Framework de persistance
HJPA : Hibernate Java Persistance Api

A. MADANI

Madani.a@ucd.ac.ma

ORM avec Hibernate JPA

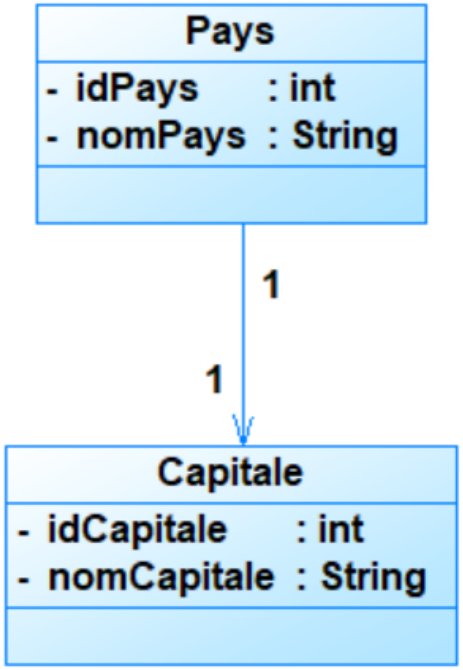
- Introduction/Rappel
- Gestion de la correspondance objet/relationnel
- ORM: Object Relationnel Mapping
- Framework: Hibernate, Tooplik,…
- JPA: Java persistance API
- Mapping Objet Relationnel Avec JPA (entités, annotations, fichier de configuration)
- Gestion d'une entité
- Requêtes : JP QL, Creteria API, SQL Native
- Mapping des relations

madani.a@ucd.ac.ma

Mapping des associations

madani.a@ucd.ac.ma

Mapping des associations (Rappels)

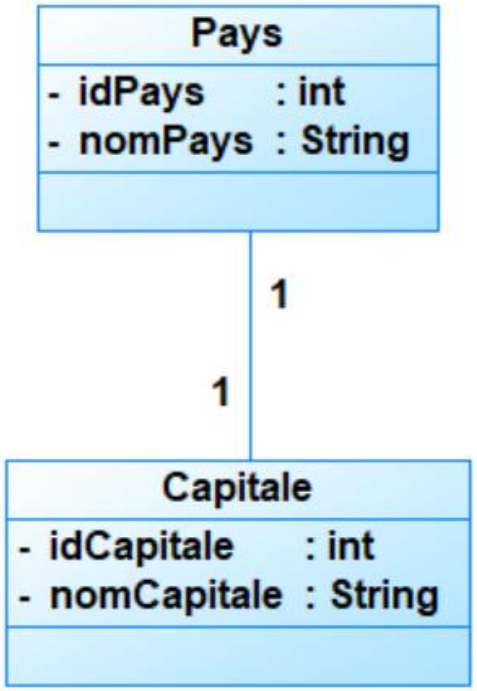



```
public class Capitale {
    private int idCapitale;
    private String nomCapitale;
}
```

```
public class Pays {
    private int idPays;
    private String nomPays;
    public Capitale capitale;
}
```


madani.a@ucd.ac.ma

Mapping des associations (Rappels)

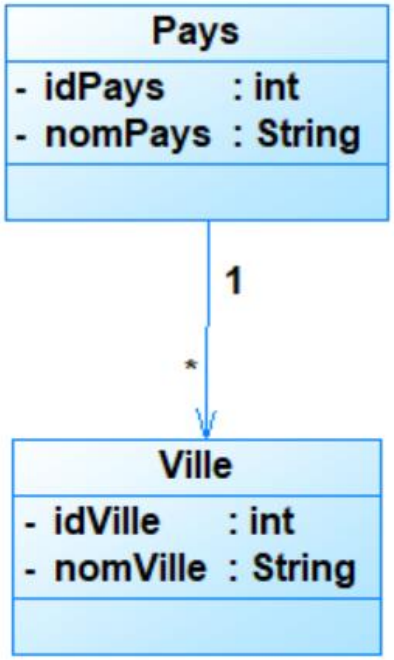



```
public class Capitale {
    private int idCapitale;
    private String nomCapitale;
    private Pays pays;
}
```

```
public class Pays {
    private int idPays;
    private String nomPays;
    public Capitale capitale;
}
```


madani.a@ucd.ac.ma

Mapping des associations (Rappels)

