

# Java Enterprise Edition (JEE)

## Chapitre 05 : ORM et DAO avec Spring Data

Sébastien Chèvre

- Sémantique des diapositives



À savoir théoriquement (TE)



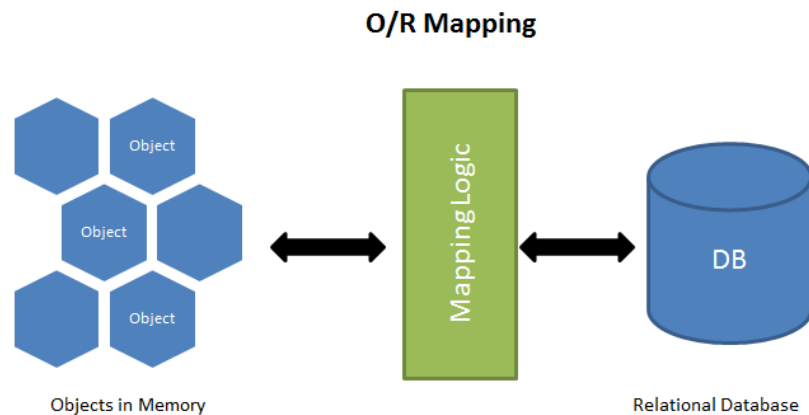
Sensibilisation et aspect pratiques

# ORM



- **Object Relational Mapping**

- Conversion objet POO  $\leftrightarrow$  enregistrement DB
- Réduit de la quantité de code
- Homogénéité globale d'accès aux données



# ORM – Avantages/Inconvénients



- **Avantages**

- Portable. Abstraction de l'implémentations SQL
- Simplification des données liées (l'ORM se charge des relations)
- Langage propre à l'implémentation ORM (plus de SQL)
- Le traitement des INSERT et des UPDATE est souvent le même

- **Inconvénients**

- Moins rapides que du SQL brut (couche logicielle en plus)
- Optimisation des requêtes limitées à l'outil ORM
- Requêtes complexes peuvent être limitées
- Prise en main de concepts nouveaux, hors SQL

# DAO



- **Data Access Object**

- Objet d'accès aux données
- Patron de conception, complétant le modèle MVC
- Abstraction de la manière dont sont récupérés/stockés les données

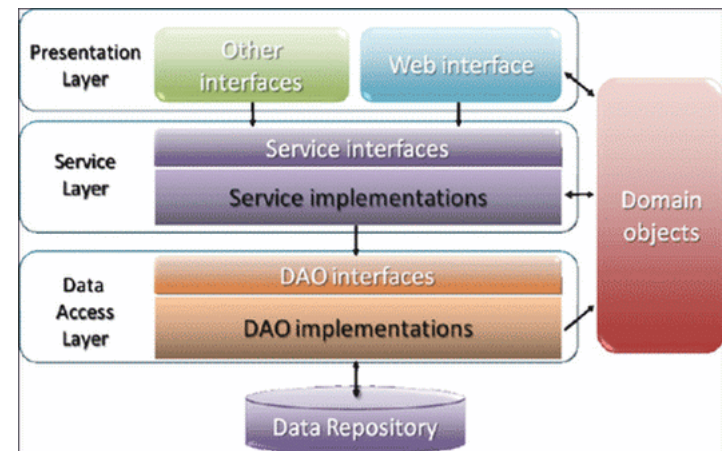
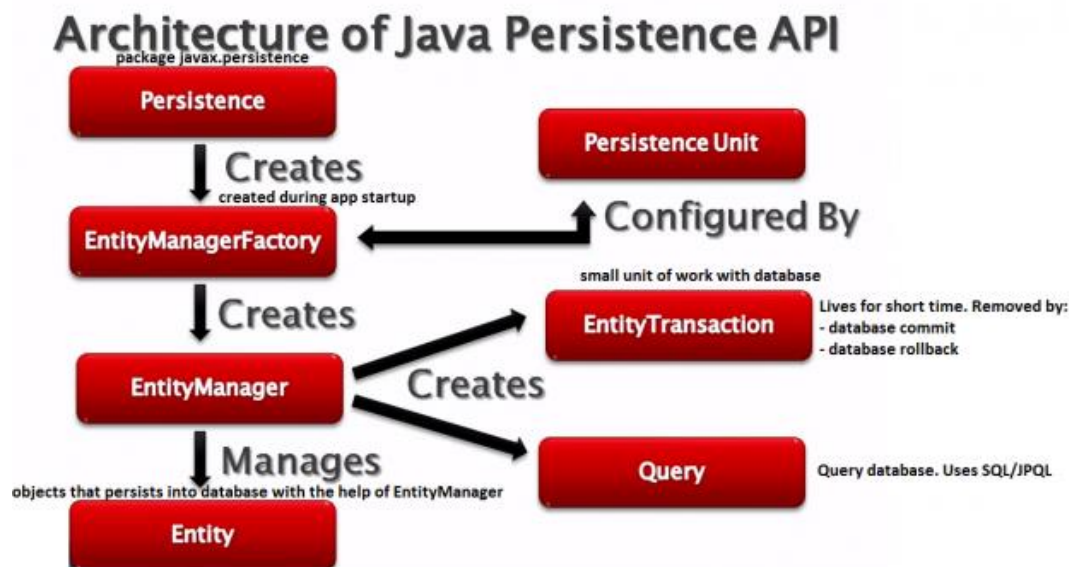


