# LA SPÉCIFICATION JPA – ASSOCIATION
# SPRING DATA JPA
# SPRING BOOT

2021-2022

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)

# Séance 5



- Séquence 5: Le mapping des différentes associations
  - Charge : 3 heures


    - Présentation des différentes stratégies d'héritage.
    - Le mapping des différentes associations entre les entités (unidirectionnelle et bidirectionnelle)
      - One to one
      - One to many/many to one
      - Many to many

# Séance 5



- Plan
  - Relations dans la base de données relationnelles
  - Relations entre les entités
  - Association 1 :1 unidirectionnelle
  - Association 1 :1 bidirectionnelle

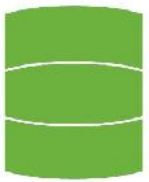
  One to One (1:1): Uni-Bi
  - Association 1 : N unidirectionnelle

  One to Many (1:N): Uni
  - Association N : 1 unidirectionnelle
  - Association N : 1 bidirectionnelle

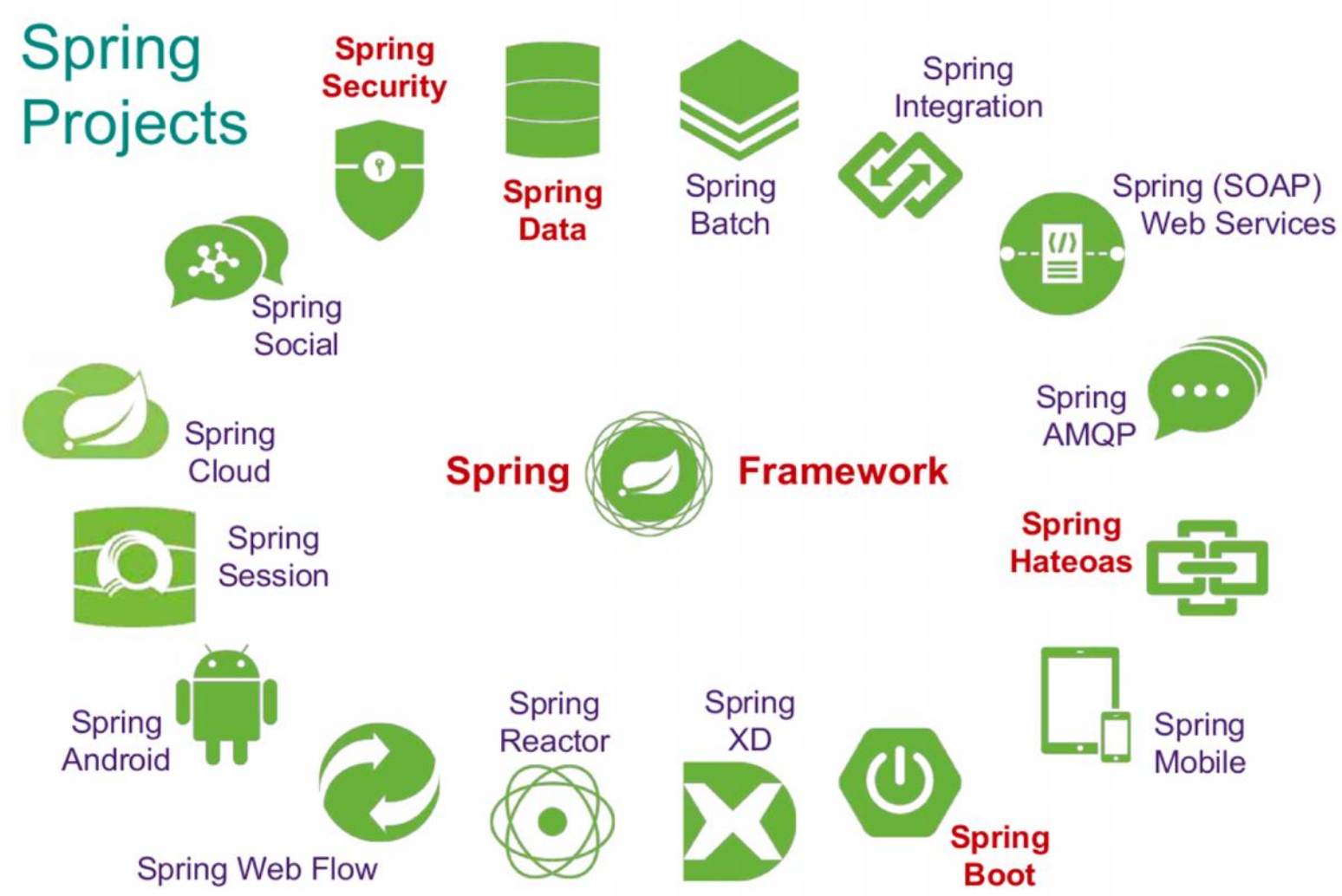
  Many to One (N:1): Uni-Bi
  - Association M : N unidirectionnelle
  - Association M : N bidirectionnelle

  Many to Many (N:M): Uni-Bi
  - Les associations réflexives
  - Association M:N porteuses de données

# Spring Data



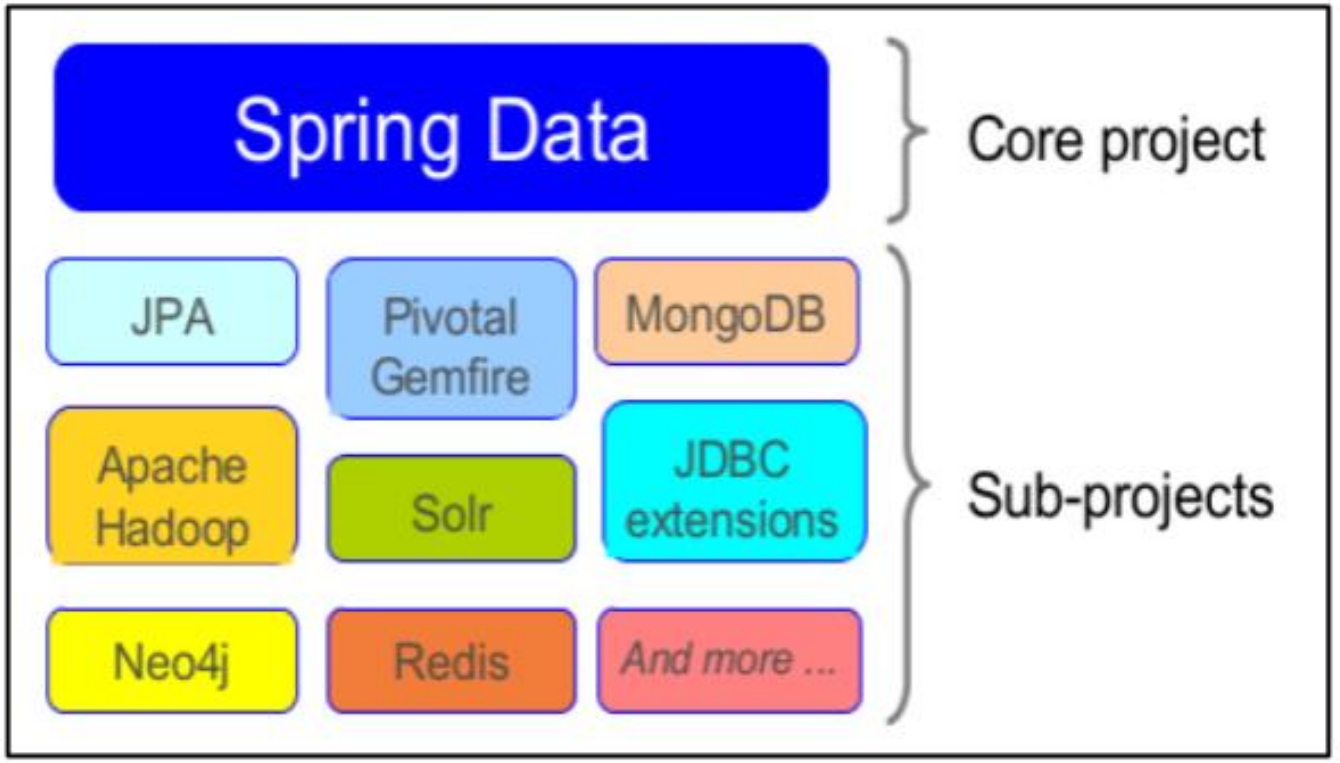



Spring Projects
Spring Security
Spring Data
Spring Batch
Spring Integration
Spring (SOAP) Web Services
Spring Social
Spring Cloud
Spring AMQP
Spring Framework
Spring Session
Spring Hateoas
Spring Android
Spring Reactor
Spring XD
Spring Mobile
Spring Web Flow
Spring Boot