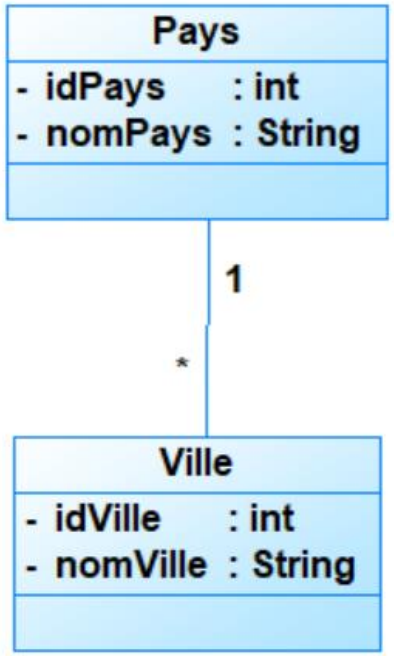



```
public class Ville {
    private int idVille;
    private String nomVille;
}
```

```
public class Pays {
    private int idPays;
    private String nomPays;
    Public Collection<Ville> villes;
}
```


madani.a@ucd.ac.ma

Mapping des associations (Rappels)

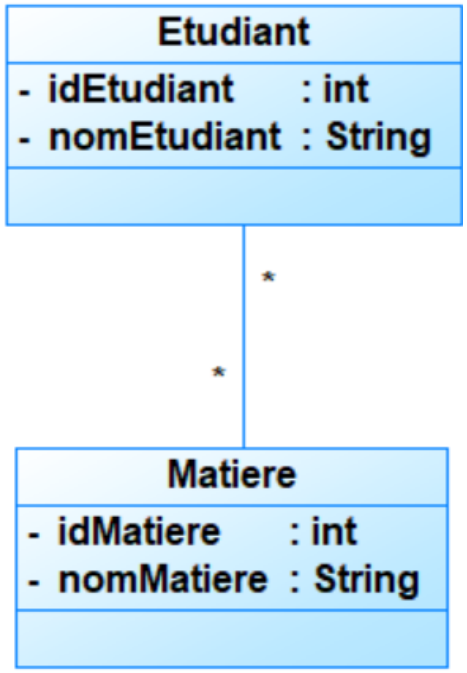



```
public class Ville {
    private int idVille;
    private String nomVille;
    private Pays pays;
}
```

```
public class Pays {
    private int idPays;
    private String nomPays;
    Public Collection<Ville> villes;
}
```


madani.a@ucd.ac.ma

Mapping des associations (Rappels)

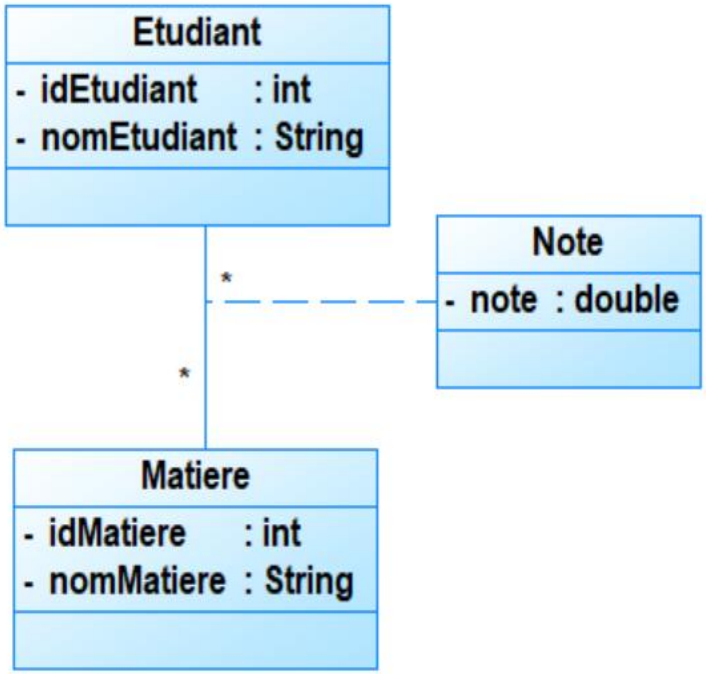



```
public class Etudiant {
    private int idEtudiant;
    private String nomEtudiant;
    private Collection<Matiere> matieres;
}
```

```
public class Matiere {
    private int idMatiere;
    private String nomMatiere;
    private Collection<Etudiant> etudiants;
}
```


