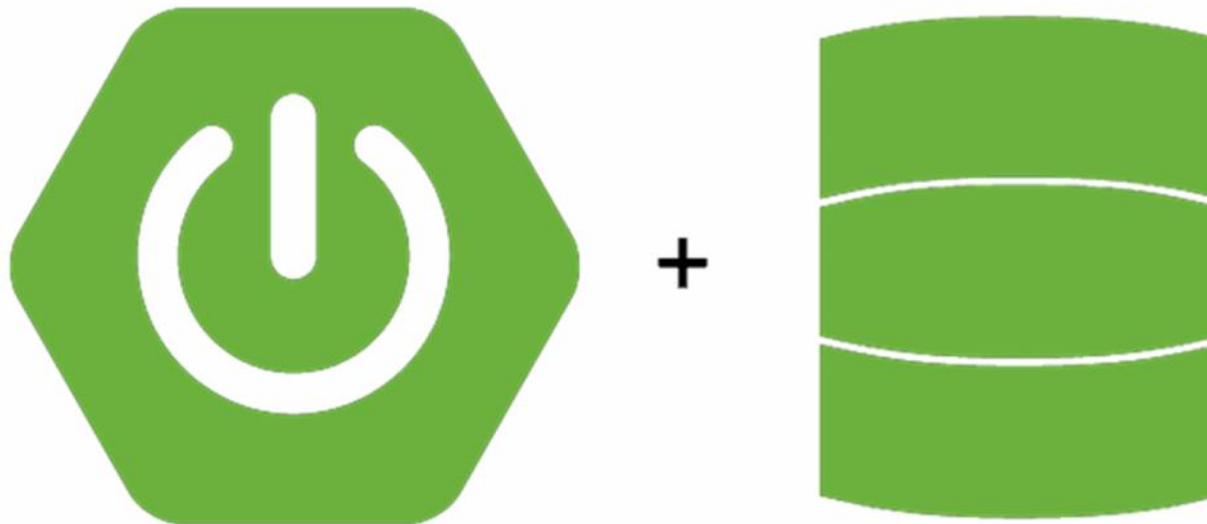


SPRING DATA JPA – ASSOCIATIONS



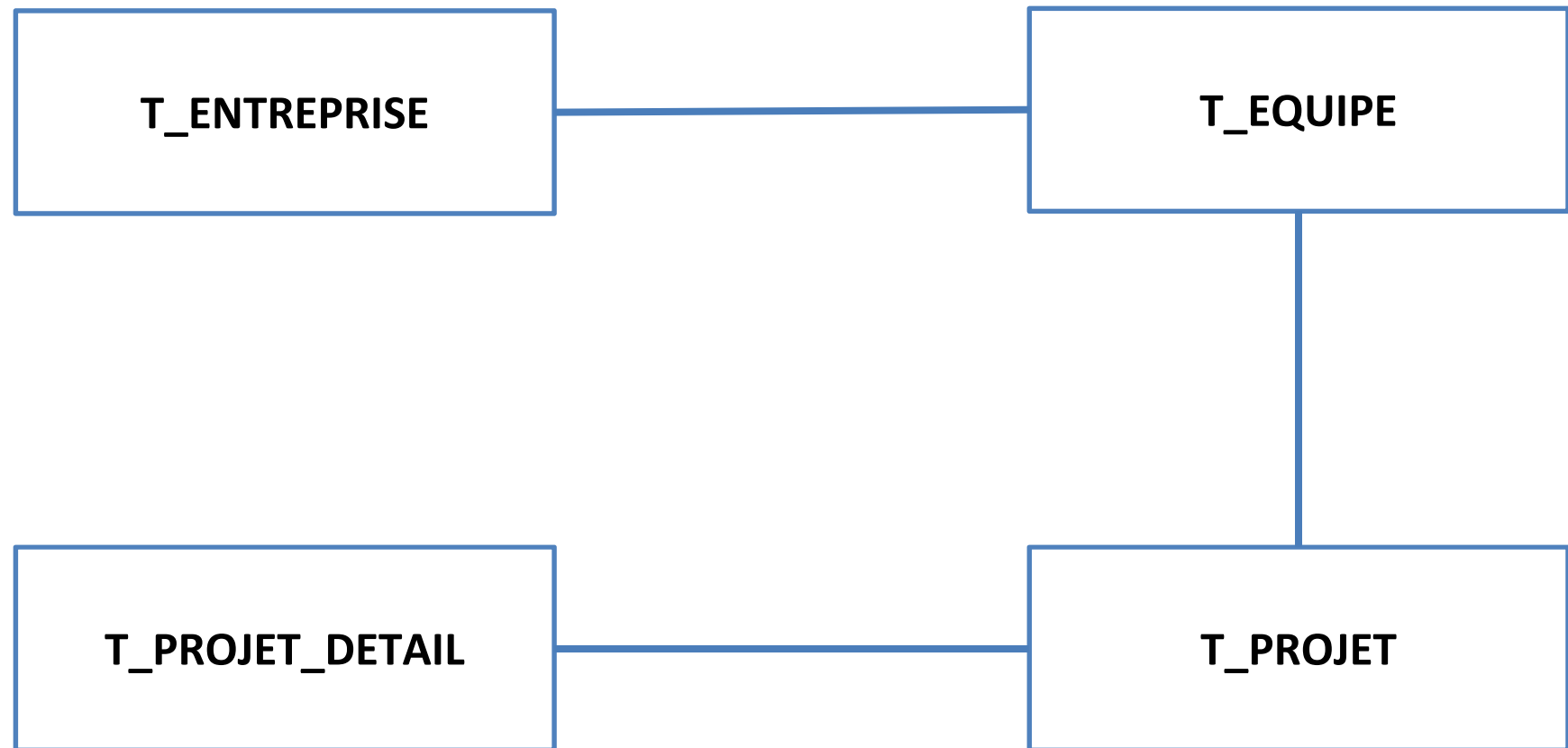
**UP ASI
Bureau E204**

PLAN DU COURS

- Associations entre entités :
 - **One to One** (1:1) - Unidirectionnelle / Bidirectionnelle
 - **One to Many** (1:N) - Unidirectionnelle / Bidirectionnelle
 - **Many to One** (N:1) - Unidirectionnelle / Bidirectionnelle
 - **Many to Many** (N:M) - Unidirectionnelle / Bidirectionnelle