# Spring Data



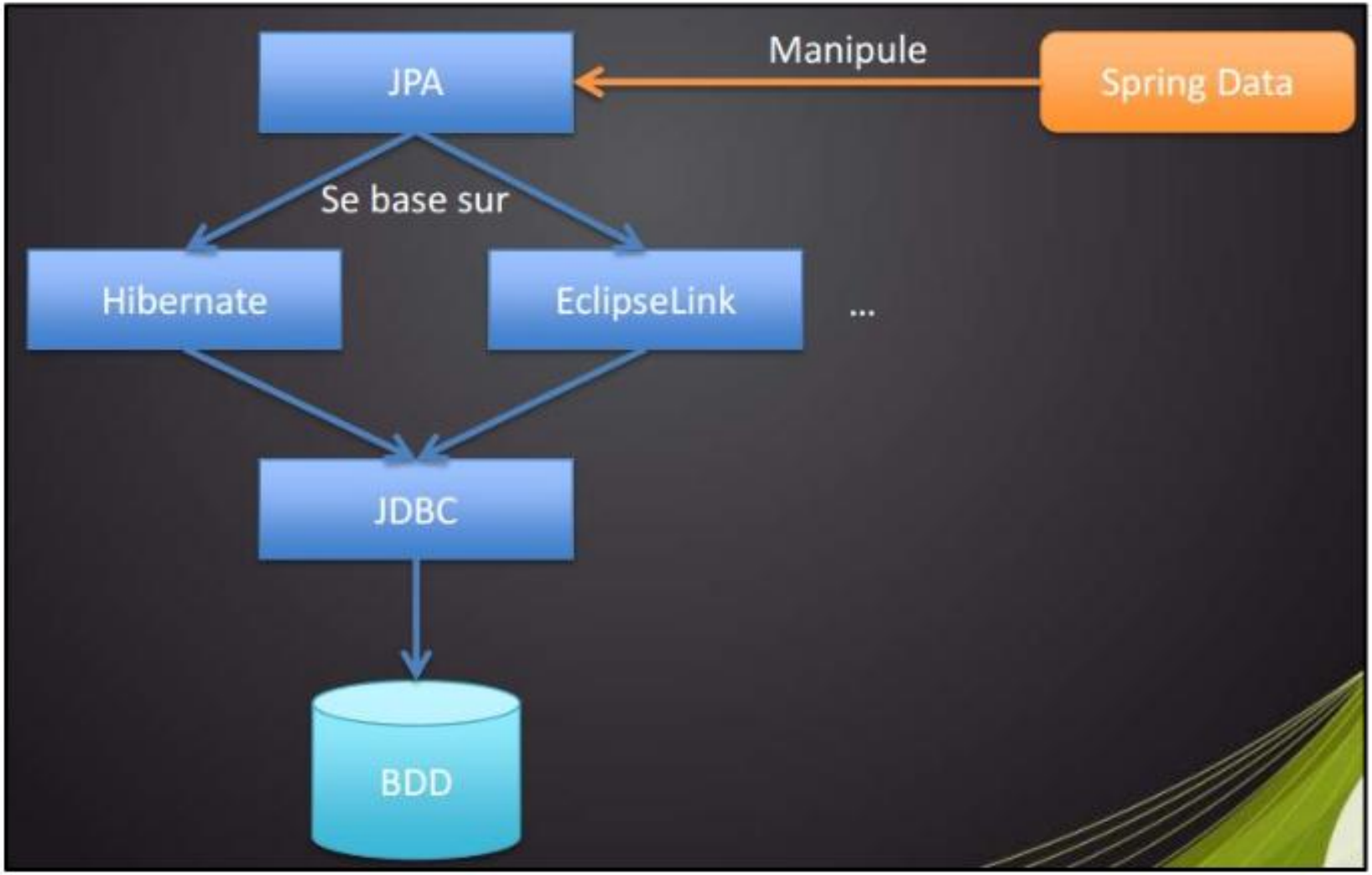
- C'est un module Spring qui a pour but de:
  - Faciliter l'écriture des couches d'accès aux données.
  - Offrir une abstraction commune pour l'accès aux données quelle que soit la source de données (SQL ou NoSQL).



# Spring Data JPA

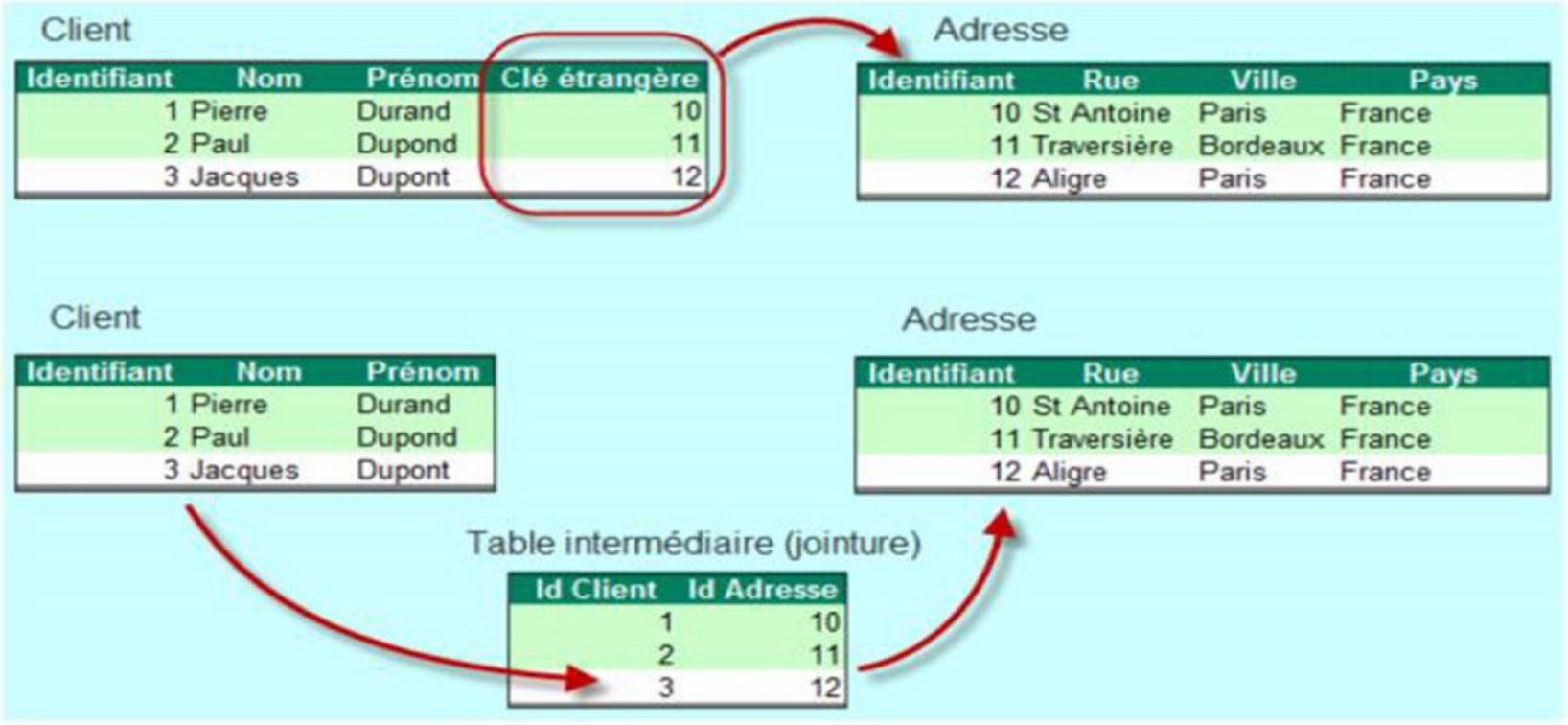


- **Spring Data JPA** est un sous projet du projet **Spring Data.**



# Relations dans la base de données relationnelle






- La cardinalité indique le nombre d'instances de classe étant en relation avec la classe située à l'autre extrémité de l'association.
  - Exemple: On trouve la cardinalité 1:1, i.e. un enregistrement d'une table correspond à un enregistrement d'une autre table.
  - On trouve, aussi, la cardinalité n:n, i.e. n enregistrements sont associés à n autres enregistrements.

# Relations entre les entités



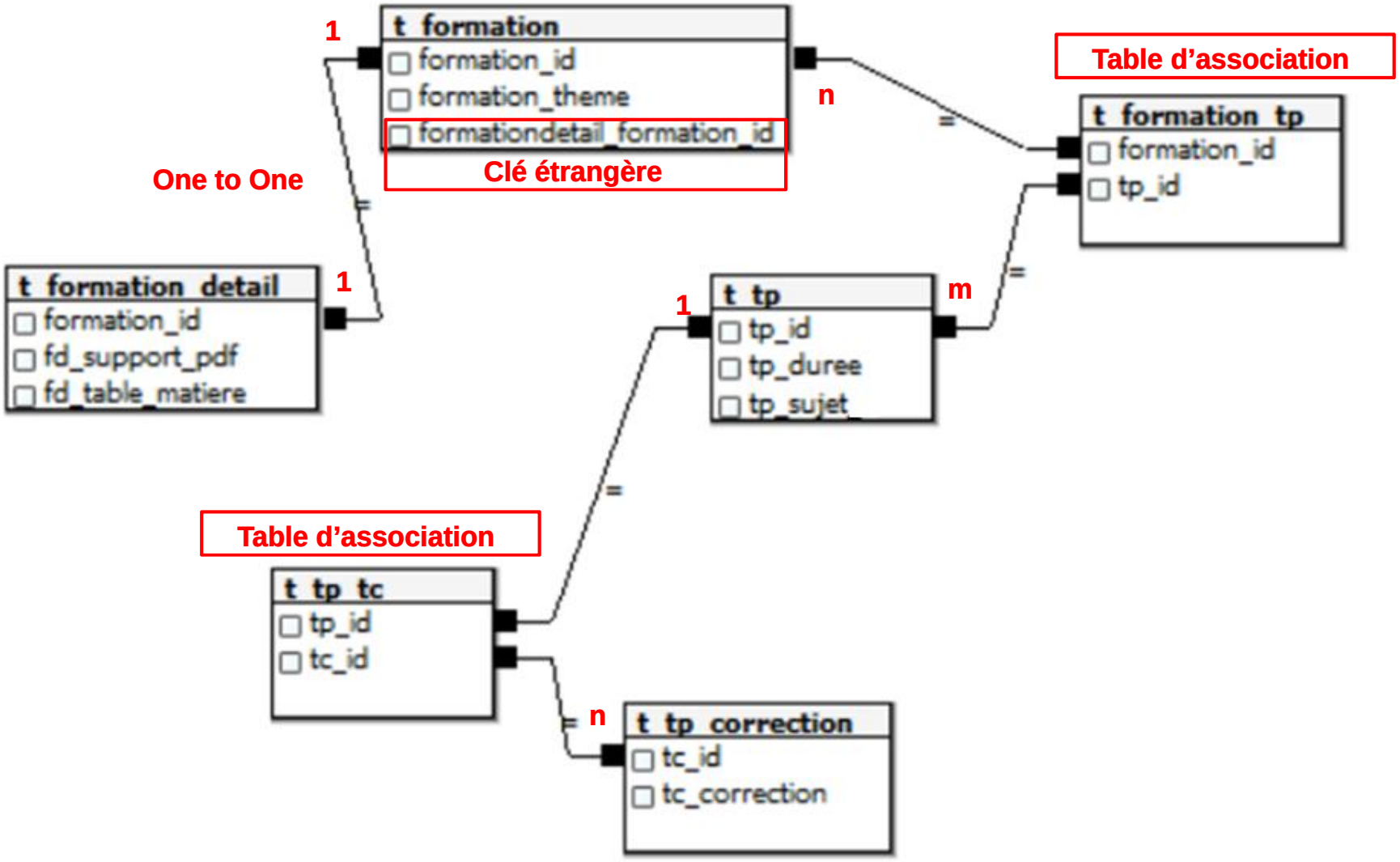
- Ces cardinalités sont classiques dans les schémas de base de données et correspondent à des annotations JPA.
- La plupart des entités doivent pouvoir référencer ou être en relation avec d'autres entités.
- Les types d'associations entre deux entités sont :
  - 1-1 : @OneToOne,
  - 1-N : @OneToMany,
  - N-1 : @ManyToOne,
  - N-M : @ManyToMany.