Figure 1

# JPA – Java Persistence API



- Interface de programmation Java (abstraction)
- JSR-220
- Définit le contrat minimal ORM dans le monde Java



# JPA – Implémentations



- Hibernate
- EclipseLink
- OpenJPA
- DataNucleus
  
- Alternative à JPA (hybride)
  - **MyBatis**
  - **Jooq**

# Spring Data

<https://spring.io/projects/spring-data>

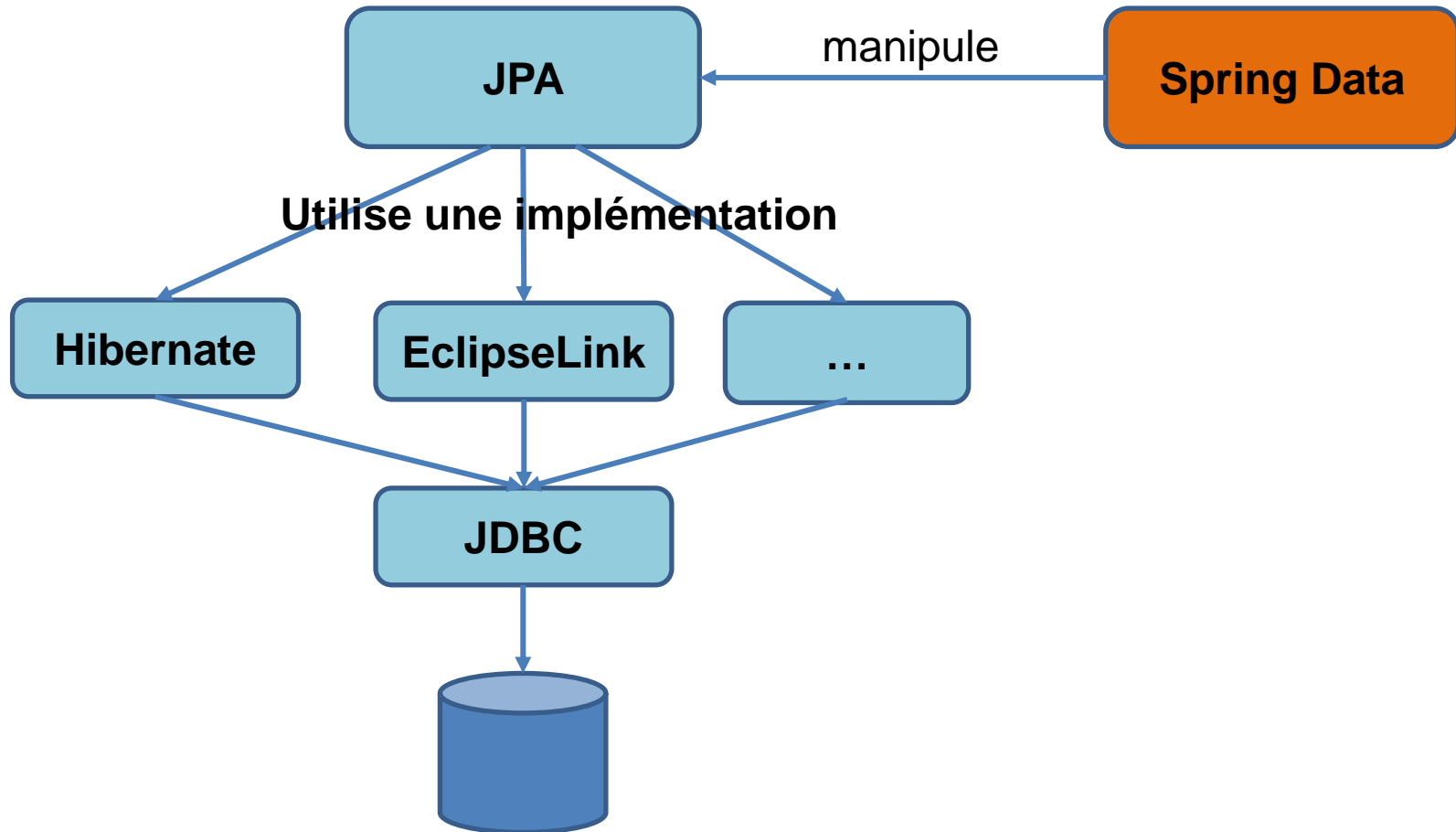


## Spring Data



- Ajoute une couche d'abstraction supplémentaire à JPA
- Implémentations « automatique » des opérations de bases de données courantes (CRUD)
- Concentration sur la valeur ajoutée: l'écriture des requêtes
- Utilise **Hibernate** comme implémentation par défaut (configurable)

# Spring Data



# Repository, DAO, ...



- Deux concepts proches...
- Abstraction des détails de persistance
- **DAO**  
*Data Centric, manipule des objets de persistance  
(mapping tables db)*
- **Repository**  
*Domain Centric, manipule des  
objets métiers. Concept plus haut niveaux*

# Repository



abstraction  
↓

## Spring Data Commons

Repository

Aucune méthodes de base

CrudRepository

Méthodes CRUD (save, find, ...)

PagingAndSortingRepository

Pagination et tri

## Spring Data JPA

JpaRepository

Méthodes Jpa

# CrudRepository



Modifier and Type	Method and Description
long	<a href="#"><u>count()</u></a> Returns the number of entities available.
void	<a href="#"><u>delete(T)</u></a> entity)Deletes a given entity.