- **TP** : Mise en œuvre des différentes associations

Diagramme de Classes (sans cardinalité)



Trouvez et Expliquez les Associations

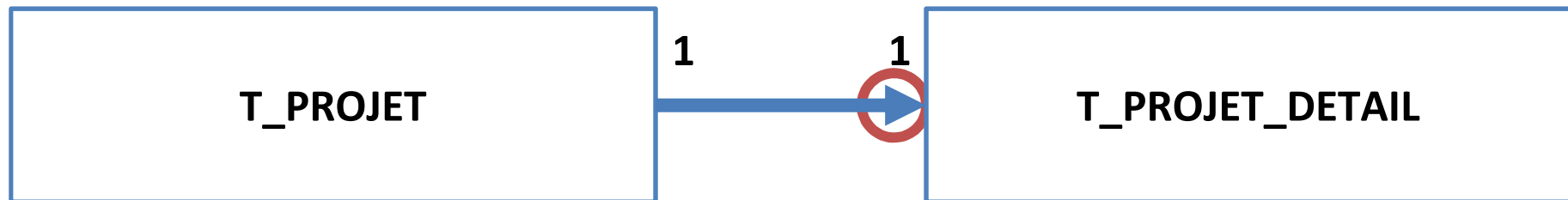
- **One To One** entre T_PROJET et T_PROJET_DETAIL (Clé étrangère projet_detail_pd_id).
- Le projet a un seul détail (une seule ligne dans la table T_PROJET_DETAIL). Le détail d'un projet est lié à un seul Projet.
- **Many To Many** : T_EQUIPE et T_PROJET (Table d'association T_EQUIPE_PROJETS).
- L'Equipe a plusieurs Projets (Projet1, Projet2, ...). Un même projet peut être lié à plusieurs équipes (équipe Développement web, équipe DevOps...).
- **One To Many** : L'Entreprise peut avoir plusieurs Equipes (Développement web, Développement mobile, DevOps, Test...). Une équipe n'est liée qu'à une seule entreprise.
- Place à la pratique, pour créer vous-même toutes ces associations :

Diagramme de Classes (avec cardinalité)



One To One Unidirectionnelle

- **One To One** : Le Projet a un seul détail (une seule ligne dans la table T_PROJET_DETAIL). Le détail d'un projet est lié à un seul Projet.
- **Unidirectionnelle** : Le Projet connaît le détail (contient un attribut de type Projet_Detail), alors que le détail n'a aucune information sur le Projet auquel il est associé.



One To One Unidirectionnelle

```
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "PROJET_SUJET")
    private String sujet;

    @OneToOne
    private Projet_Detail projetDetail;

}
```

One To One Unidirectionnelle

```
@Entity
@Table(name = "T_PROJET_DETAIL")
public class Projet_Detail implements Serializable {
    private static final long serialVersionUID = 1L;

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

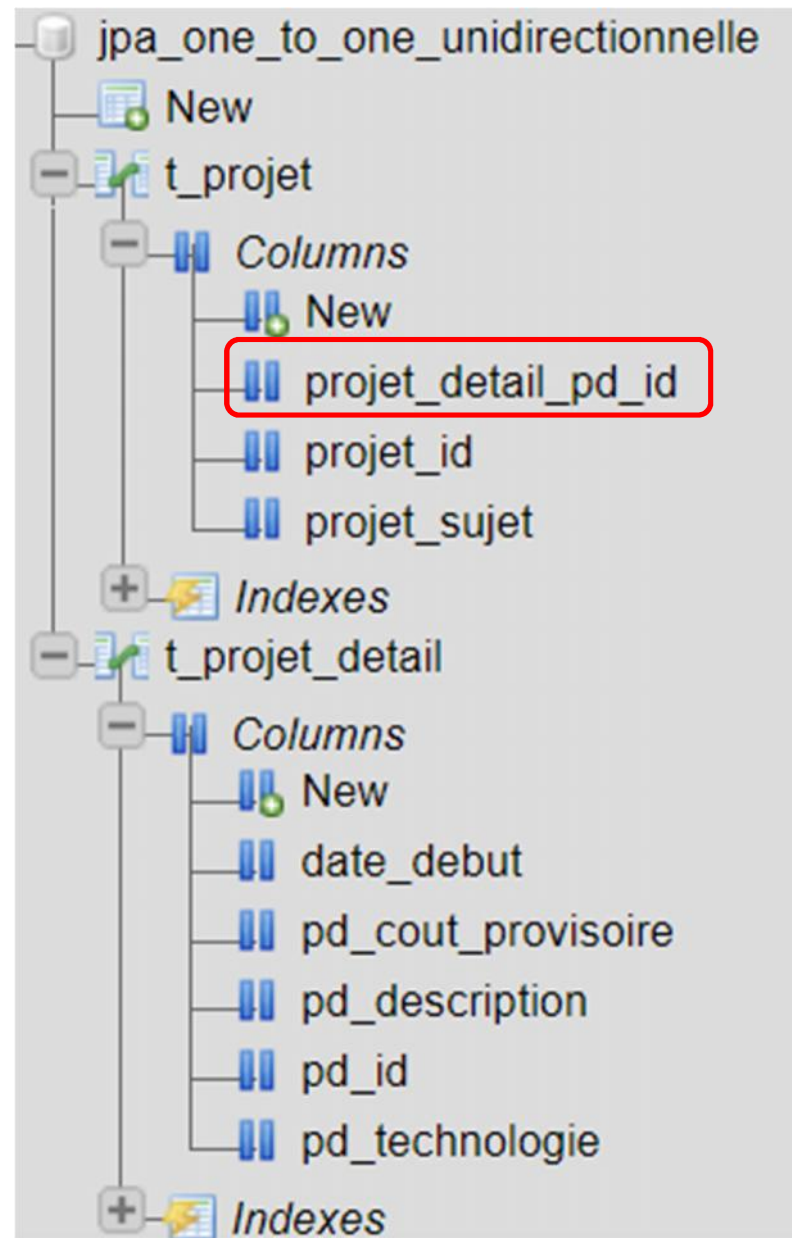
    @Column(name = "PD_DESCRIPTION")
    private String description;

    @Column(name = "PD_TECHNOLOGIE")
    private String technologie;

    @Column(name = "PD_COUT_PROVISOIRE")
    private Long cout_provisoire;

    @Temporal(TemporalType.DATE)
    private Date dateDebut;
}
```