**Cardinalité : Annotation JPA**

- Chaque type d'association peut être :
  - Unidirectionnelle
  - Bidirectionnelle

# Trouver et expliquer les associations





# Trouver et expliquer les associations

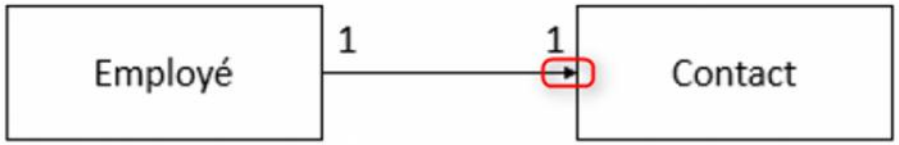


- **One To One** entre T_FORMATION et T_FORMATION_DETAIL (Clé étrangère formationdetail_formation_id).
  - La Formation a un ensemble de détails (une seule ligne dans la table T_FORMATION_DETAIL). Le détail d'une formation est liée à une seule Formation.
- **Many To Many** : T_FORMATION et T_TP (Table d'association T_FORMATION_TP).
  - La Formation a plusieurs TPs (TP1, TP2, …). Un même TP peut être lié à plusieurs formations (Maven, JPA, EJB, JSF).
- **One To Many**: T_TP et T_TP_CORRECTION (Table d'association T_TP_TC).
  - Le TP peut avoir plusieurs Corrections (Avec Maven / Sans Maven, Avec Hibernate / Avec EclipseLink, …). Une Correction n'est liée qu'à un TP.
  - Si l'association Many To One était utilisée entre T_TP et T_TP_CORRECTION, à la place de One To Many, quel changement dans le diagramme ci-dessus?

# Association 1 :1 unidirectionnelle

# Association 1 :1 unidirectionnelle






```
@Entity
public class Employe implements Serializable {


    private Contact contact;
    @OneToOne
    public Contact getContact() {
        return contact;
    }

    public void setContact(Contact contact) {
        this.contact = contact;
    }
```

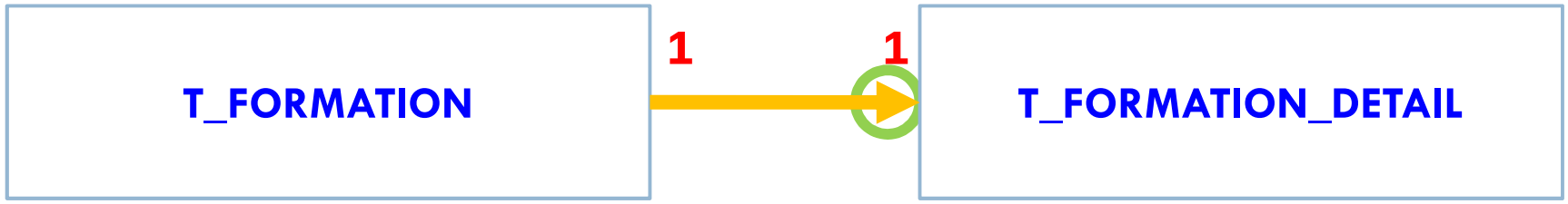

- L'employé possède un «Contact», alors que le «Contact» n'a aucune information sur «l'employé» auquel il est associé.
- On peut changer le nom de la colonne de jointure en utilisant l'annotation @JoinColumn.

# Association 1 :1 unidirectionnelle

## One To One Unidirectionnelle



- **One To One:** La Formation a un ensemble de détails (une seule ligne dans la table T_FORMATION_DETAIL). Le détail d'une formation est liée à une seule Formation.
- **Unidirectionnelle:** La Formation connait le détail (contient un attribut de type Formation_Detail), alors que le détail n'a aucune information sur la Formation auquel il est associé.



# Association 1 :1 unidirectionnelle

## One To One Unidirectionnelle



```
@Entity
@Table(name = "T_FORMATION")
public class Formation implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FORMATION_ID")
private Long id; // Identifiant formation (Clé primaire)

@Column(name="FORMATION_THEME")
private String theme; // Thème formation

@OneToOne
@JoinColumn(name="FK_FD_ID")
private FormationDetail formationDetail;

// Constructeurs, getters, setters
}
```

# Association 1 :1 unidirectionnelle

## One To One Unidirectionnelle

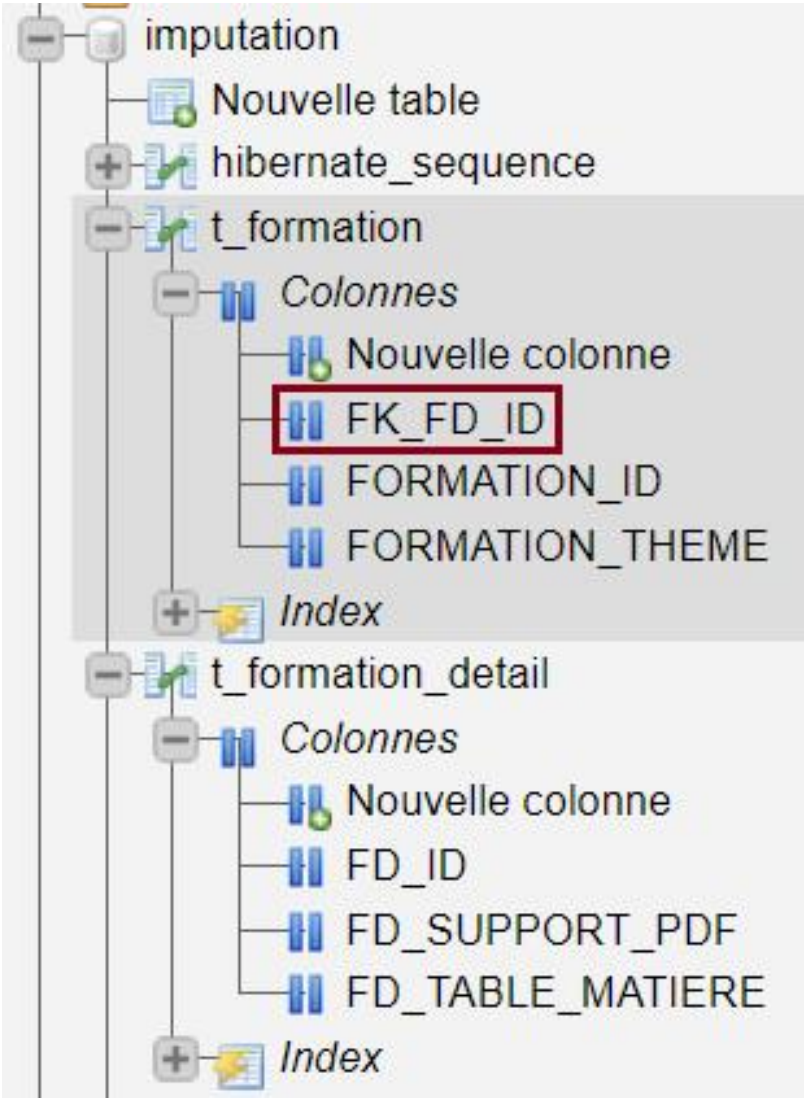


```
@Entity
@Table(name="T_FORMATION_DETAIL")
public class FormationDetail implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FD_ID")
private Long fdId;

@Column(name="FD_TABLE_MATIERE")
private String fdTableMatiere;

@Column(name="FD_SUPPORT_PDF")
private String fdSupportPDF;

// Constructeurs, getters, setters
}
```