void	<a href="#"><u>deleteAll()</u></a> Deletes all entities managed by the repository.
void	<a href="#"><u>deleteAll(Iterable&lt;? extends T&gt;)</u></a> entities)Deletes the given entities.
void	<a href="#"><u>deleteById(ID)</u></a> id)Deletes the entity with the given id.
boolean	<a href="#"><u>existsById(ID)</u></a> id)Returns whether an entity with the given id exists.
<a href="#"><u>Iterable&lt;T&gt;</u></a>	<a href="#"><u>findAll()</u></a> Returns all instances of the type.
<a href="#"><u>Iterable&lt;T&gt;</u></a>	<a href="#"><u>findAllById(Iterable&lt;ID&gt;)</u></a> ids)Returns all instances of the type with the given IDs.
<a href="#"><u>Optional&lt;T&gt;</u></a>	<a href="#"><u>findById(ID)</u></a> id)Retrieves an entity by its id.
<S extends <a href="#"><u>T</u></a> > S	<a href="#"><u>save(S)</u></a> entity)Saves a given entity.
<S extends <a href="#"><u>T</u></a> > <a href="#"><u>Iterable&lt;S&gt;</u></a>	<a href="#"><u>saveAll(Iterable&lt;S&gt;)</u></a> entities)Saves all given entities.

# PagingAndSortingRepository



Modifier and Type	Method and Description
<a href="#"><u>Page</u></a> < <a href="#"><u>T</u></a> >	<a href="#"><u>findAll</u></a> ( <a href="#"><u>Pageable</u></a> pageable)Returns a <a href="#"><u>Page</u></a> of entities meeting the paging restriction provided in the Pageable object.
<a href="#"><u>Iterable</u></a> < <a href="#"><u>T</u></a> >	<a href="#"><u>findAll</u></a> ( <a href="#"><u>Sort</u></a> sort)Returns all entities sorted by the given options.