madani.a@ucd.ac.ma

Mapping des associations (Rappels)




```
public class Etudiant{
    private int idEtudiant;
    private String nomEtudiant;
    private Collection<Note> notes;
}
```

```
public class Matiere {
    private int idMatiere;
    private String nomMatiere;
    private Collection<Note> notes;
}
```

```
public class Note {
    private int idMatiere;
    private int idEtudiant;
    Private double note;
}
```


madani.a@ucd.ac.ma

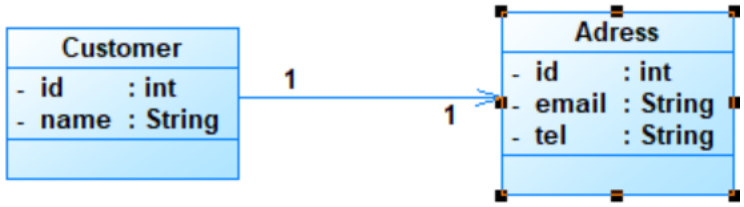
Mapping des associations

A retenir

- Les relations entre entités, telles que définies en JPA peuvent être unidirectionnelles ou bidirectionnelles.
- Dans ce second cas, l'une des deux entités doit être maître et l'autre esclave.
- Dans le cas des relations 1:1 et n:p, on peut choisir le côté maître comme on le souhaite.
- Dans le cas des relations 1:p et n:1, l'entité du côté 1 est l'entité esclave.
- Dans une relation bidirectionnelle, l'entité esclave doit préciser l'attribut **mappedBy** dans l'annotation @.

madani.a@ucd.ac.ma

Relation OneToOne (Unidirectionnelle)




```
@Entity
public class Address {
    @Id @GeneratedValue()
    private int id;
```

```
@Entity
public class Customer{
    @Id @GeneratedValue()
    private int id;
    @OneToOne
    private Address address;
```

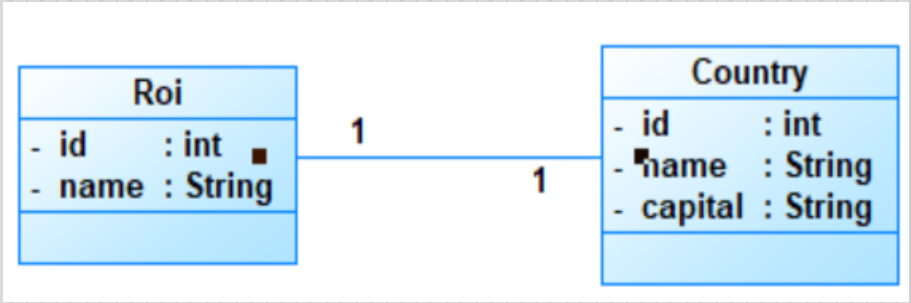

madani.a@ucd.ac.ma

Relation OneToOne (Bidirectionnelle)

- Relation 1 :1, on peut choisir l'une comme esclave, par exemple, la classe Country.
- L'entité esclave doit préciser un champ retour par une annotation @OneToOne et un attribut mappedBy, qui doit référencer le champ qui porte la relation côté maître

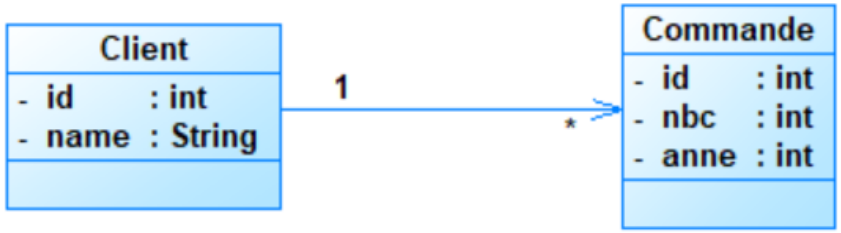
```
@Entity(name="King")
public class Roi {
        @Id @GeneratedValue()
        private int id;
        @OneToOne
        private Country country;
```

```
@Entity
public class Country{
        @Id
        @GeneratedValue()
        private int id;
        @OneToOne (mappedBy = "country")
        private Roi roi;
```



madani.a@ucd.ac.ma

Relation OneToMany (Unidirectionnelle)