One To One Unidirectionnelle



One To One Bidirectionnelle

- **One To One** : Le Projet a un seul détail (une seule ligne dans la table T_PROJET_DETAIL). Le détail d'un projet est lié à un seul Projet.
- **Bidirectionnelle** : l'Entité «Projet» contient un attribut de type «Projet_Detail» et l'Entité «Projet_Detail» contient un attribut de type «Projet».
- 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.



One To One Bidirectionnelle

```
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "PROJET_SUJET")
    private String sujet;

    @OneToOne
    private Projet_Detail projetDetail;

}
```

One To One Bidirectionnelle

```
@Entity
@Table(name = "T_PROJET_DETAIL")
public class Projet_Detail implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "PD_DESCRIPTION")
    private String description;

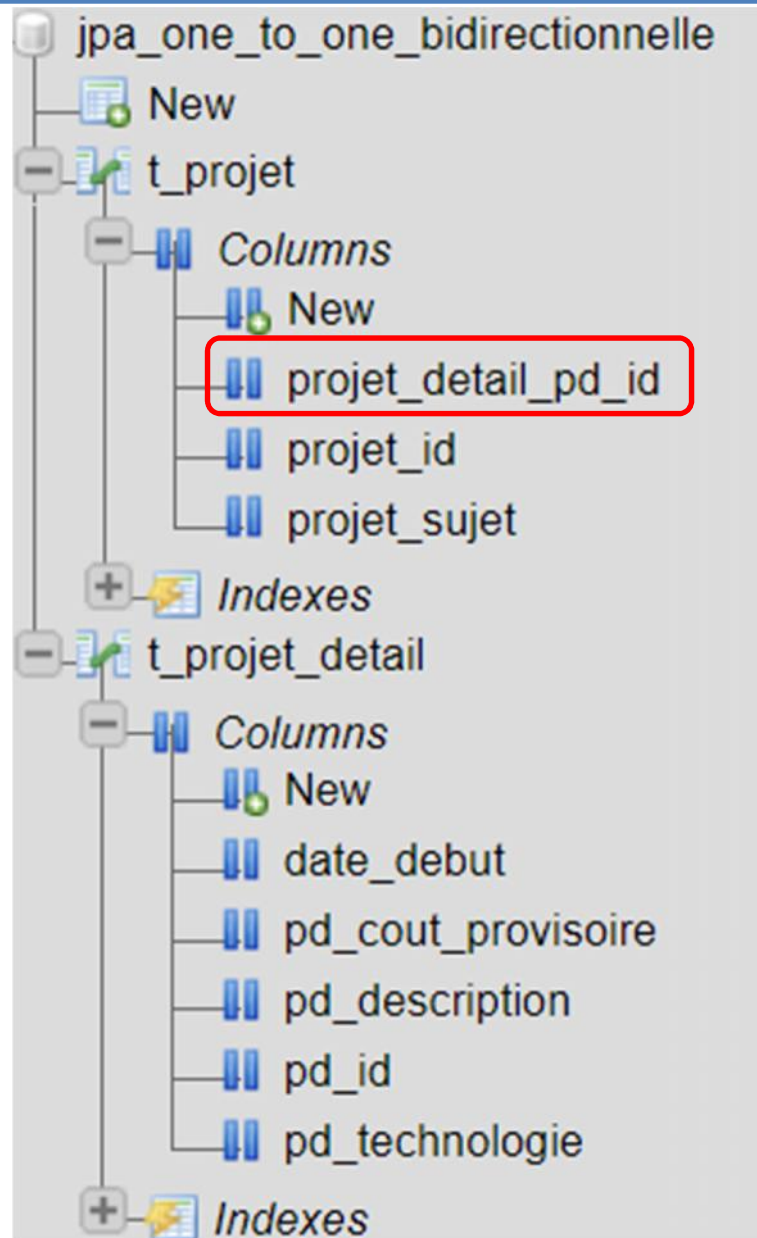
    @Column(name = "PD_TECHNOLOGIE")
    private String technologie;

    @Column(name = "PD_COUT_PROVISOIRE")
    private Long cout_provisoire;

    @Temporal(TemporalType.DATE)

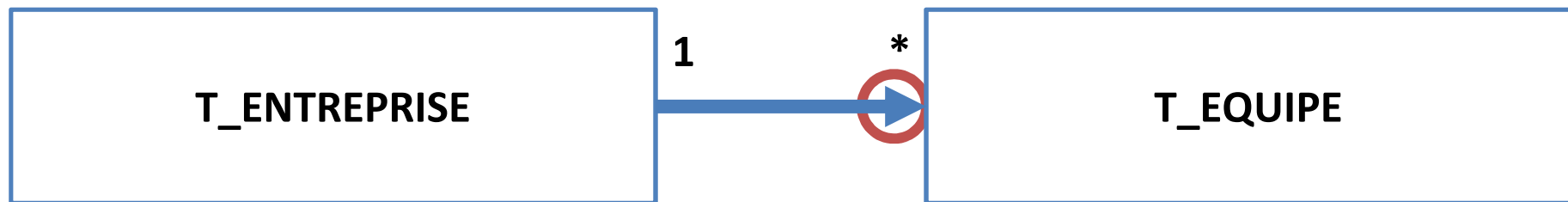
    private Date dateDebut;
    @OneToOne(mappedBy="projetDetail")
    private Projet projet;
}
```

One To One Bidirectionnelle



One To Many Unidirectionnelle

- **One To Many** : Une entreprise peut avoir plusieurs équipes (Développement web, Développement mobile, DevOps, Test...). Une équipe n'est liée qu'à une seule entreprise.
- **Unidirectionnelle Entreprise → Equipe** : L'entreprise connaît les équipes, alors que l'équipe n'a aucune information sur l'entreprise à laquelle elle appartient.