# JpaRepository



Type	Method and Description
void	<a href="#"><u>deleteAllInBatch()</u></a> Deletes all entities in a batch call.
void	<a href="#"><u>deleteInBatch(Iterable&lt;T&gt; entities)</u></a> Deletes the given entities in a batch
<a href="#"><u>List&lt;T&gt;</u></a>	<a href="#"><u>findAll()</u></a>
<S extends <a href="#"><u>T</u></a> > <a href="#"><u>List&lt;S&gt;</u></a>	<a href="#"><u>findAll(Example&lt;S&gt; example)</u></a>
<S extends <a href="#"><u>T</u></a> > <a href="#"><u>List&lt;S&gt;</u></a>	<a href="#"><u>findAll(Example&lt;S&gt; example, Sort sort)</u></a>
<a href="#"><u>List&lt;T&gt;</u></a>	<a href="#"><u>findAll(Sort sort)</u></a>
<a href="#"><u>List&lt;T&gt;</u></a>	<a href="#"><u>findAllById(Iterable&lt;ID&gt; ids)</u></a>
void	<a href="#"><u>flush()</u></a> Flushes all pending changes to the database.
<a href="#"><u>T</u></a>	<a href="#"><u>findOne(ID id)</u></a> Returns a reference to the entity with the given identifier.
<S extends <a href="#"><u>T</u></a> > <a href="#"><u>List&lt;S&gt;</u></a>	<a href="#"><u>saveAll(Iterable&lt;S&gt; entities)</u></a>
<S extends <a href="#"><u>T</u></a> >	<a href="#"><u>saveAndFlush(S entity)</u></a> Saves an entity and flushes changes instantly.

# Annotations de bases

Référentiel complet:

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



## @Entity



*Annotation permettant de spécifier à JPA que la classe en question est une entité JPA. C'est l'annotation de base permettant de rendre une classe éligible pour JPA.*

- **Utilisation**
  - classe
- **Obligatoire**
  - oui
- **Attributs**
  - name: redéfinit le nom de l'entité (par défaut le nom de la classe)

## @Table



*Permet de redéfinir le nom de la table associé à l'entité (par défaut c'est le nom de la classe)*

- **Utilisation**
  - classe
- **Obligatoire**
  - non
- **Attributs**
  - name: définit le nom de la table à laquelle l'entité sera mappé

## @Column



*Permet de redéfinir le nom de la colonne sur laquelle sera mappé le champ (par défaut c'est le nom du champ)*

- **Utilisation**
  - champ
- **Obligatoire**
  - non
- **Attributs principaux**
  - name: définit le nom de la colonne à laquelle l'entité sera mappé
  - nullable: spécifie si le champ peut être null (par défaut)

@Id



*Définit le (ou les) champ qui sera utilisé en tant que clé de la table*

- **Utilisation**
  - champ
- **Obligatoire**
  - oui
- **Attributs**
  - aucun

# Relations: les bases

# Relations



- **@OneToOne :** relation 1:1
- **@OneToMany:** relation 1:n
- **@ManyToOne:** relation n:1
- **@ManyToMany:** relation n:p

# Relations – rappel OO



## • Association

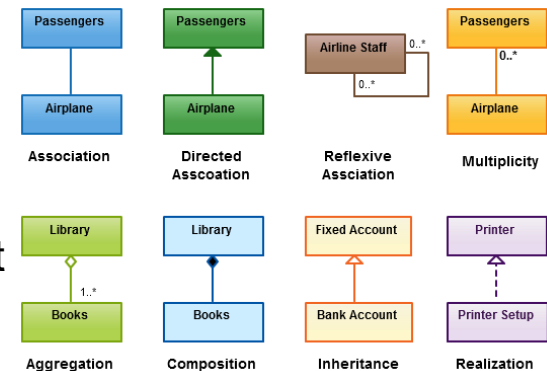
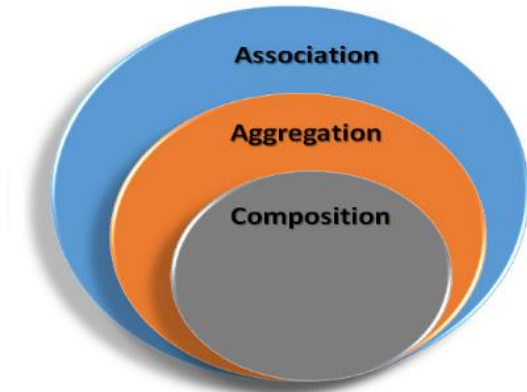
- Chaque instance possède son cycle de vie
- Pas de propriétaire
- *Exemple: Elève - Professeur*