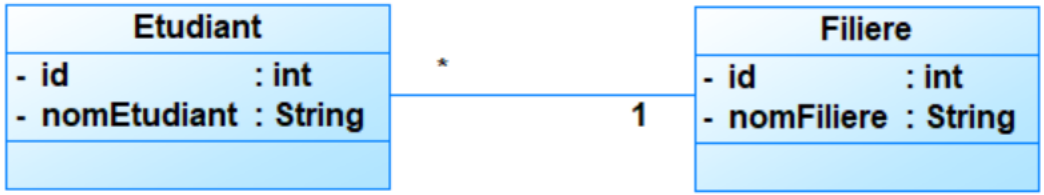



```
@Entity
public class Client {
        @Id @GeneratedValue()
        private int id;
        @OneToMany
        List<Commande> cdes = new ArrayList<Commande>();
```

```
@Entity
public class Commande {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int nbc;
```


madani.a@ucd.ac.ma

Relation OneToMany (Bidirectionnelle)

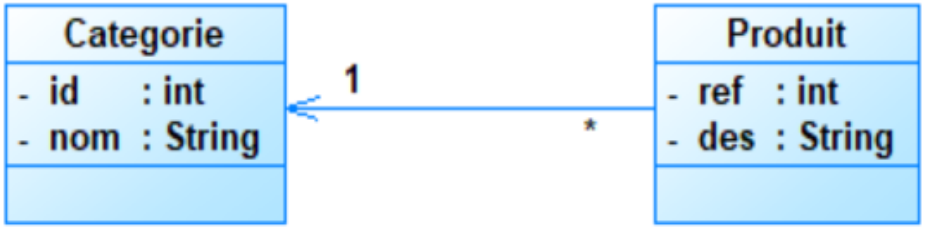



```
@Entity
public class Filiere {
        @Id @GeneratedValue()
        private int codeFiliere;
        @OneToMany(mappedBy = "filiere")
        private Collection<Etudiant> ets = new ArrayList<Etudiant>();
```

```
@Entity
public class Etudiant{
        @Id @GeneratedValue()
        private int codeEtudiant;
        @ManyToOne
        private Filiere filiere;
```


madani.a@ucd.ac.ma

Relation ManyToOne (Unidirectionnelle)




```
@Entity
public class Produit {
        @Id @GeneratedValue()
        private int ref;
        @ManyToOne
        private Categorie cat;
```

```
@Entity
public class Categorie {
        @Id @GeneratedValue()
        private int id;
```


madani.a@ucd.ac.ma

Relation ManyToMany (Unidirectionnelle)


```
@Entity
 public class PR {
        @Id @GeneratedValue()
        private int id;
        @ManyToMany
        private Collection<CD> cds = new ArrayList<CD>();
```

```
@Entity
public class CD {
        @Id @GeneratedValue()
        private int id;
```