# Association 1 :1 unidirectionnelle

## One To One Unidirectionnelle



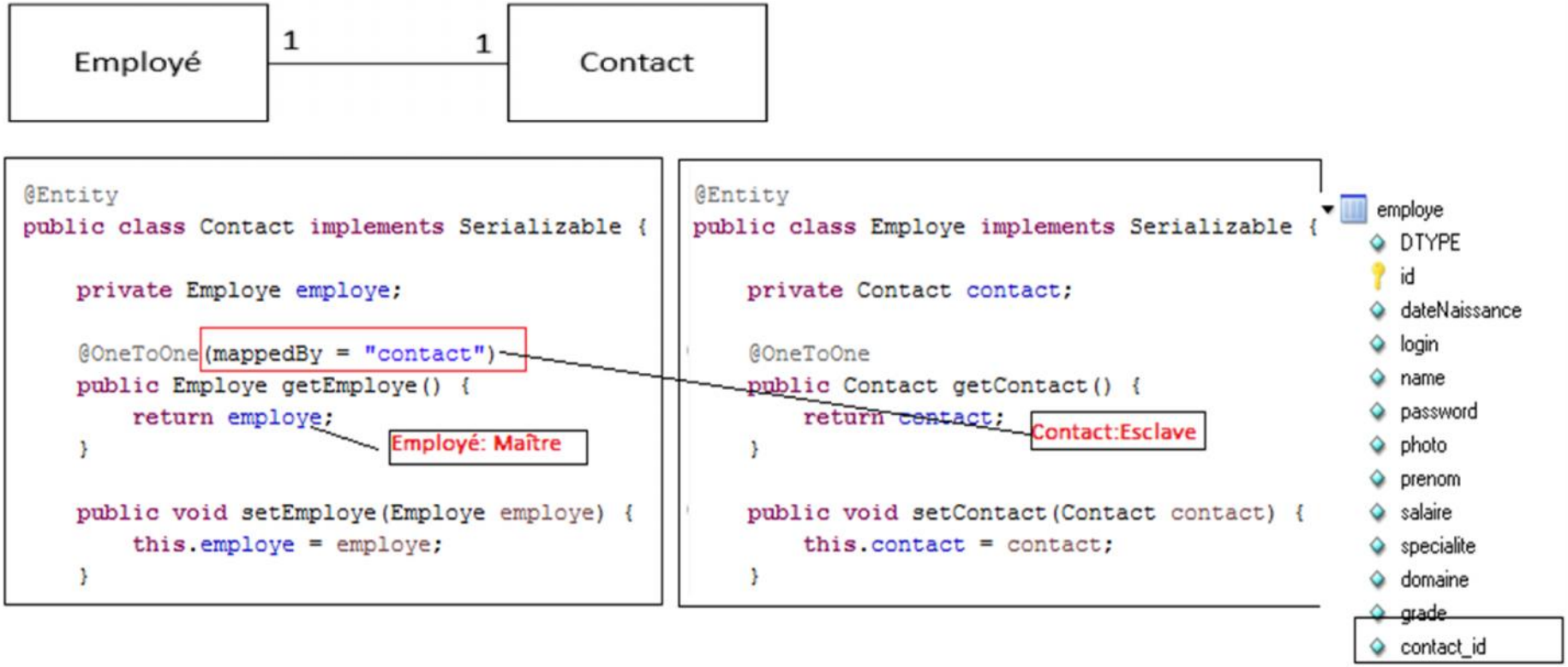


# Association 1 :1 bidirectionnelle

# Association 1 :1 bidirectionnelle

## One To One Bidirectionnelle





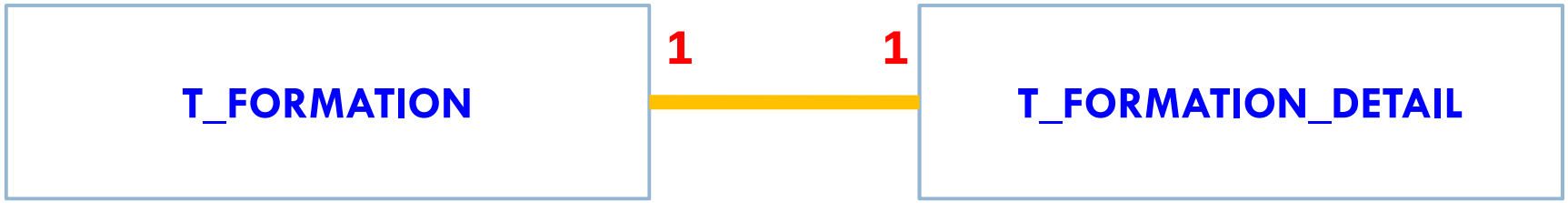

- C'est l'attribut «**mappedBy**» qui crée le caractère bidirectionnel de la relation et qui permet de définir les deux bouts de l'association « le maître et l'esclave », autrement, le sens de migration des clés étrangères.
- La classe « Employé » contient un objet de type « Contact » et le « Contact » contient un objet de type « Employé ».

# Association 1 :1 bidirectionnelle

## One To One Bidirectionnelle



- **One To One** : La Formation a un ensemble de détails (une seule ligne dans la table T_FORMATION_DETAIL). Le détail d'une formation est liée à une seule Formation.
- **Bidirectionnelle** : l'Entité «Formation» contient un attribut de type «Formation_Detail» et l'Entité «Formation_Detail» contient un attribut de type «Formation».
- C'est l'attribut «**mappedBy**» qui crée le caractère bidirectionnel de la relation et qui permet de définir les deux bouts de l'association «Parent / Child».
- Au niveau des Entités Java, c'est le fils qui contient l'attribut «**mappedBy**».
- En base de données, c'est le Parent qui contiendra la Clé étrangère qui pointera vers le Child.



# Association 1 :1 bidirectionnelle

## One To One Bidirectionnelle



```
@Entity
@Table(name = "T_FORMATION")
public class Formation implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FORMATION_ID")
private Long id; // Identifiant formation (Clé primaire)

@Column(name="FORMATION_THEME")
private String theme; // Thème formation

@OneToOne
@JoinColumn(name="FK_FD_ID")
private FormationDetail formationDetail;

// Constructeurs, getters, setters
}
```

# Association 1 :1 bidirectionnelle

## One To One Bidirectionnelle

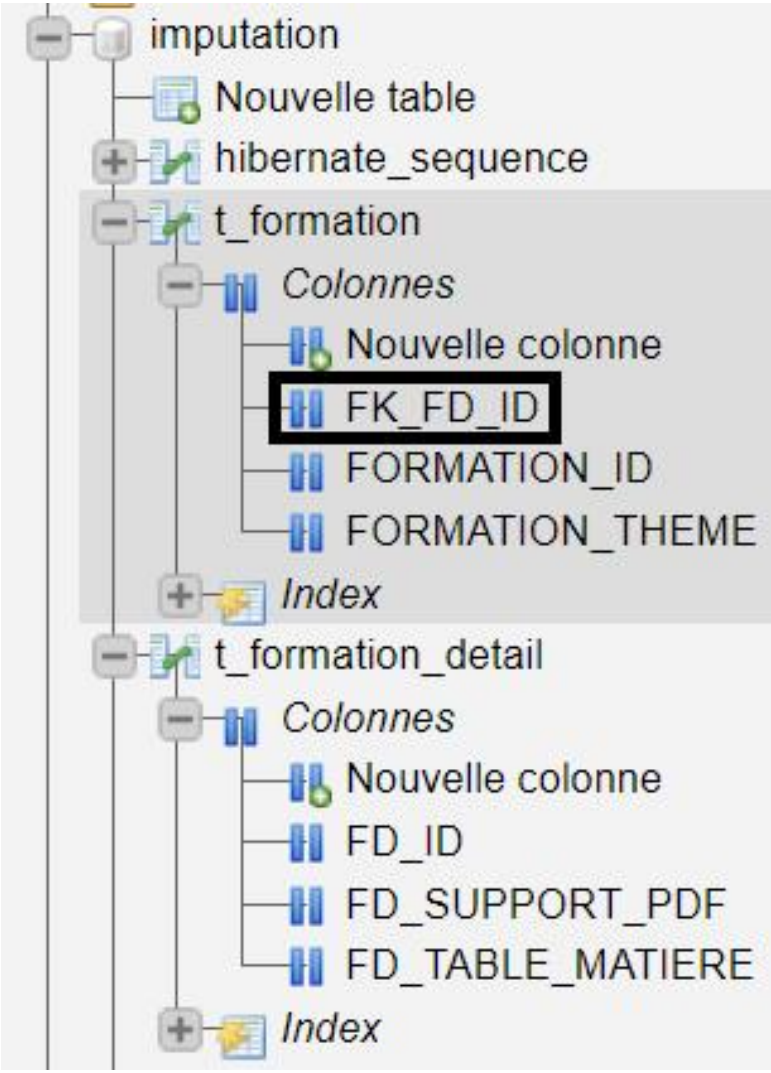


```
@Entity
@Table(name="T_FORMATION_DETAIL")
public class FormationDetail implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FD_ID")
private Long fdId;

@Column(name="FD_TABLE_MATIERE")
private String fdTableMatiere;

@Column(name="FD_SUPPORT_PDF")
private String fdSupportPDF;

@OneToOne(mappedBy="formationDetail")
private Formation formation;

// Constructeurs, getters, setters
}
```

# Association 1 :1 bidirectionnelle

## One To One Bidirectionnelle



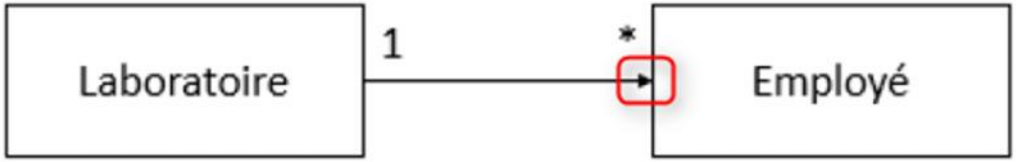


# Association 1 : N unidirectionnelle

# Association 1 :N Unidirectionnelle

## One To Many Unidirectionnelle






```
public class Laboratoire implements Serializable {

    private List<Employe> employes;
    @OneToMany
    public List<Employe> getEmployes() {
        return employes;
    }
    public void setEmployes(List<Employe> employes) {
        this.employes = employes;
    }
```

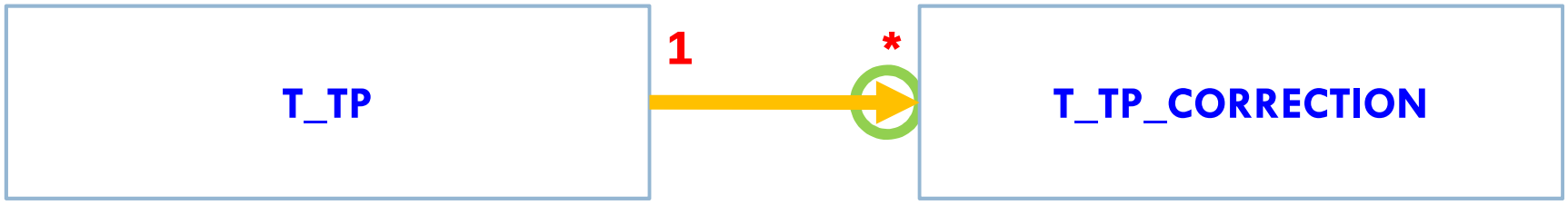

- Le laboratoire possède une collection d'employés alors que « l'employé » n'a aucune information sur « le laboratoire » auquel il appartient.

# Association 1 :N Unidirectionnelle

## One To Many Unidirectionnelle



- **One To Many :** Le TP peut avoir plusieurs Corrections (Avec Maven / Sans Maven, Avec Hibernate / Avec EclipseLink, …). Une Correction n'est liée qu'à un TP.
- **Unidirectionnelle TP → Correction :** Le TP connait les corrections, alors que la Correction n'a aucune information sur le TP dont elle est la Solution.



# Association 1 :N Unidirectionnelle

## One To Many Unidirectionnelle



```
@Entity
@Table(name = "T_TP")
public class TravauxPratiques implements Serializable {

private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="TP_ID")
Long tpId;

@Column(name="TP_SUJET")
String tpSujet;
@Column(name="TP_DUREE")
Long tpDuree;
@OneToMany
@JoinTable(name="T_TP_TC", joinColumns={@JoinColumn(name="TP_ID")},
   inverseJoinColumns={@JoinColumn(name="TC_ID")})
private Set<TpCorrection> TpCorrections;

// Constructeurs, getters, setters
}
```