## • Agrégation

- Chaque instance possède son cycle de vie
- Relation parent – enfant
- *Exemple: Classe - Professeur*

## • Composition

- Le cycle de vie de l'objet enfant dépend du parent
- Relation parent – enfant
- *Exemple: Maison - Chambre*



# @OneToOne



*Définit une relation 1 – 1 entre deux entités*



```
@Entity
public class Employee {
```

```
    @Id
    private Long id;
```

```
    private String nom;
```

```
    @OneToOne
    private Bureau bureau;
```

```
@Entity
public class Bureau {
```

```
    @Id
    private Long id;
```

```
    private String position;
```

EMPLOYEE

ID

NOM

BUREAU\_ID

BUREAU

ID

POSITION

Indexes



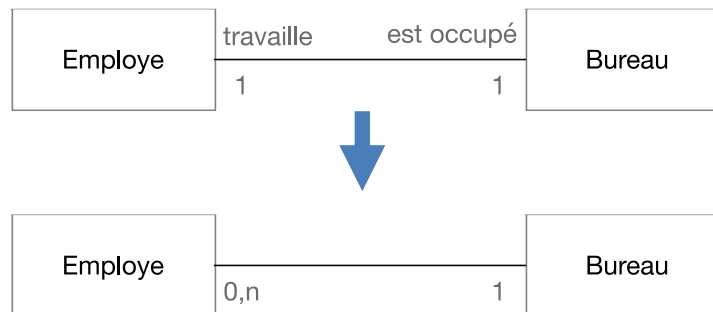
# @OneToOne



Résultat en base de données (implémentation):

ID	POSITION
1	gauche

ID	NOM	BUREAU_ID
1	seb	1
2	seb2	1
3	seb3	null



## → Association

Les entités possèdent un cycle de vie propre et sont indépendantes dans la relation

- Bureau occupé par plusieurs employés?
- Employé sans bureaux?
- Bureau sans employé?

## @OneToOne



- Bureau occupé par plusieurs employés? **NON**
- Employé sans bureaux? **NON**
- Bureau sans employé? **OUI**

```
@OneToOne(optional = false)  
private Bureau bureau;
```

→ Ajout de contrainte **unique** et **not null** sur la clé étrangère

### → Composition

Les entités possèdent un cycle de vie propre, mais l'entité employé et bureau ont une relation de type parent-enfant (un employé dépend d'un bureau). Le cycle de vie de l'employé est lié au cycle de vie du bureau

## @OneToOne



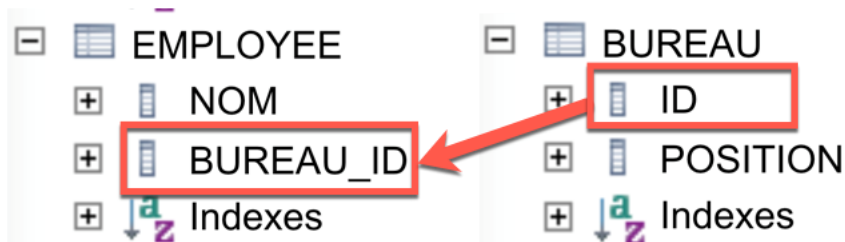
*Solution possible: partage de clé primaire*

@OneToOne

@MapsId

private Bureau bureau;

→ Partage de la clé primaire de la table bureau en tant que clé primaire de la table employe



### → Composition

Les entités possèdent un cycle de vie propre, mais l'entité employé et bureau ont une relation de type parent-enfant (un employé dépend d'un bureau). Le cycle de vie de l'employé est lié au cycle de vie du bureau