madani.a@ucd.ac.ma

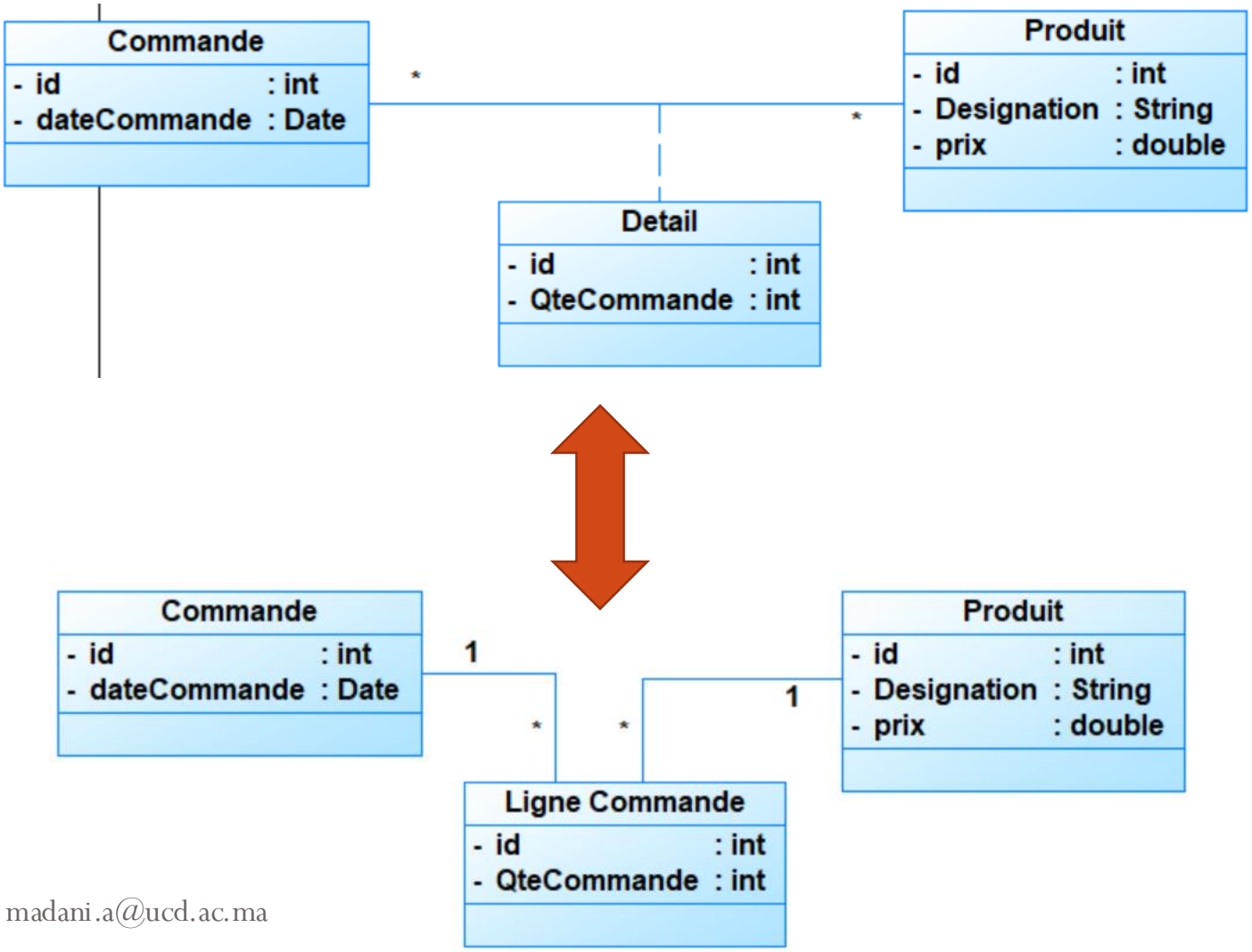
Relation ManyToMany (Bidirectionnelle)


```
@Entity
public class PR1{
        @Id @GeneratedValue()
        private int id;
        @ManyToMany(mappedBy = "prs")
        private Collection<CD1> cds = new ArrayList<CD1>();
```

```
@Entity
public class CD1 {
        @Id @GeneratedValue()
        private int id;
        @ManyToMany
        private Collection<PR1> prs = new ArrayList<PR1>();
```