One To Many Unidirectionnelle

```
@Entity
@Table(name = "T_ENTREPRISE")
public class Entreprise implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "ENTREPRISE_NOM")
    private String nom;

    @Column(name = "ENTREPRISE_ADRESSE")
    private String adresse;

    @OneToMany(cascade = CascadeType.ALL)
    private Set<Equipe> Equipes;
}
```

One To Many Unidirectionnelle

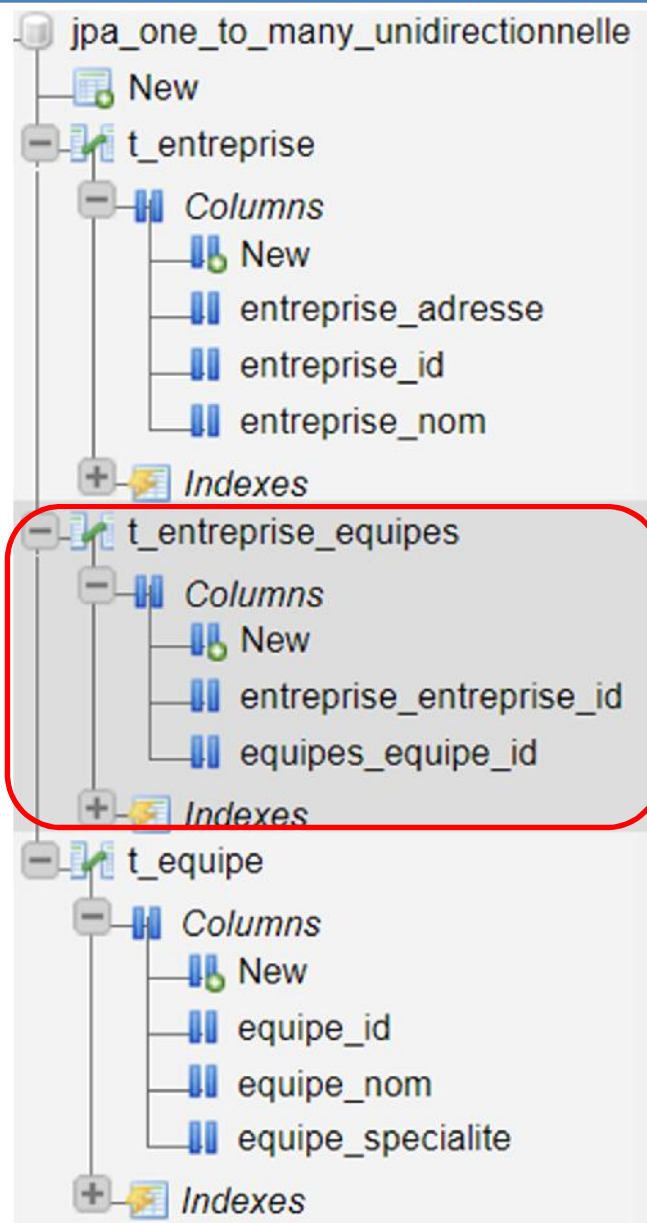
```
@Entity
@Table(name = "T_EQUIPE")
public class Equipe implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "EQUIPE_NOM")
    private String nom;

    @Column(name = "EQUIPE_SPECIALITE")
    private String specialite;
}
```


One To Many Unidirectionnelle

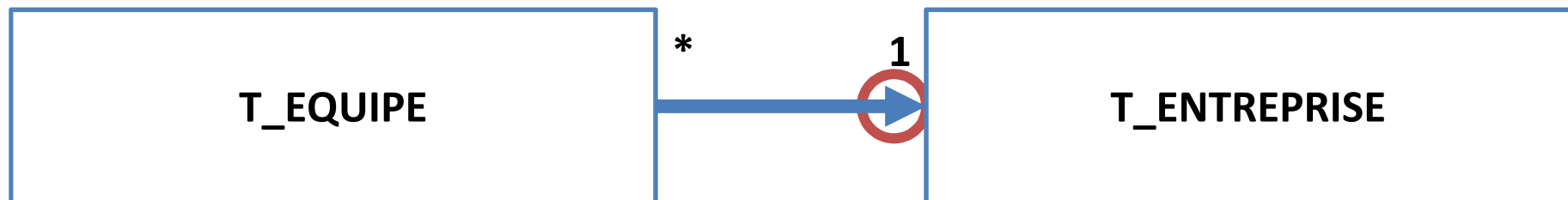


One To Many Bidirectionnelle

- **One To Many Bidirectionnelle = Many To One Bidirectionnelle :**
- Voir Slide Many To One Bidirectionnelle dans la suite du cours

Many To One Unidirectionnelle

- **Many To One** : Une entreprise peut avoir plusieurs équipes. Chaque équipe est liée à une unique entreprise.
- **Unidirectionnelle** : Chaque équipe a l'information concernant l'entreprise (attribut Entreprise dans l'Entité Equipe, Clé étrangère dans la table T_EQUIPE), alors que l'Entreprise n'a aucune information sur ses «Equipes».