# Association 1 :N Unidirectionnelle

## One To Many Unidirectionnelle

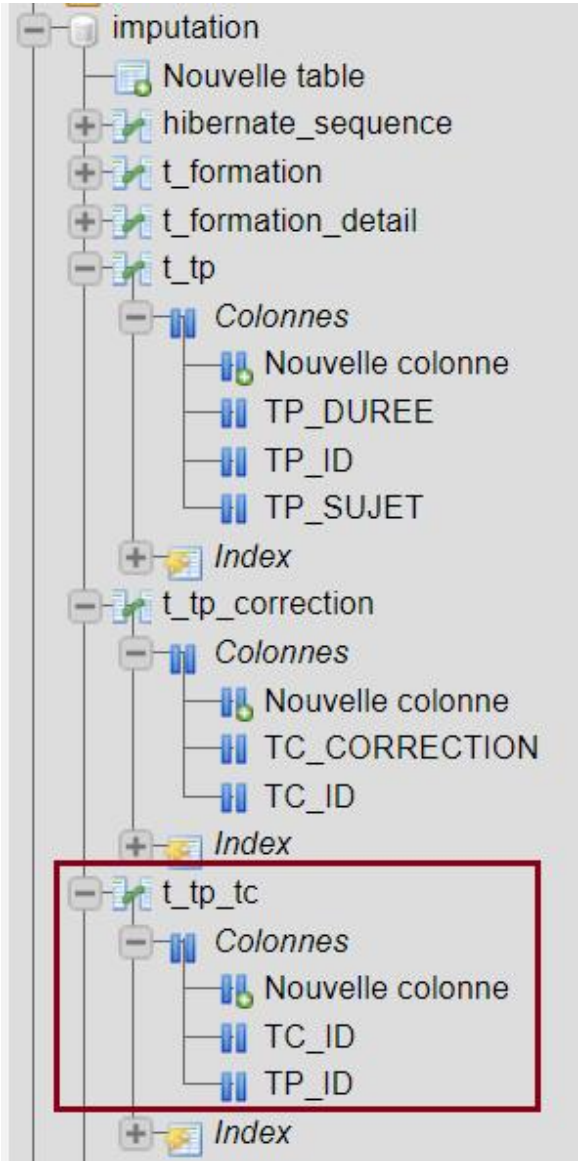


```
@Entity
@Table(name="T_TP_CORRECTION")
public class TpCorrection implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="TC_ID")
private Long tcId;

@Column(name="TC_CORRECTION")
private String tcCorrection;

// Constructeurs, getters, setters
}
```

# Association 1 :N Unidirectionnelle

## One To Many Unidirectionnelle



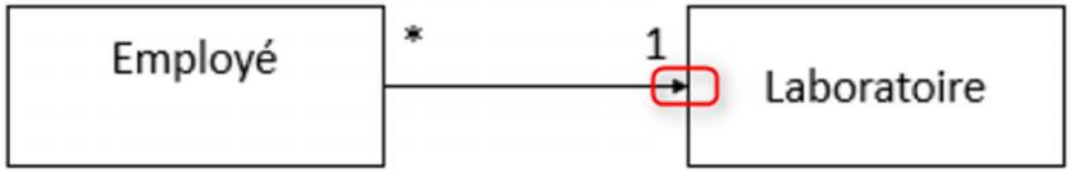


# Association N : 1 unidirectionnelle

# Association N : 1 unidirectionnelle

## Many To One Unidirectionnelle






```
@Entity
public class Employe implements Serializable {

    private Laboratoire laboratoire;

    @ManyToOne
    public Laboratoire getLaboratoire() {
        return laboratoire;
    }

    public void setLaboratoire(Laboratoire laboratoire) {
        this.laboratoire = laboratoire;
    }
```

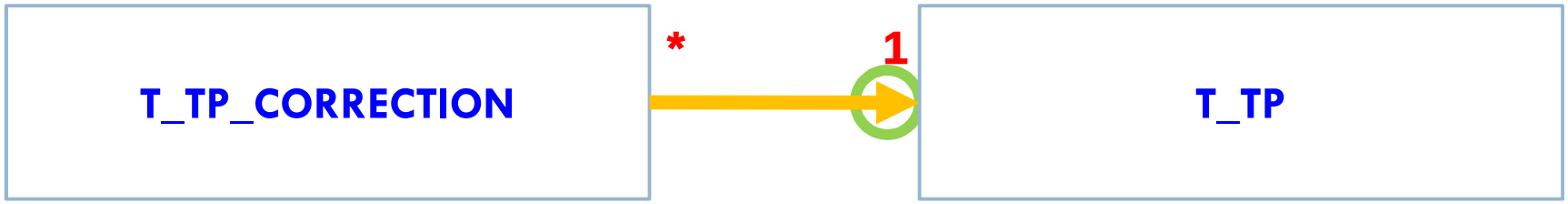

- L'employé possède un laboratoire alors que « le laboratoire » n'a aucune information sur « les employés » dont ils font partie.

# Association N : 1 unidirectionnelle

## Many To One Unidirectionnelle



- **Many To One :** Un TP peut avoir plusieurs Corrections. Chaque Correction est liée à un unique TP.
- **Unidirectionnelle :** Chaque Correction de TP a l'information concernant le TP (attribut TravauxPratiques dans l'Entité TP_Correction, Clé étrangère dans la table T_TP_CORRECTION), alors que le TP n'a aucune information sur ses «Corrections».



# Association N : 1 unidirectionnelle

## Many To One Unidirectionnelle



```
@Entity
@Table(name="T_TP_CORRECTION")
public class TpCorrection implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="TC_ID")
private Long tcId;

@Column(name="TC_CORRECTION")
private String tcCorrection;

@ManyToOne
@JoinColumn(name="FK_TP_ID")
TravauxPratiques travauxPratiques;

// Constructeurs, getters, setters
}
```

# Association N : 1 unidirectionnelle

## Many To One Unidirectionnelle

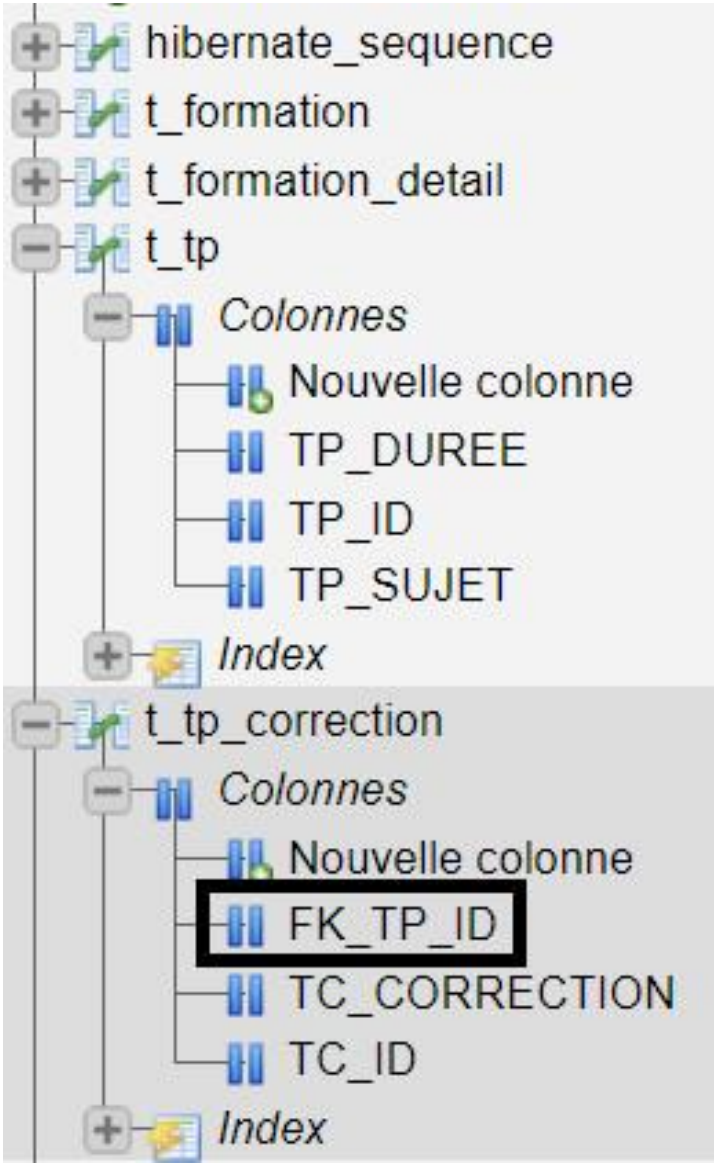


```
@Entity
@Table(name = "T_TP")
public class TravauxPratiques implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="TP_ID")
Long tpId;

@Column(name="TP_SUJET")
String tpSujet;

@Column(name="TP_DUREE")
Long tpDuree;

// Constructeurs, getters, setters
}
```

# Association N : 1 unidirectionnelle

## Many To One Unidirectionnelle





# Association N : 1  bidirectionnelle

# Association N : 1 bidirectionnelle

## Many To One Bidirectionnelle




**Slave**

```
@Entity
public class Laboratoire implements Serializable {

    private List<Employe> employes;

    @OneToMany(mappedBy="laboratoire")
    public List<Employe> getEmployes() {
        return employes;
    }

    public void setEmployes(List<Employe> employes) {
        this.employes = employes;
    }
```

**Master**

```
@Entity
public class Employe implements Serializable {
    private Laboratoire laboratoire;

    @ManyToOne
    public Laboratoire getLaboratoire() {
        return laboratoire;
    }

    public void setLaboratoire(Laboratoire laboratoire)
        this.laboratoire = laboratoire;
    }
```


- L'attribut mappedBy est défini pour l'annotation @OneToMany, mais pas pour l'annotation @ManyToOne.

# Association N : 1 bidirectionnelle

## Many To One Bidirectionnelle



- **Many To One :** Un TP peut avoir plusieurs Correction. Une Correction est liée à un seul TP.
- **Bidirectionnelle :** Le TP connait ses Corrections. Chaque Correction connait elle aussi le TP dont elle est la Solution.


- L'attribut **mappedBy** est défini pour l'annotation @OneToMany, mais pas pour l'annotation @ManyToOne, car elle est toujours liée au «Child».