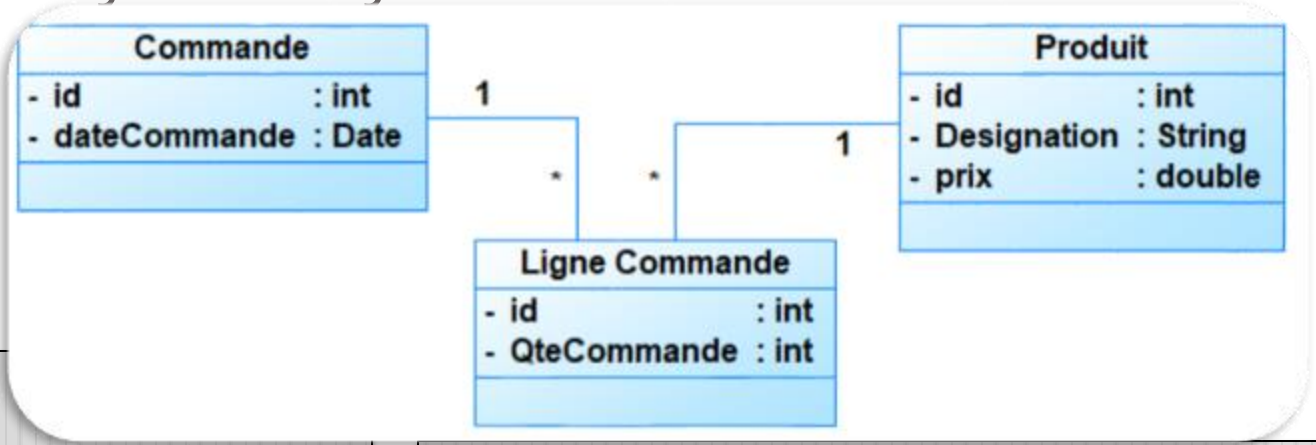

madani.a@ucd.ac.ma

Relation ManyToMany avec Classe association



madani.a@ucd.ac.ma

Relation ManyToMany avec Classe association




```
@Entity
public class LigneCommande{
    @Id @GeneratedValue()
    private int id;
    @ManyToOne
    private Commande commande;
    Private Produit produit;
```

```
@Entity
public class Produit{
    @Id @GeneratedValue()
    private int id;
    @OneToMany(mappedBy="produit");
    private Collection<LigneCommande> lignes;
```

```
@Entity
public class Commande{
    @Id @GeneratedValue()
    private int id;
    @OneToMany(mappedBy="commande");
    private Collection<LigneCommande> lignes;
```


madani.a@ucd.ac.ma

Mapping des associations -Eager VS Lazy-

- Lorsqu'une entité est chargé dans le « persistenceContext » par l'EntityManager, ses liens peuvent être:
  - Immédiatement chargés : « eagerloading »
  - Chargés plus tard, quand l'application va les utiliser « lazy loading »

```
@Entity public class Employee {
   @Id private int id;
   @OneToOne(fetch=FetchType.LAZY)
   private ParkingSpace parkingSpace;
   // ...
}
```


madani.a@ucd.ac.ma

Mapping d’une hiérarchie de classes

madani.a@ucd.ac.ma

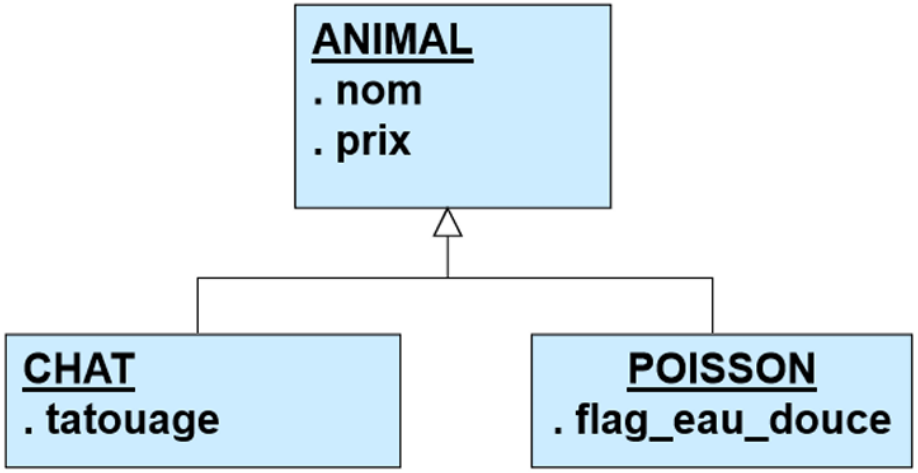
Mapping une hiérarchie de classes

- Trois stratégies d'héritage de base possible sous Hibernate :
  - Une table par hiérarchie de classe (**SINGLE_TABLE**)
  - Une table pour chaque classe concrète (**TABLE_PER_CLASSE**)
  - Une table pour la classe parente et une table pour chaque classe fille (**JOINED_TABLE**)
- Possibilité d'utiliser différentes stratégies de mapping pour différentes branches d'une même hiérarchie

madani.a@ucd.ac.ma

Mapping une hiérarchie de classes

- import javax.persistence.InheritanceType;
- @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
- @Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
- @Inheritance(strategy=InheritanceType. JOINED)
- Strategie par défaut :
  - InheritanceType.SINGLE_TABLE;

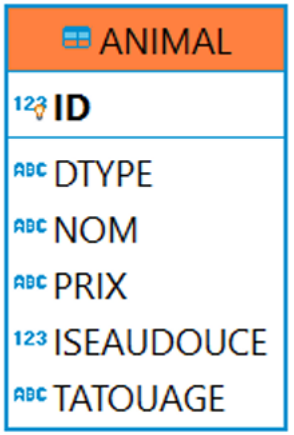


madani.a@ucd.ac.ma

Single Table

- InheritanceType.SINGLE_TABLE
- Toutes les instances d'une hiérarchie sont dans une seule et même table
- La table contient une colonne supplémentaire discriminant
- Le discriminant peut être définit par une annotation