# @OneToMany



## OneToMany

Définit une relation 1 – n entre deux entités



```
@Entity
public class Departement {
```

```
    @Id
    private Long id;
```

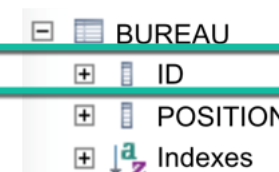
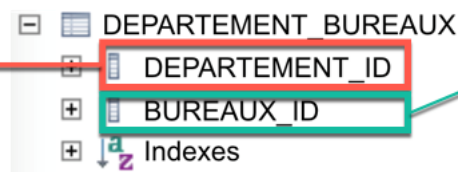
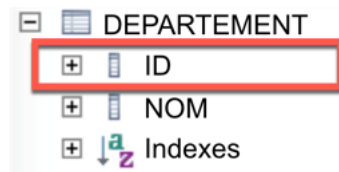
```
    private String nom;
```

```
    @OneToMany
    List<Bureau> bureaux = new ArrayList<>();
```

```
@Entity
public class Bureau {
```

```
    @Id
    private Long id;
```

```
    private String position;
```

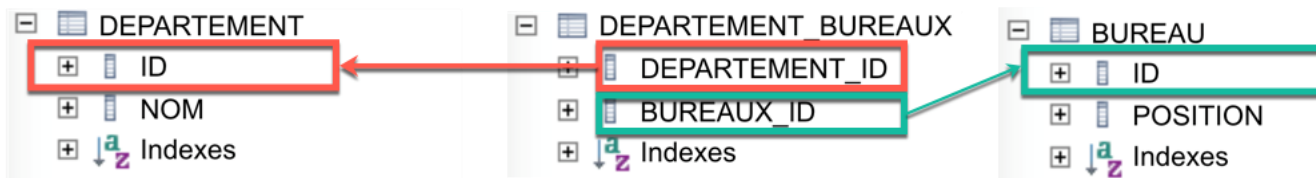


!?!?

# @OneToMany



Résultat en base de données (implémentation):



- La relation générée par JPA est de type n-m!
- C'est le comportement par défaut de *OneToMany*
- Relation « standard »:

@OneToMany

**@JoinColumn(name = "dep\_id")**

```
List<Bureau> bureaux = new ArrayList<>();
```



# @ManyToOne



## ManyToOne

Définit une relation  $n - 1$  entre deux entités



```
@Entity
public class Departement {
    @Id
    private Long id;
    private String nom;
}
```



```
@Entity
public class Bureau {
    @Id
    private Long id;
    private String position;
    @ManyToOne
    private Departement departement;
}
```

DEPARTEMENT

- ID
- NOM
- Indexes

BUREAU

- ID
- POSITION
- DEPARTEMENT\_ID
- Indexes



# @ManyToMany



## ManyToOne

Définit une relation  $n - m$  entre deux entités



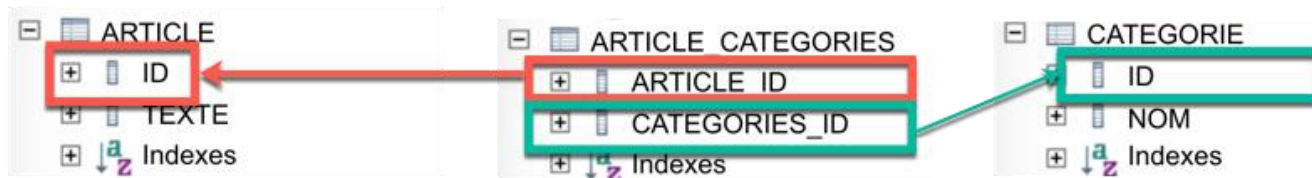
```

@Entity
public class Article {
    @Id
    private Long id;
    private String texte;
    @ManyToMany
    private List<Catégorie> categories = new ArrayList<>();
}
  
```



```

@Entity
public class Catégorie {
    @Id
    private Long id;
    private String nom;
    @ManyToMany
    private List<Article> articles = new ArrayList<Article>();
}
  
```



## JPA – à étudier



- Types de relations
  - **Bidirectionnelle**
  - *Lazy vs Eager*
- Paramétrage globale
  - **Clé primaire, composée**