# Association N : 1 bidirectionnelle

## Many To One Bidirectionnelle



```
@Entity
@Table(name="T_TP_CORRECTION")
public class TpCorrection implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="TC_ID")
private Long tcId;

@Column(name="TC_CORRECTION")
private String tcCorrection;

@ManyToOne
@JoinColumn(name="FK_TP_ID")
TravauxPratiques travauxPratiques;

// Constructeurs, getters, setters
}
```

# Association N : 1 bidirectionnelle

## Many To One Bidirectionnelle

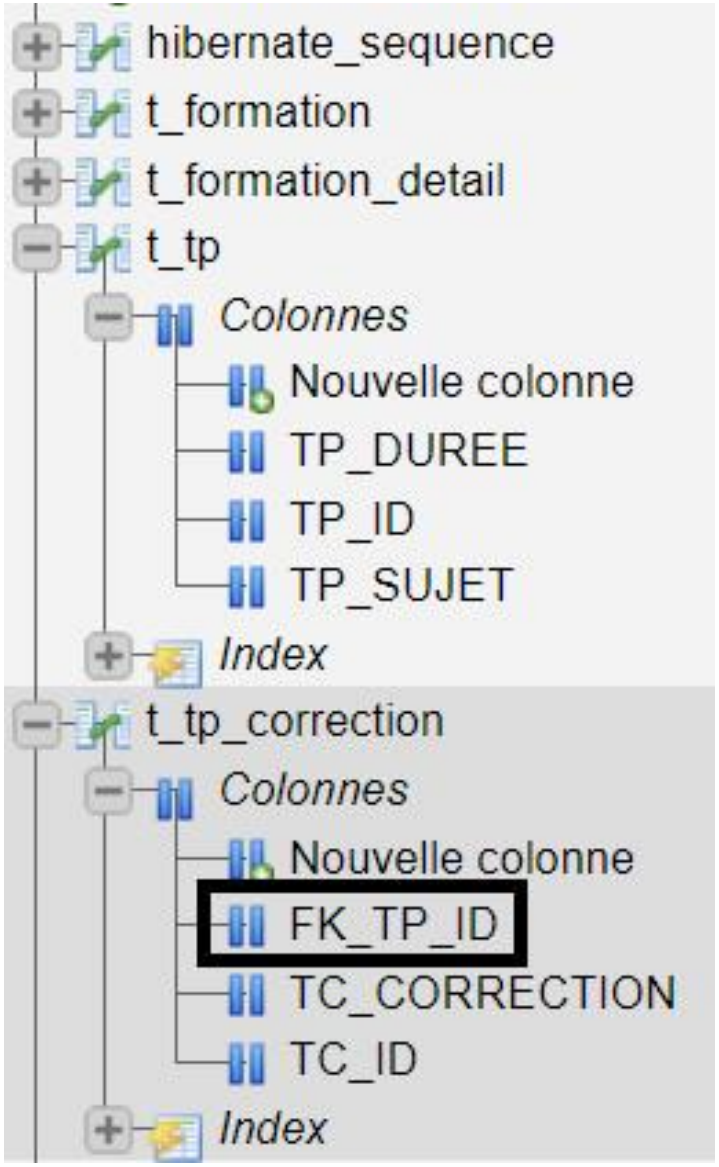


```
@Entity
@Table(name = "T_TP")
public class TravauxPratiques implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="TP_ID")
Long tpId;

@Column(name="TP_SUJET")
String tpSujet;

@Column(name="TP_DUREE")
Long tpDuree;

@OneToMany(mappedBy="travauxPratiques")
private Set<TpCorrection> TpCorrections;

// Constructeurs, getters, setters
}
```

# Association N : 1  bidirectionnelle

**Many To One Bidirectionnelle**

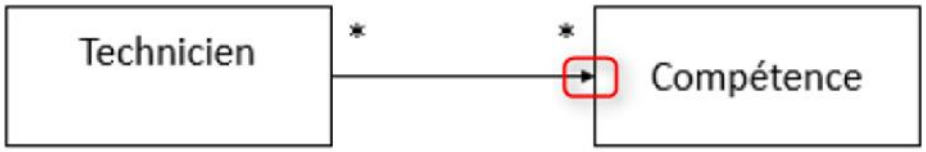




# Association M : N unidirectionnelle

# Association M : N unidirectionnelle





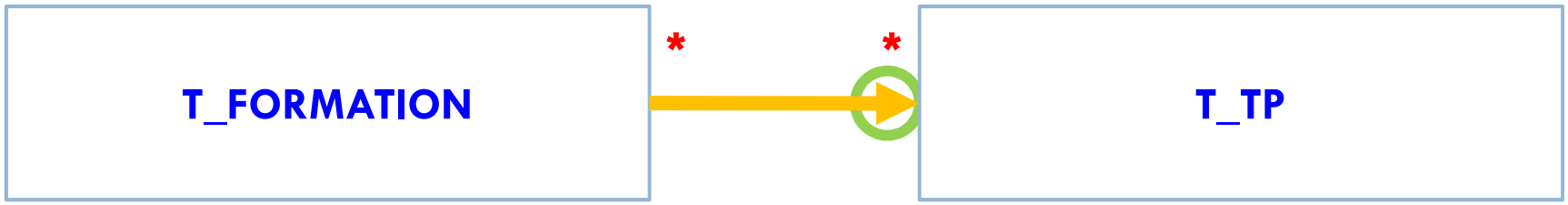

```
public class Technicien extends Employe implements Serializable {

    private List<Competence> competences;

    @ManyToMany

    public List<Competence> getCompetences() {
        return competences;
    }
    public void setCompetences(List<Competence> competences) {
        this.competences = competences;
    }
```


- Le technicien possède une «collection de compétences» alors que «la compétence» n'a aucune information sur «les techniciens» auquel elle est associée.

# Many To Many Unidirectionnelle



- **Many To Many** : La Formation a plusieurs TPs (TP1, TP2, …). Un même TP peut être lié à plusieurs formations (Maven, JPA, EJB, JSF).
- **Unidirectionnelle** : La formation a plusieurs TPs et les connait. Mais, le TP n'a aucune information sur «les Formations» auxquelles il est associé.



# Many To Many Unidirectionnelle



```
@Entity
@Table(name = "T_FORMATION")
public class Formation implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FORMATION_ID")
private Long id; // Identifiant formation (Clé primaire)

@Column(name="FORMATION_THEME")
private String theme; // Thème formation

@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name="FK_FD_ID")
private FormationDetail formationDetail;

@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "T_FORMATION_TP",
joinColumns={@JoinColumn(name="FORMATION_ID")},
inverseJoinColumns={@JoinColumn(name ="TP_ID")})
private Set<TravauxPratiques> formationTps;

// Constructeurs, getters, setters
```

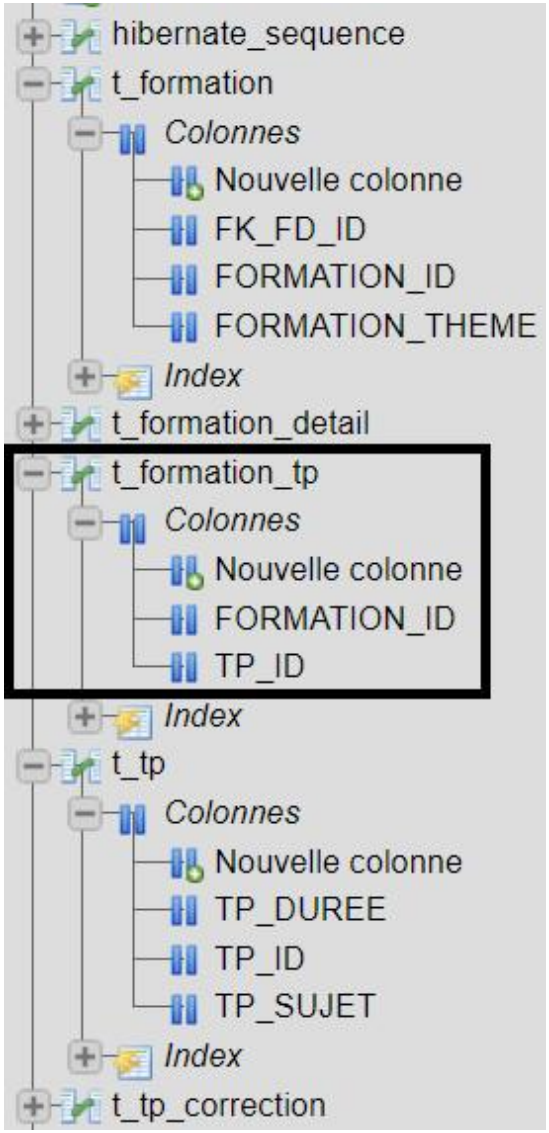
# Many To Many Unidirectionnelle



```
@Entity
@Table(name = "T_TP")
public class TravauxPratiques implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="TP_ID")
Long tpId;

@Column(name="TP_SUJET")
String tpSujet;

@Column(name="TP_DUREE")
Long tpDuree;

// Constructeurs, getters, setters
```

# Many To Many Unidirectionnelle





# Association M : N bidirectionnelle

# Association M : N bidirectionnelle

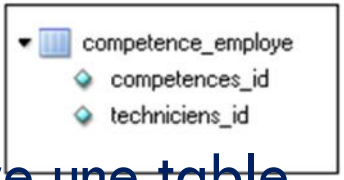



```java
@Entity
public class Competence implements Serializable {

    private List<Technicien> techniciens;

    @ManyToMany
    public List<Technicien> getTechniciens() {
        return techniciens;
    }

    public void setTechniciens(List<Technicien> techniciens) {
        this.techniciens = techniciens;
    }
```

technicien: Slave

```java
@Entity

public class Technicien extends Employe implements Serializable {

    private List<Competence> competences;

    @ManyToMany (mappedBy="techniciens")

    public List<Competence> getCompetences() {
        return competences;
    }
    public void setCompetences(List<Competence> competences) {
        this.competences = competences;
    }
```

Competence : master




- Le Mapping de cette relation dans la base de données génère une table associative qui contient les deux clés étrangères des deux entités.

# Many To Many Bidirectionnelle



- **Many To Many :** La Formation a plusieurs TPs (TP1, TP2, …). Un même TP peut être lié à plusieurs formations (Maven, JPA, EJB, JSF).
- **Bidirectionnelle :** La formation a plusieurs TPs et les connait. Chaque TP est associé à plusieurs Formations, et les connait aussi.

# Many To Many Bidirectionnelle



```
@Entity
@Table(name = "T_FORMATION")
public class Formation implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name="FORMATION_ID")
private Long id; // Identifiant formation (Clé primaire)

@Column(name="FORMATION_THEME")
private String theme; // Thème formation

@OneToOne(cascade = CascadeType.ALL)
@JoinColumn(name="FK_FD_ID")
private FormationDetail formationDetail;

@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "T_FORMATION_TP",
joinColumns={@JoinColumn(name="FORMATION_ID")},
inverseJoinColumns={@JoinColumn(name ="TP_ID")})
private Set<TravauxPratiques> formationTps;

// Constructeurs, getters, setters
```