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public class Animal {
...
}
```



madani.a@ucd.ac.ma

Single Table

Avantages

- Bon support pour les associations et les requêtes polymorphiques
  - Pas besoin de jointures

Inconvénients

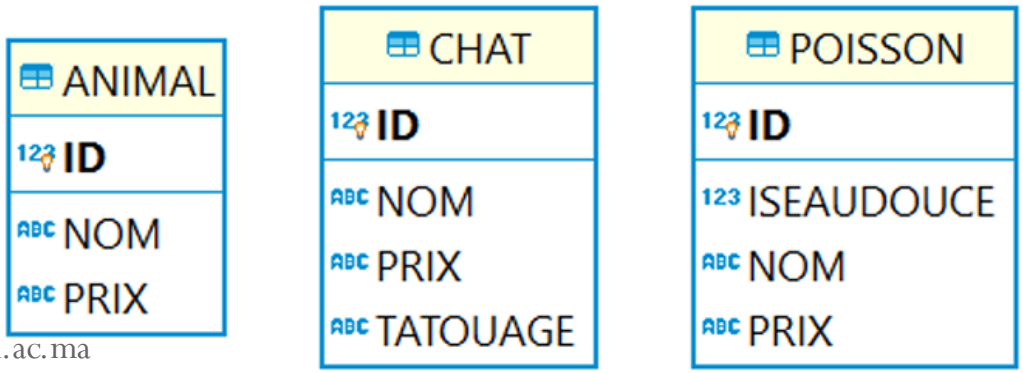
- Les colonnes doivent être nullable
- Peut contenir beaucoup de valeurs null.

madani.a@ucd.ac.ma

Une table par classe concrète

- InheritanceType.TABLE_PER_CLASS
- Chaque classe concrète a sa propre table
- Chaque table contient toutes les propriétés (y compris des superclasses)

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Animal {
...
}
```



madani.a@ucd.ac.ma

Une table par classe concrète

Avantages

- Diminue les valeurs nulles

Inconvénients

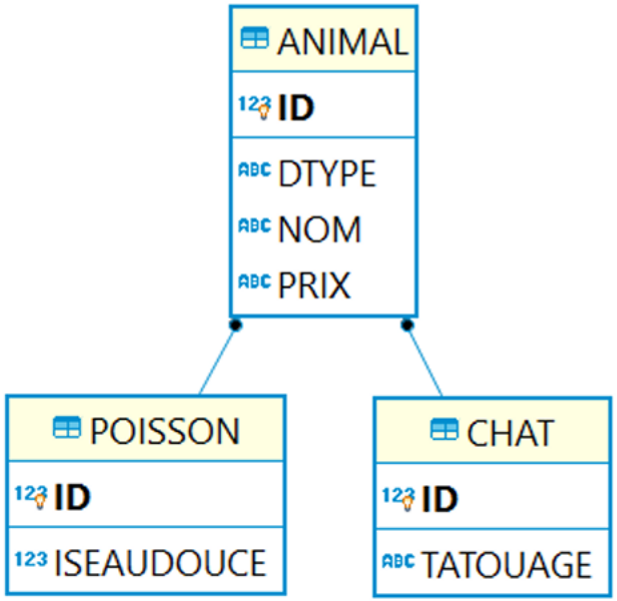
- Mauvais support du polymorphisme
  - Des jointures sur les tables sont requises
- Cette stratégie est optionnel
  - Elle n'est pas supporté par toutes les implémentation de JPA

madani.a@ucd.ac.ma

Strategie JOINED

- InheritanceType.JOINED,
- La classe racine de la hiérarchie a sa propre table
  - Chaque sous-classe a sa table
  - Une table ne contient que les propriétés de sa classe

```
@Entity@Inheritance(strategy=InheritanceType.JOINED)
public class Animal {
...
}
```



madani.a@ucd.ac.ma

Strategie JOINED

Avantages

- Bon support pour les associations et les requêtes polymorphiques
- Pas de valeurs nulles superflues