Many To One Unidirectionnelle

```
@Entity
@Table(name = "T_EQUIPE")
public class Equipe implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "EQUIPE_NOM")
    private String nom;

    @Column(name = "EQUIPE_SPECIALITE")
    private String specialite;

    @ManyToOne(cascade = CascadeType.ALL)
    Entreprise entreprise;
}
```

Many To One Unidirectionnelle

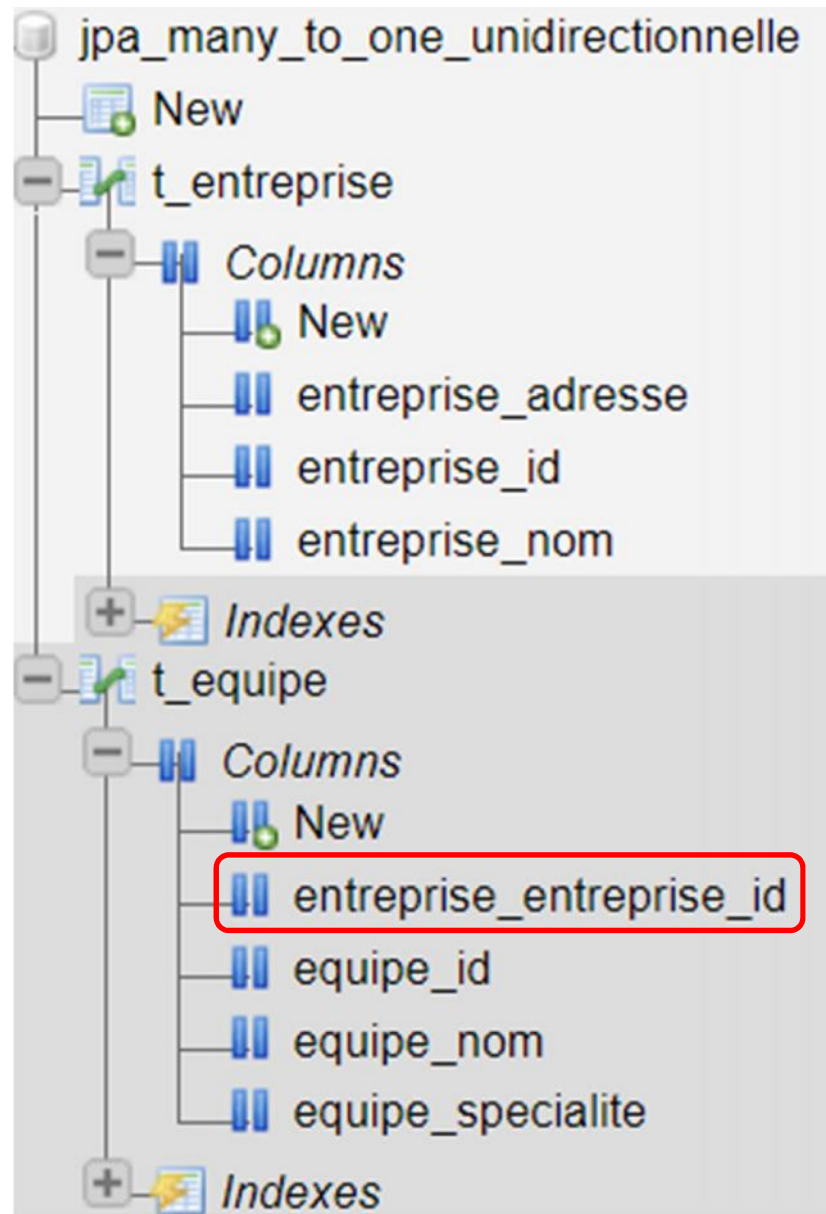
```
@Entity
@Table(name = "T_ENTREPRISE")
public class Entreprise implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "ENTREPRISE_NOM")
    private String nom;

    @Column(name = "ENTREPRISE_ADRESSE")
    private String adresse;
}
```

Many To One Unidirectionnelle



Many To One Bidirectionnelle

- **Many To One** : Une entreprise peut avoir plusieurs Equipes. Une Equipe est liée à une seule Entreprise.
- **Bidirectionnelle** : L'entreprise connaît ses Equipes. Chaque Equipe connaît elle aussi l'Entreprise associée.



- L'attribut **mappedBy** est défini pour l'annotation @OneToMany (toujours au niveau de l'entité qui a **la cardinalité la plus faible**).

Many To One Bidirectionnelle

```
@Entity
@Table(name = "T_EQUIPE")
public class Equipe implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "EQUIPE_NOM")
    private String nom;

    @Column(name = "EQUIPE_SPECIALITE")
    private String specialite;

    @ManyToOne
    Entreprise entreprise;
}
```