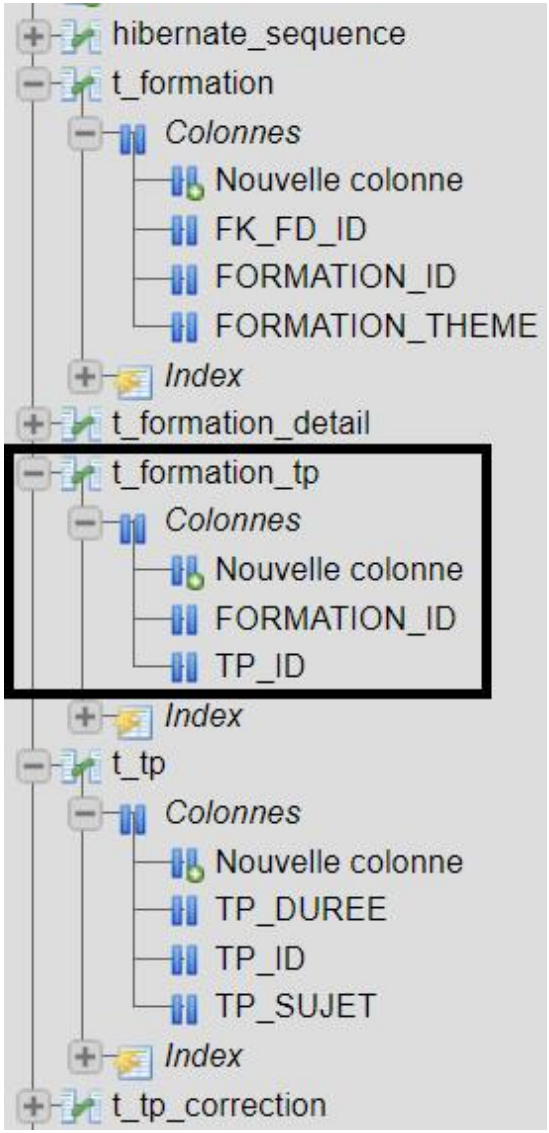
# Many To Many Bidirectionnelle



```
@Entity
@Table(name = "T_TP")
public class TravauxPratiques implements Serializable {

private static final long serialVersionUID = 1L;

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name="TP_ID")
Long tpId;

@Column(name="TP_SUJET")
String tpSujet;

@Column(name="TP_DUREE")
Long tpDuree;

@ManyToMany(mappedBy="formationTps", cascade = CascadeType.ALL)
private Set<Formation> formations;

// Constructeurs, getters, setters
}
```

# Many To Many Bidirectionnelle



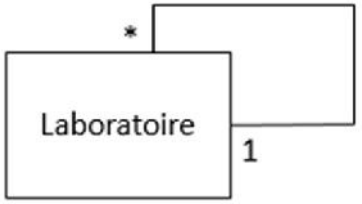


# Les associations réflexives

# Les associations réflexives



```java
@Entity
public class Laboratoire implements Serializable {

    private Laboratoire labo;

    private List<Laboratoire> miniLabos;

    @ManyToOne
    @JoinColumn(name="labo_fk")
    public Laboratoire getLabo() {
        return labo;
    }

    public void setLabo(Laboratoire labo) {
        this.labo = labo;
    }

    @OneToMany(mappedBy = "labo", fetch = FetchType.EAGER)

    public List<Laboratoire> getMiniLabos() {
        return miniLabos;
    }

    public void setMiniLabos(List<Laboratoire> miniLabos) {
        this.miniLabos = miniLabos;
    }
```


- Une association réflexive est une association qui relie des occurrences de la même entité, elle peut être de type 1:1, 1:N, N:1, M:N.

# Association M:N porteuses de données

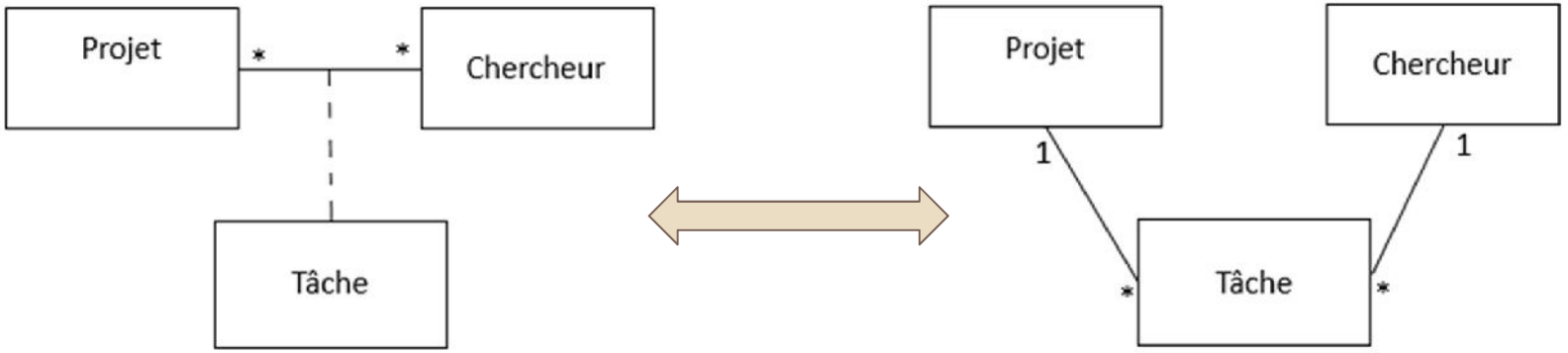
# Association M:N porteuses de données



- Une association **ManyToMany** est dite porteuse de données si la classe associative comporte des données autres que sa clé primaire.

**Exemple :**

Une tâche est caractérisée par un projet, un chercheur et un nom.



# Association M:N porteuses de données



## Définition de la clé primaire

```java
@Embeddable
public class TachePk implements Serializable {

    private int idChercheur;
    private int idProjet;
    private String nom;
```

```java
@Entity
public class Tache implements Serializable {

    private TachePk tachePk;
    private int duree;

    @EmbeddedId
    public TachePk getTachePk() {
        return tachePk;
    }
```

# Association M:N porteuses de données



```java
@Entity
public class Tache implements Serializable {

    private Projet projet;
    private Chercheur chercheur;

    @ManyToOne
    @JoinColumn(name="idProjet",referencedColumnName="id"
    ,insertable=false,updatable=false)
    public Projet getProjet() {
        return projet;
    }

    public void setProjet(Projet projet) {
        this.projet = projet;
    }

    @ManyToOne
    @JoinColumn(name="idChercheur",referencedColumnName="id"
    ,insertable=false,updatable=false)
    public Chercheur getChercheur() {
        return chercheur;
    }
```

```java
@Entity

public class Chercheur extends Employe implements Serializable {

    private List<Tache> taches;
    @OneToMany(mappedBy="chercheur")
    public List<Tache> getTaches() {
        return taches;
    }
    public void setTaches(List<Tache> taches) {
        this.taches = taches;
    }
```


```java
@Entity
public class Projet implements Serializable {

    private List<Tache> taches;

    @OneToMany(mappedBy = "projet")
    public List<Tache> getTaches() {
        return taches;
    }

    public void setTaches(List<Tache> taches) {
        this.taches = taches;
    }
```

# FETCH CASCADE



```
@OneToMany(mappedBy="entreprise",
cascade = {CascadeType.PERSIST, CascadeType.REMOVE},
fetch=FetchType.EAGER)
private List<Departement> departements = new ArrayList<>();
```

# Modèle de données existant



- Principe:
  - Correspondance attributs et colonnes
  - Annotations @Table et @Column

```
@Entity
@Table(name="ECRIVAINS")
public class Auteur {
  @Column("NOM_ID")
  String nom;
  @OneToMany(mappedBy="auteur")
  List<Ouvrage> oeuvres;
  // constructeur
  // getter/setter nom
  // getter/setter oeuvres
}
```

```
@Entity
@Table(name="LIVRES")
public class Ouvrage {
  @COLUMN("TITRE_ID")
  String titre;
  @ManyToOne
  @JoinColumn("NOM_ID")
  Auteur auteur;
  // constructeur
  // getter/setter titre
  // getter/setter auteur
}
```

# Résumé



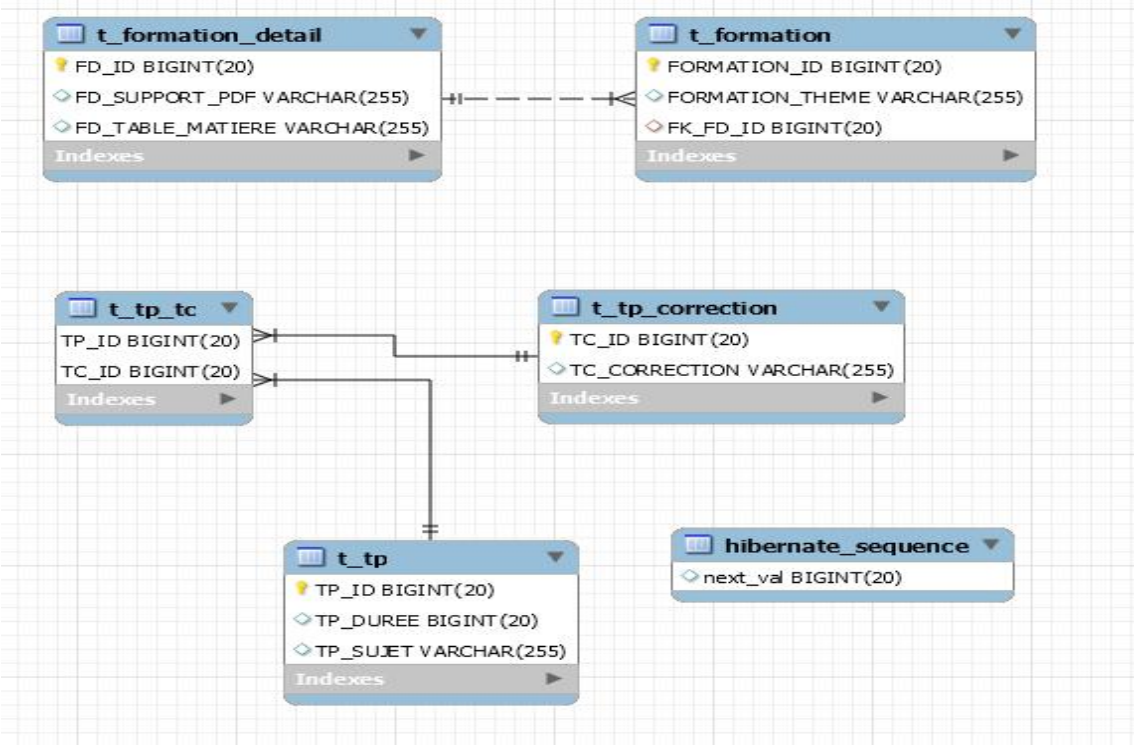
- **Les associations**
- Pour marquer les relations entre nos entités il est possible d'utiliser les annotations :
- **@OneToMany :** pour désigner une relation 1-n, soit une instance faisant référence à plusieurs autres. Permet d'accéder aux objets du coté N à travers une liste. A utiliser avec vigilance selon la volumétrie des données associées.
- **@ManyToOne** : pour désigner, une relation inverse de 1-n. Permet d'accéder facilement à l'objet du coté "1".
- **@OneToOne** : pour désigner une relation 1-1 entre deux objets. Peut être utile pour séparer un ensemble de propriétés dont le sens est différent. Peut être utile pour concevoir une relation d'héritage également.
- **@ManyToMany** : pour les relations n-n devenant une table associative. Certaines associations N-N du modèle peuvent comporter des informations. Dans ce cas, il est préférable d'utiliser deux liens OneToMany vers une classe qui portera ces informations. Pour gérer l'association correctement, il y a ensuite deux stratégies concernant la clé primaire :