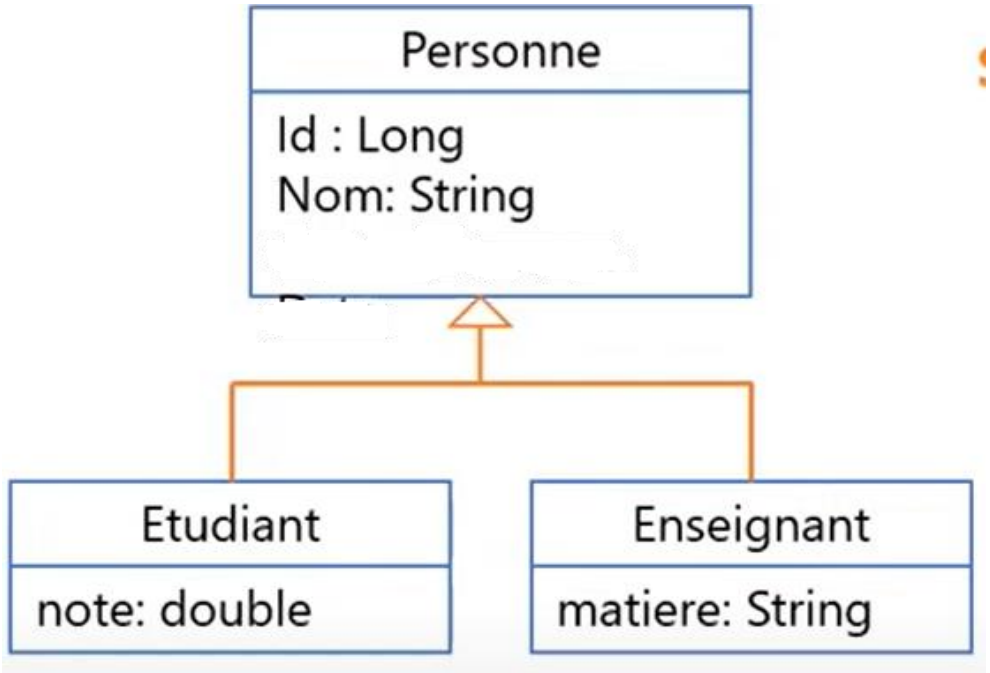
Inconvénients

- Des jointures sont requises pour obtenir un objet
  - Peut entrainer des mauvaises performances
- Les classes peuvent nécessiter une colonne discriminant
  - Dépend de l'implémentation
  - Pas un problème si on laisse la création de table auto

madani.a@ucd.ac.ma

Mapping une hiérarchie de classes -Exemples-

- Les exemples qui suivent montre comment utiliser les différentes stratégies pour mapper une hiérarchie de classes.
- L'exemple que nous traiter est le suivant :



madani.a@ucd.ac.ma

Single Table : Exemple

```
@Entity @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Personne {
@Id @GeneratedValue(strategy = GenerationType.IDENTITY)
private int id; private String nom;
```

```
@Entity @DiscriminatorValue("Student")
public class Etudiant extends Personne {
private double note;
```

```
@Entity @DiscriminatorValue("Teacher")
public class Prof extends Personne {
private double matiere;
```

Person_Type	id	nom	note	matiere
Student	1	Hamdi	12	*NULL*
Teacher	2	Madani	*NULL*	Informatique

madani.a@ucd.ac.ma

Table_Per_Class: Exemple

```
@Entity @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Personne {
@Id @GeneratedValue(strategy = GenerationType.TABLE)
private int id; private String nom;
```

```
@Entity
public class Etudiant extends Personne {
private double note;
```

```
@Entity
public class Prof extends Personne {
private double matiere;
```

id	nom	matiere
2	Madani	Informatique

id	nom	note
1	Hamdi	12

madani.a@ucd.ac.ma

Joined_Table: Exemple

```
@Entity @Inheritance(strategy = InheritanceType.JOINED)
public class Personne {
@Id @GeneratedValue(strategy = GenerationType.TABLE)//IDENTITY
private int id; private String nom;
```

```
@Entity
public class Etudiant extends Personne {
private double note;
```

```
@Entity
public class Prof extends Personne {
private double matiere;
```

id	nom
1	Hamdane
	Madani

note	id
12.75	1

matiere	id
Informatique	2

madani.a@ucd.ac.ma