Many To One Bidirectionnelle

```
@Entity
@Table(name = "T_ENTREPRISE")
public class Entreprise implements Serializable {
    private static final long serialVersionUID = 1L;

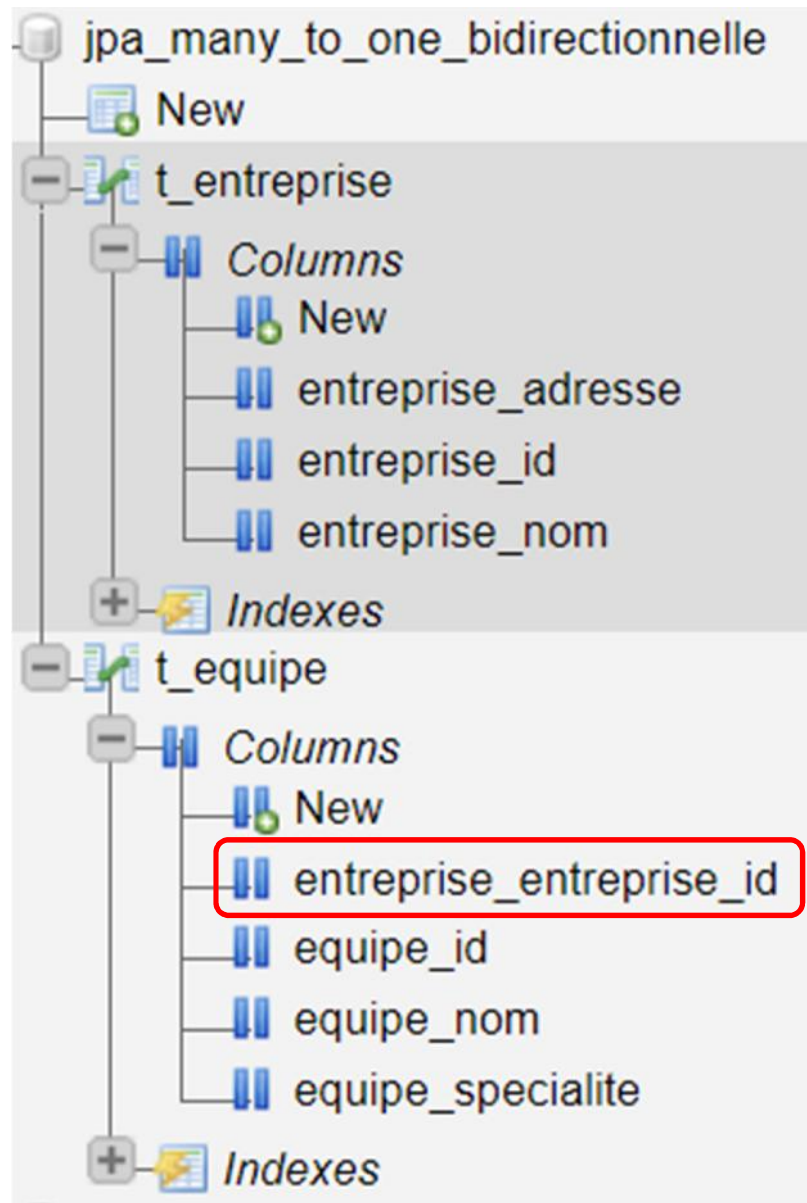
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ENTREPRISE_ID")
    private Long id; // Identifiant entreprise (Clé primaire)

    @Column(name = "ENTREPRISE_NOM")
    private String nom;

    @Column(name = "ENTREPRISE_ADRESSE")
    private String adresse;

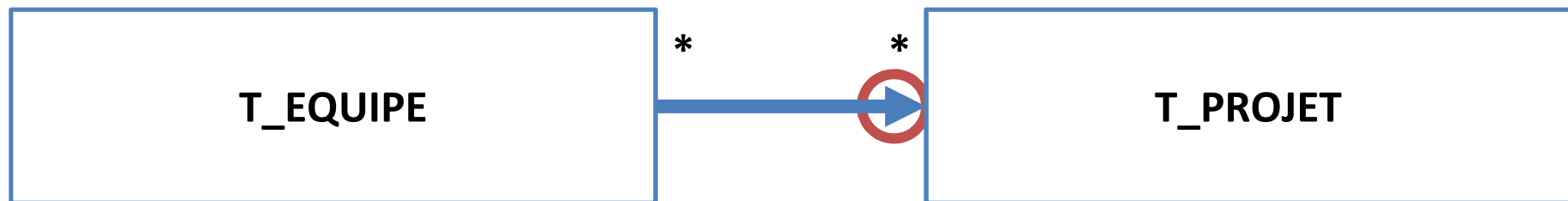
    @OneToMany(cascade = CascadeType.ALL, mappedBy="entreprise")
    private Set<Equipe> Equipes;
}
```

Many To One Bidirectionnelle



Many To Many Unidirectionnelle

- **Many To Many** : Une équipe peut travailler sur plusieurs Projets (Projet1, Projet2, ...). Un même projet peut être lié à plusieurs équipes (équipe Développement web, équipe DevOps...).
- **Unidirectionnelle** : L'équipe a plusieurs Projets et les connaît. Mais, le Projet n'a aucune information sur «les équipes» auxquelles il est associé.



Many To Many Unidirectionnelle

```
@Entity
@Table(name = "T_EQUIPE")
public class Equipe implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "EQUIPE_NOM")
    private String nom;

    @Column(name = "EQUIPE_SPECIALITE")
    private String specialite;

    @ManyToMany(cascade = CascadeType.ALL)
    private Set<Projet> projets;
}
```

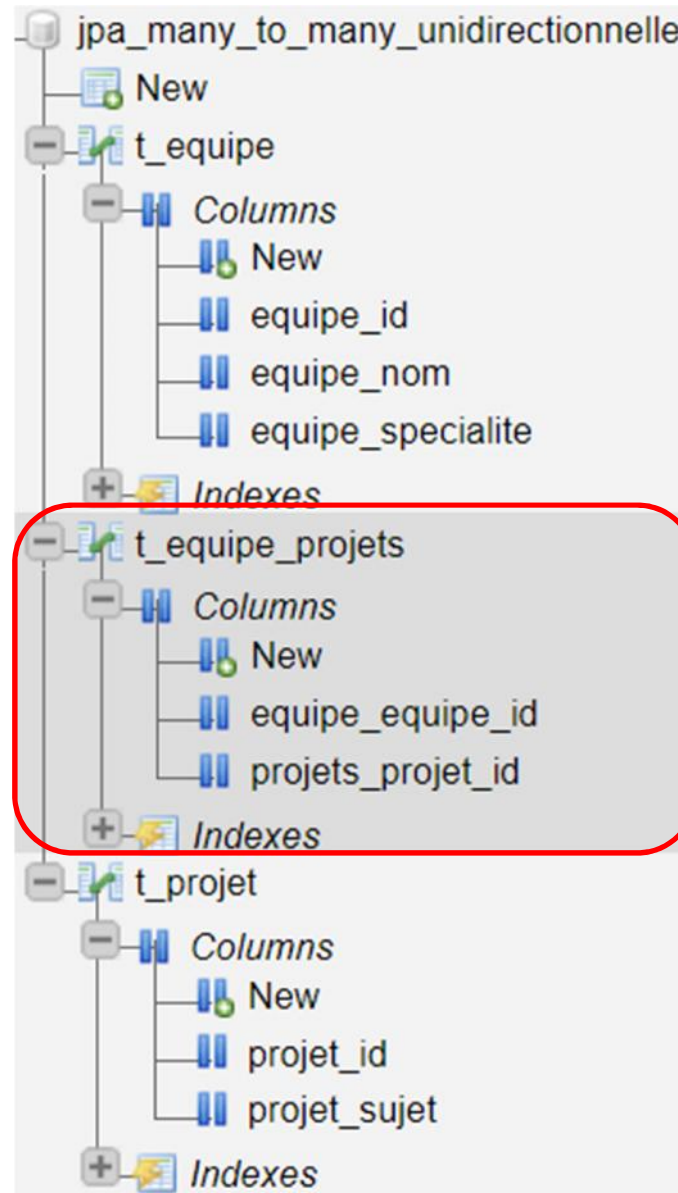
Many To Many Unidirectionnelle

```
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
    private static final long serialVersionUID = 1L;

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

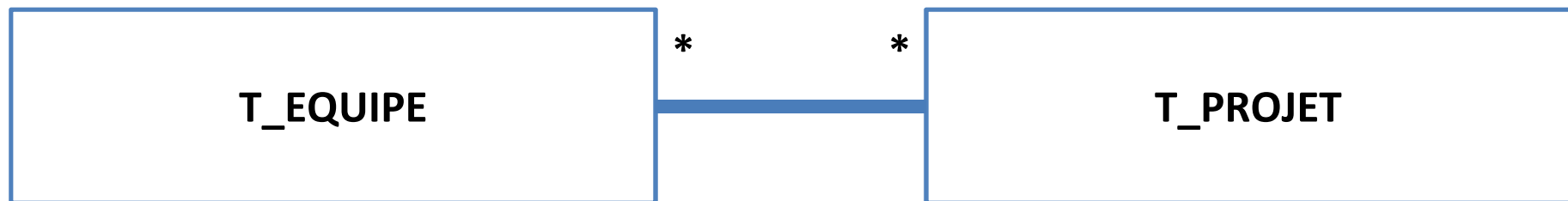
    @Column(name = "PROJET_SUJET")
    private String sujet;
}
```