- 1. Réaliser une clé composée à l'aide de EmbeddedId ; (Voir cette explication http://www.mkyong.com/hibernate/hibernate-many-to-many-example-join-table-extra-column-annotation/
- 2. Ajouter une simple clé (colonne supplémentaire) et mettre une contrainte d'unicité sur les colonnes des deux clés étrangères. ; (Voir cette explication http://stackoverflow.com/questions/5127129/mapping-many-to-many-association-table-with-extra-columns

# TP1

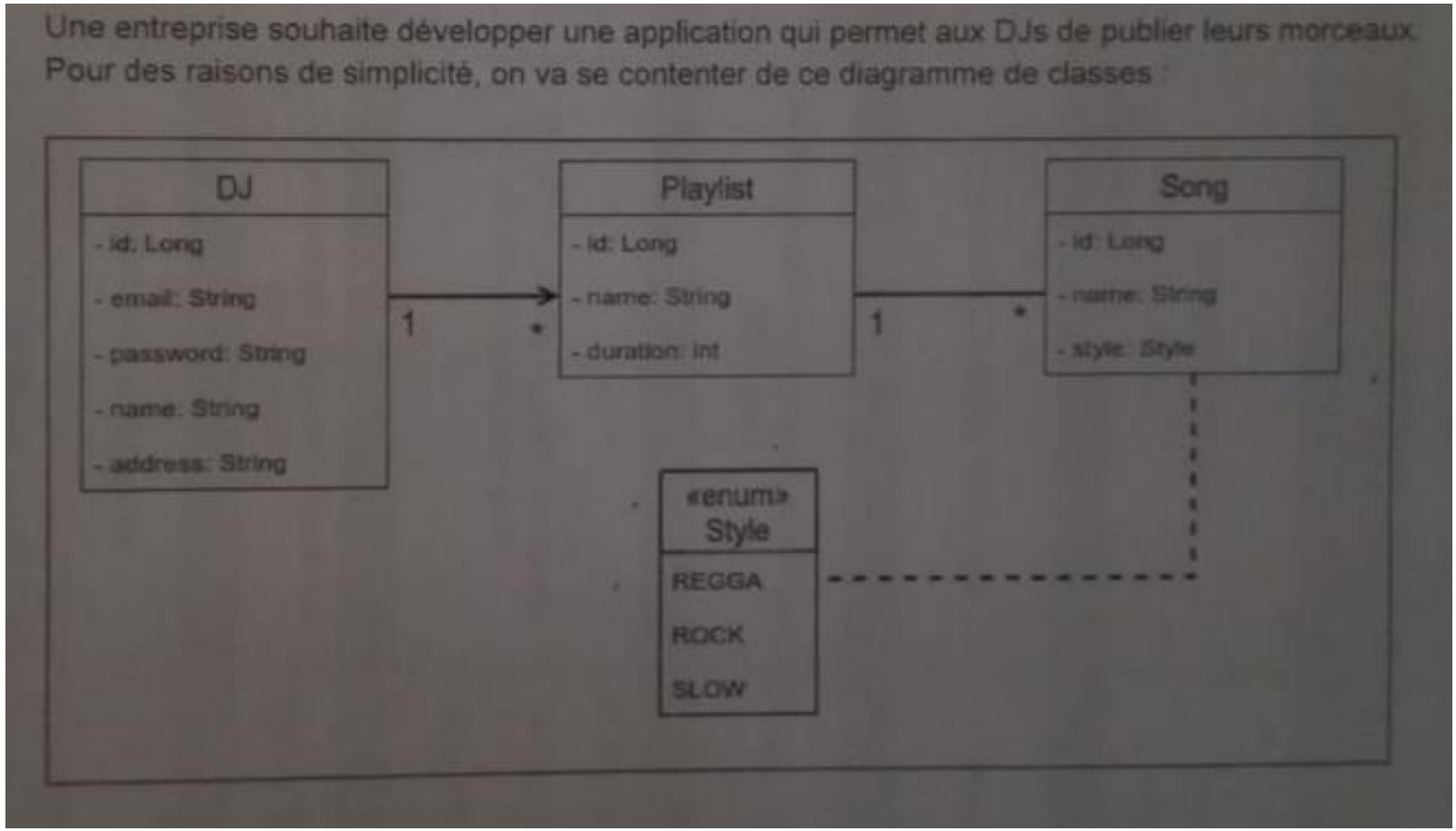


- Tester toutes les associations vues en cours.
- Utiliser MySQL Workbench pour générer le diagramme correspondant aux différentes tables (Cliquer sur l'onglet Database – Reverse Engineer).



# TP2



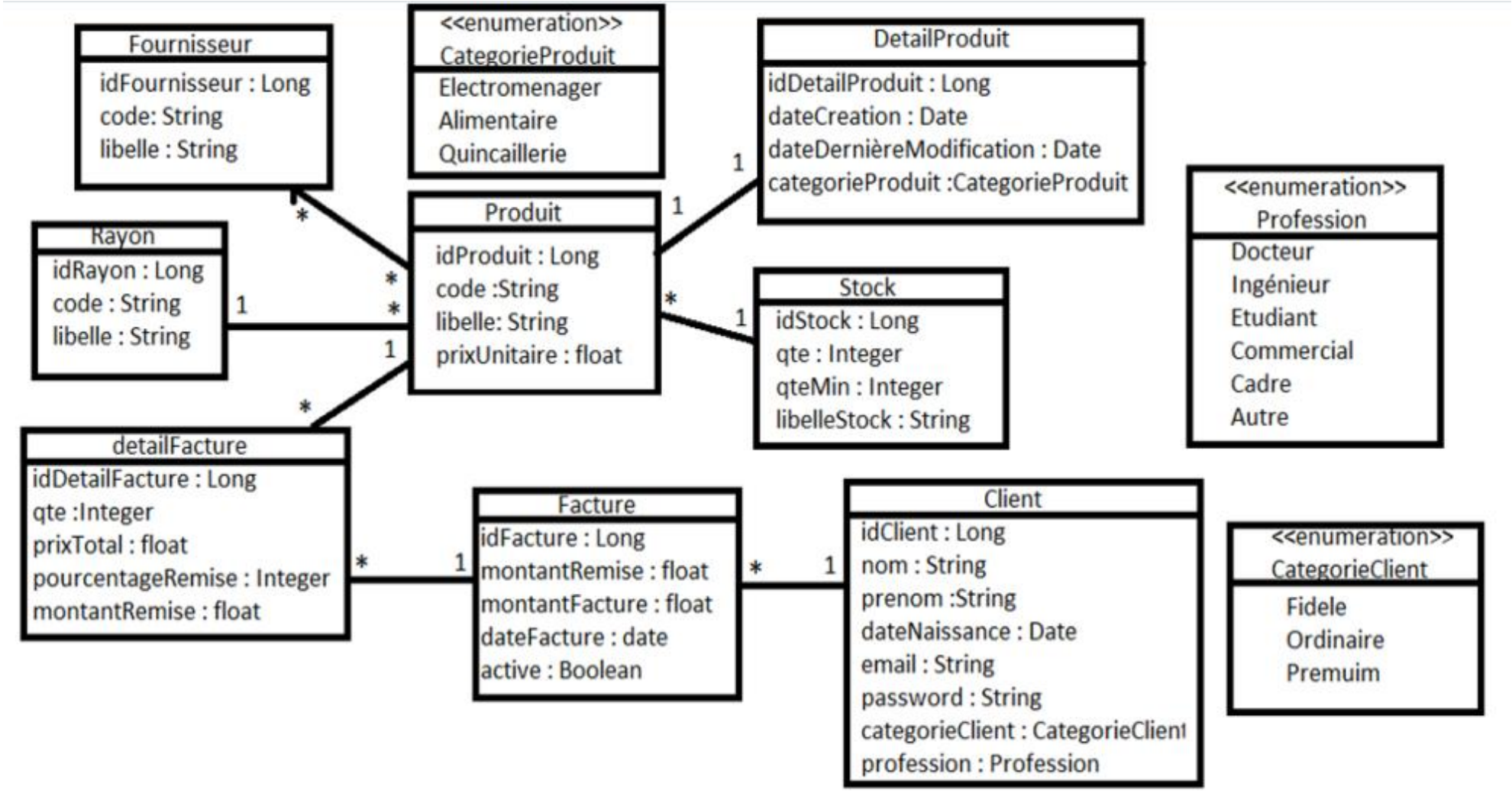



Implémenter les entités qui permettent de générer le schéma de la base de données comme illustré dans le diagramme de classe que:

- Les ids des entités DJ, Playlist et Song ne sont pas auto-générés.

# TP3




Fournisseur
idFournisseur : Long
code: String
libelle : String
<<enumeration>>
CategorieProduit
Electromenager
Alimentaire
Quincaillerie
DetailProduit
idDetailProduit : Long
dateCreation : Date
dateDernièreModification : Date
categorieProduit :CategorieProduit
<<enumeration>>
Profession
Docteur
Ingénieur
Etudiant
Commercial
Cadre
Autre
Rayon
idRayon : Long
code : String
libelle : String
Produit
idProduit : Long
code :String
libelle: String
prixUnitaire : float
Stock
idStock : Long
qte : Integer
qteMin : Integer
libelleStock : String
detailFacture
idDetailFacture : Long
qte :Integer
prixTotal : float
pourcentageRemise : Integer
montantRemise : float
Facture
idFacture : Long
montantRemise : float
montantFacture : float
dateFacture : date
active : Boolean
Client
idClient : Long
nom : String
prenom :String
dateNaissance : Date
email : String
password : String
categorieClient : CategorieClient
profession : Profession
<<enumeration>>
CategorieClient
Fidele
Ordinaire
Premuim
1
*

# TP3



- **Partie 2 Spring Data JPA – Le mapping des différentes associations**
  - Dans le projet tp magasin/Stock et après avoir créer les entités lors de la dernière séance, vous devez :
    - Supprimer les tables existantes dans la base de données.
    - Créer les associations entre les différentes entités.
    - Générer la base de donné de nouveau et vérifier que le nombre de tables crées est correcte.

# LA SPÉCIFICATION JPA – ASSOCIATION
# SPRING DATA JPA
# SPRING BOOT

2021-2022

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)