.STRING)	De <i>string</i> -representatie.

# JPA – relaties

## Soorten:

- One to one
- One to many
- Many to one
- Many to many

## Type Navigeerbaarheid:

- unidirectional
- bidirectional

# JPA – relaties – one to one

```
package medical;
import javax.persistence.*;

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

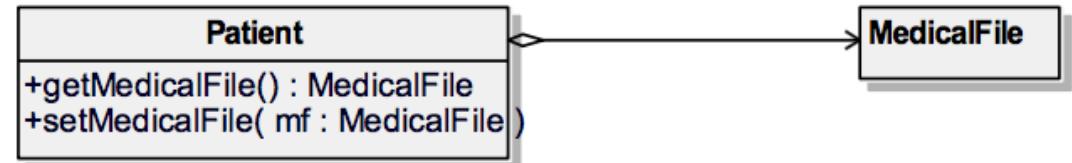
    @OneToOne
    private MedicalFile medicalFile;

    public MedicalFile getMedicalFile() {
        return medicalFile;
    }

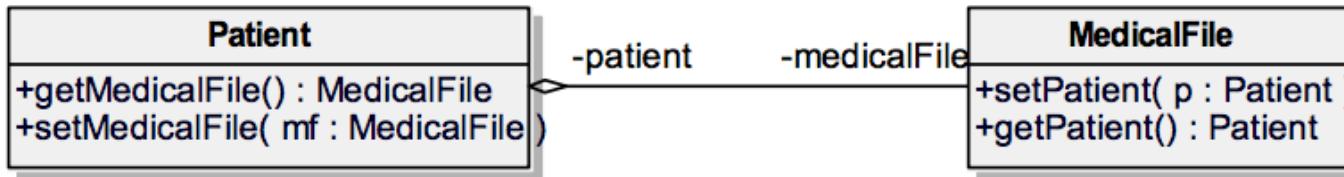
    public void setMedicalFile(MedicalFile medicalFile) {
        this.medicalFile = medicalFile;
    }

    public long getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```



# JPA – relations – one to one bidirectional



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

public Patient getPatient() {
    return patient;
}

public void setPatient(Patient patient) {
    this.patient = patient;
}
```

# JPA – relaties – one to one annotation elementen

<b>Element</b>	<b>Omschrijving</b>
cascade	Geeft het cascade-gedrag van de relatie aan.
fetch	Geeft het <i>fetch</i> -type van de relatie aan.
mappedBy	Geeft het veld aan dat bij de eigenaar de relatie vertegenwoordigt.
optional	Geeft aan of deze relatie optioneel is. Standaard true.
orphanRemoval	Geeft aan of objecten die losgekoppeld worden van de relatie automatisch verwijderd moeten worden. Standaard false.
targetEntity	Geeft de klasse aan van het gerelateerde object. Dit is enkel nodig indien dit niet duidelijk is aan de hand van de variabele of verzameling; bijvoorbeeld bij een <i>raw collection</i> .

# JPA – relaties – one to one cascadetype

<b>Cascade type</b>	<b>Omschrijving</b>
CascadeType.ALL	Alle operaties.
CascadeType.MERGE	Enkel <i>merge</i> -operaties.
CascadeType.PERSIST	Enkel <i>persist</i> -operaties.
CascadeType.REFRESH	Enkel <i>refresh</i> -operaties.
CascadeType.REMOVE	Enkel <i>remove</i> -operaties.
CascadeType.DETACH	Enkel <i>detach</i> -operaties.

```
@OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})
```

Default: geen cascading

# JPA – relaties – one to one fetch-type

Fetch type	Omschrijving
FetchType.LAZY	Het gerelateerde object wordt vertraagd geladen wat wil zeggen pas op het moment dat het object wordt opgevraagd
FetchType.EAGER	Het gerelateerde object wordt onmiddelijk geladen. Default

```
@OneToOne(fetch=FetchType.LAZY)
```

# JPA – relaties

## One to many - Many to one

Verzameling referentie	Omschrijving
Collection	Algemene verzameling
List	Geordend Gesorteerd → @OrderBy
Set	Geen duplicaten
Map	Elementen opvragen via een veld