Many To Many Unidirectionnelle



Many To Many Bidirectionnelle

- **Many To Many** : Une équipe peut travailler sur plusieurs Projets (Projet1, Projet2, ...). Un même projet peut être lié à plusieurs équipes (équipe Développement web, équipe DevOps...).
- **Bidirectionnelle** : L'équipe a plusieurs Projets et les connaît. Chaque Projet est associé à plusieurs Equipes, et peut accéder aux attributs de la table équipe.



Many To Many Bidirectionnelle

```
@Entity
@Table(name = "T_EQUIPE")
public class Equipe implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "EQUIPE_NOM")
    private String nom;

    @Column(name = "EQUIPE_SPECIALITE")
    private String specialite;

    @ManyToMany(cascade = CascadeType.ALL)
    private Set<Projet> projets;
}
```


Many To Many Bidirectionnelle

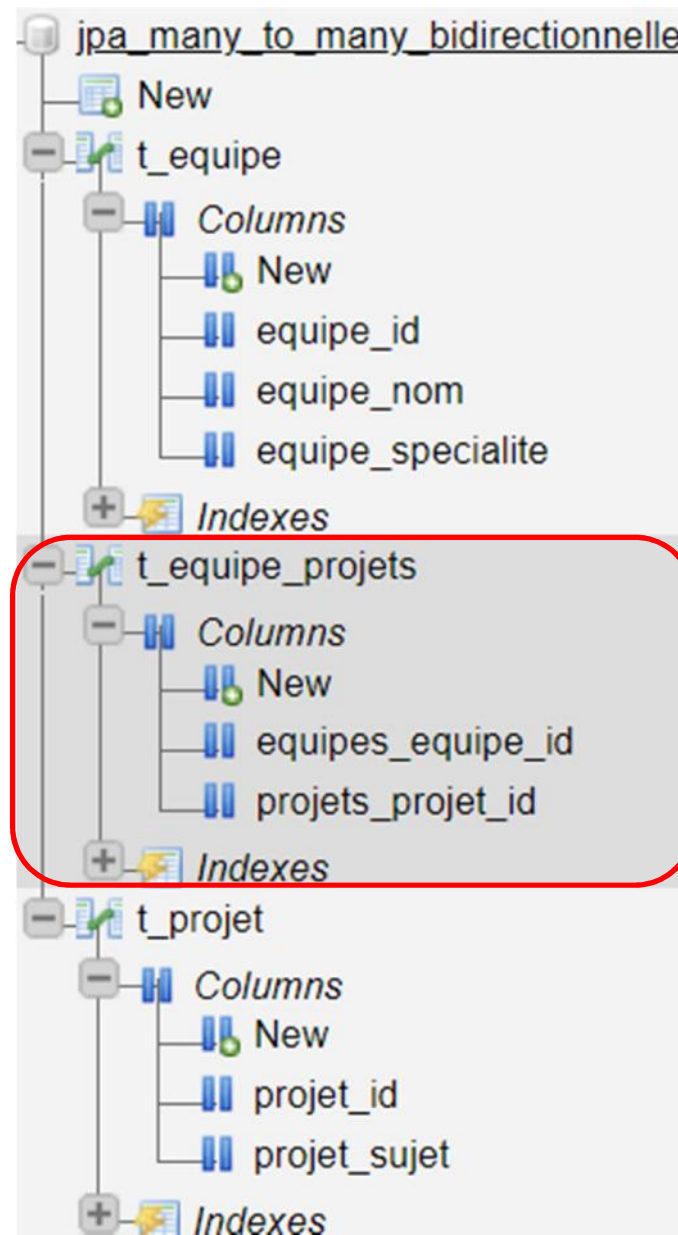
```
@Entity
@Table(name = "T_PROJET")
public class Projet implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "PROJET_SUJET")
    private String sujet;

    @ManyToMany(mappedBy="projets", cascade = CascadeType.ALL)
    private Set<Equipe> equipes;
}
```

Many To Many Bidirectionnelle



CASCADE

```
@ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.REMOVE},  
fetch=FetchType.EAGER)  
private Set<Projet> projets;
```

- **CascadeType.ALL** : Cascade toutes les opérations (PERSSIT, REMOVE, ...) du parent vers le child.
- **CascadeType.REMOVE** : Cascade l'opération REMOVE (Suppression) du parent vers le child. Ci-dessus, quand on supprime une équipe, alors les Projets de cette équipe seront supprimés (pour éviter d'avoir des projets orphelins : sans équipes).
- **CascadeType.PERSIST** : Cascade l'opération PERSSIT (Ajout) du parent vers le child. Ci-dessus, quand on ajoute une équipe, alors, si l'objet équipe ajouté contient des projets, l'équipe et les projets seront ajoutés.
- **Par défaut** (si on ne met pas CascadeType), aucune opération n'est cascadée.