# JPA – relaties

## One to many - Many to one

### @OneToMany

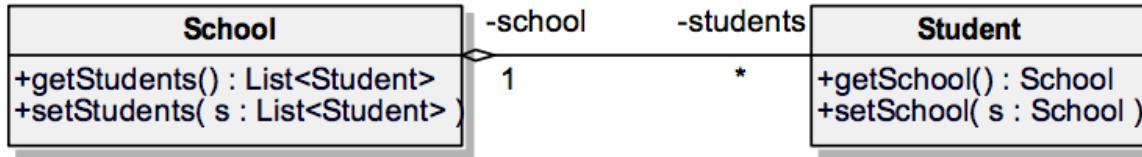
Element	Omschrijving
cascade	Geeft het cascadegedrag van de relatie aan. Standaard niet ingesteld.
fetch	Geeft het <i>fetch</i> -type van de relatie aan. Standaard LAZY.
mappedBy	Geeft het veld aan dat aan de andere kant de relatie vertegenwoordigt.
orphanRemoval	Geeft aan of objecten die losgekoppeld worden van de relatie automatisch verwijderd moeten worden. Standaard false.
targetEntity	Geeft de klasse aan van het gerelateerde object.

### @ManyToOne

Element	Omschrijving
cascade	Geeft het cascadegedrag van de relatie aan. Standaard niet ingesteld.
fetch	Geeft het <i>fetch</i> -type van de relatie aan. Standaard EAGER.
optional	Geeft aan of deze relatie optioneel is. Standaard true.
targetEntity	Geeft de klasse aan van het gerelateerde object.

# JPA – relaties

## One to many - Many to one



```
@Entity
public class School {
    @Id
    @GeneratedValue
    private long id;

    private String name;

    @OneToMany(mappedBy="school")
    private List<Student> students = new ArrayList<>();

    public List<Student> getStudents() {
        return students;
    }

    public void setStudents(List<Student> students) {
        this.students = students;
    }
}
```

```
@Entity
public class Student {
    @Id
    @GeneratedValue
    private long id;

    private String name;

    @ManyToOne
    private School school;

    public School getSchool() {
        return school;
    }

    public void setSchool(School school) {
        this.school = school;
    }
}
```

# JPA – relaties

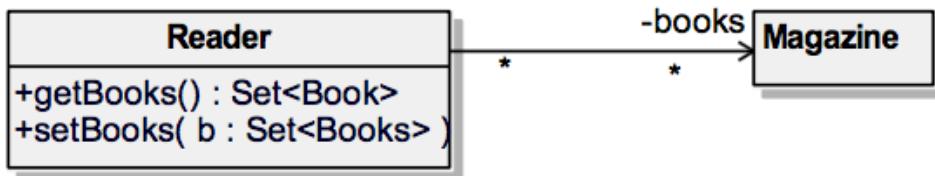
## Many to many

`@JoinTable`

<b>Element</b>	<b>Omschrijving</b>
catalog	De catalog van de tabel.
inverseJoinColumns	De kolomnaam van de <i>foreign key</i> in de tabel die niet de eigenaar is van de relatie.
joinColumns	De kolomnaam van de <i>foreign key</i> in de tabel die de eigenaar is van de relatie.
name	De naam van de tabel.
schema	Het schema van de tabel.
uniqueConstraints	<i>Unique constraints</i> die opgelegd moeten worden aan de tabel.

# JPA – relaties

## Many to many



```
@Entity
public class Reader {
    @Id
    @GeneratedValue
    private long id;

    private String name;

    @ManyToMany
    private Set<Magazine> magazines = new HashSet<>();
```



# JPA – zoekopdrachten

- **Query API en JPQL** (Java Persistence Query Language)
  - Query / TypedQuery
  - Named queries
- Criteria API
- Native SQL queries

# JPA – zoekopdrachten

## Query / TypedQuery

### Query

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

### TypedQuery

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

# JPA – zoekopdrachten

## NamedQuery - static

### annotation

```
@NamedQuery(name="findAll", query="select p from Person as p")
@Entity
```

### orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
    http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd ">
    <named-query name="findAll">
        <query>select p from Person p</query>
    </named-query>
</entity-mappings>
```

### usage:

```
EntityManagerFactory emf = Persistence
    .createEntityManagerFactory("course");
EntityManager em = emf.createEntityManager();
Query query = em.createQuery("select p from Person p");
emf.addNamedQuery("findAll",query);
```

# JPA – zoekopdrachten

## NamedQuery - dynamic

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

# JPA – zoekopdrachten parameters

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

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