FETCH

```
@ManyToMany(fetch=FetchType.EAGER)
```

```
private Set<Projet> projets;
```

- **FetchType.EAGER** (avec impatience): Quand on récupère une équipe de la base de données, tous les projets liés à cette équipe seront récupérés eux aussi.
- **FetchType.LAZY** (avec paresse): Quand on récupère une équipe de la base de données, aucun projet lié à cette équipe ne sera récupéré, jusqu'à ce que nous faisons un appel explicite dans le code : `equipe.projets` par exemple (`equipe` étant une instance de `Equipe`).
- **Par défaut** (Si on ne met pas `FetchType`), les valeurs par défaut sont :
 - **OneToMany** et **ManyToMany** : **LAZY**
 - **ManyToOne** et **OneToOne** : **EAGER**(Quand c'est Many à la fin, c'est LAZY car on risque de récupérer beaucoup d'éléments "Many" c'est dangereux. Quand c'est One à la fin c'est EAGER car le volume des données associés n'est pas très important.)

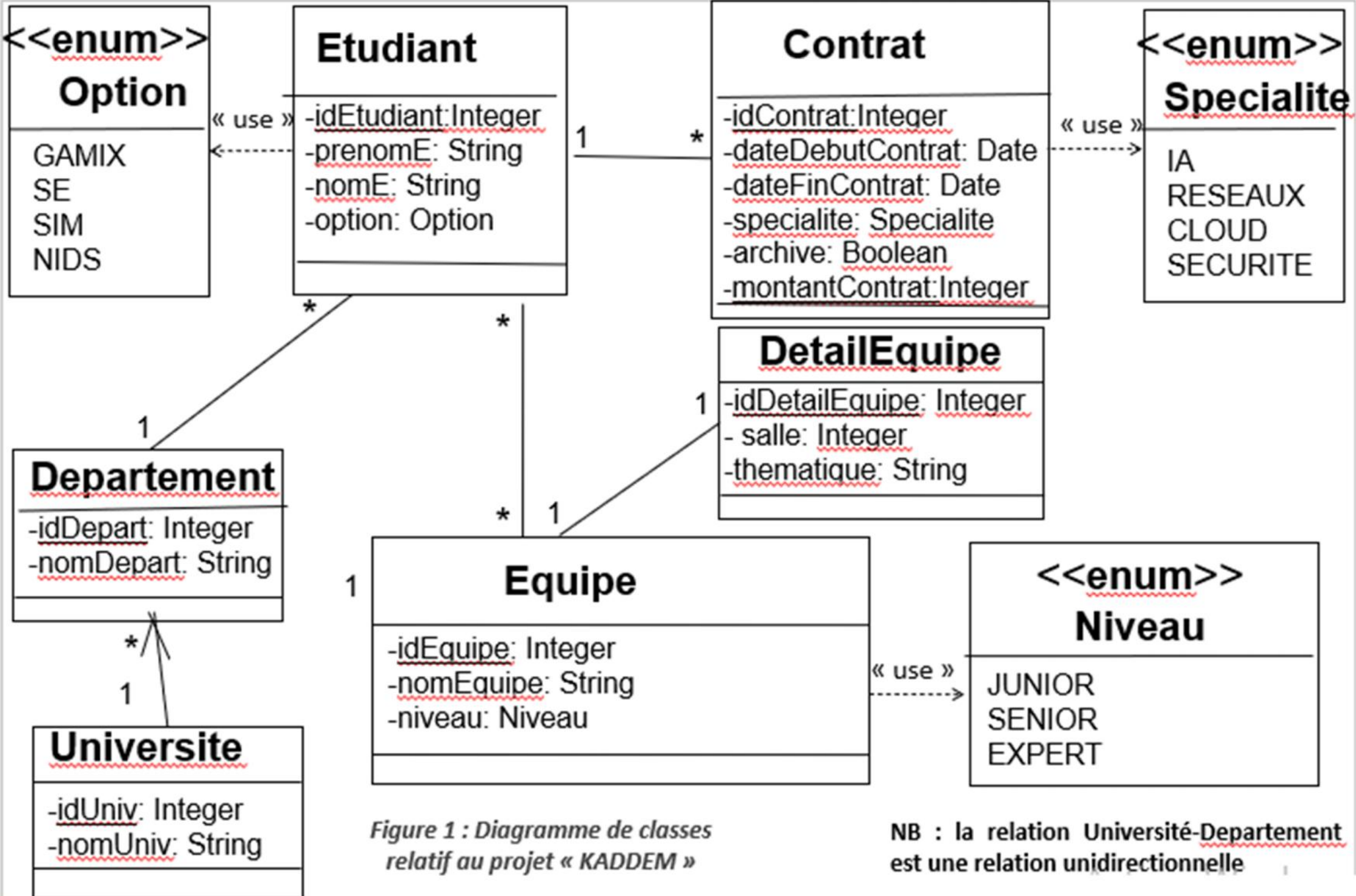


Figure 1 : Diagramme de classes
relatif au projet « KADDEM »

NB : la relation Université-Departement
est une relation unidirectionnelle

Travail à faire

Partie 2 Spring Data JPA – Le mapping des différentes associations

Dans l'étude de cas Khaddem 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 données de nouveau et vérifier que le nombre de tables créées est correct.

Spring Data JPA – Associations

Si vous avez des questions, n'hésitez pas à nous contacter :

Département Informatique
UP Architectures des Systèmes d'Information
Bureau E204