

# Mapping des Associations

# Plan

---

- Les différents aspects des associations
- Représentation des associations
- Association one-to-one
- Association one-to-many
- Association many-to-one
- Association bi-directionnelle
- Association many-to-many
- Classe Association
- Mapping et collection

# Aspects des associations

---

- Structurel
  - Clé primaire / clé étrangère
  - Cardinalité
  - Collection hibernate et collection Java
- Navigation
  - Lazy-loading
- Cycle de vie
  - Sauvegarde et suppression en cascade
  - Composition

# Représentation des associations

## Généralités

---

- Modèle objet Java
  - Représentation intrinsèque des liens entre objets par des références.
  - Donne le sens de navigation
- Modèle relationnel
  - Clé étrangère
  - Table association
  - Pas de sens de navigation

# Méthodologie

---

## Modélisation :

- Définir les cardinalités (*@OneToOne*, *@OneToMany*, *@ManyToOne*, *@ManyToMany*)
- Définir la navigation : uni-directionnel, bi-directionnel (Attribut *mappedBy*)  
=> 7 cas possibles

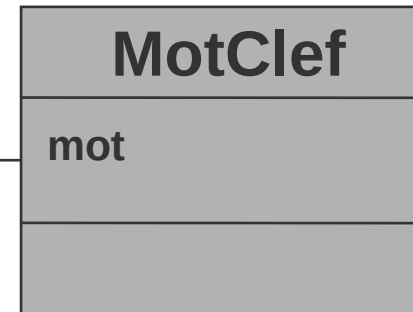
## Schéma physique :

- Laisser faire Hibernate (Le meilleur choix)
- Utiliser *@JoinColumn* et *@JoinTable* pour maîtriser le schéma physique

# Représentation des associations

## Exemple

Modèle  
objet



1..\*

Association

Table TTheme

Table TMotClef

Clé  
étrangère

Modèle  
relationnel

Id	label
1	Aventure
2	Science-fiction
3	Informatique

id	mot_clef	ID theme
1	J2EE	3
2	Ordinateur	3
3	Langage Java	3

Mapping des associations

# Association OneToOne uni.

## Modèles

---



Table TAdherent

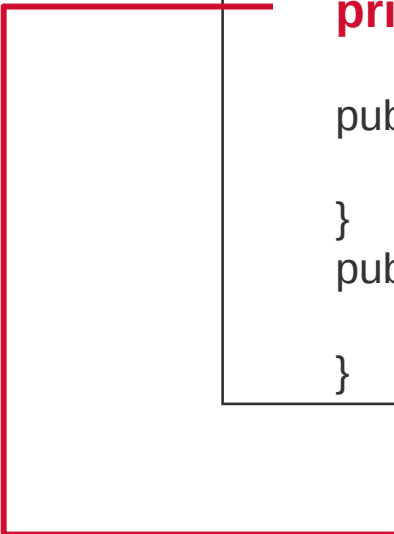
Id	nom	IdAbo
1	Marytin	1
2	Valron	2
3	Frament	3

Table TAbonnement

id	date_abo	prix
1	2001-12-08	20
2	2004-07-20	50
3	2003-03-10	20

# Association OneToOne uni. les classes

---



```
public class Adherent {  
    private Abonnement abonnement;  
  
    public Abonnement getAbonnement() {  
        return abonnement;  
    }  
    public void setAbonnement(Abonnement abonnement) {  
        this.abonnement = abonnement;  
    }  
}
```

```
public class Abonnement {  
    private int id;  
    private Date date;  
    private String formule;  
    private float prix;  
}
```



# Association OneToOne uni. JPA

---

**@Entity**

public class Adherent {

**private Abonnement abonnement;**

**@OneToOne**

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

public Abonnement getAbonnement() {  
 return abonnement;  
}

public void setAbonnement(Abonnement abonnement) {  
 this.abonnement = abonnement;  
}

public class Abonnement {

**@Id**

private int id;  
private Date date;  
private String formule;  
private float prix;

# Association OneToMany - uni. les modèles

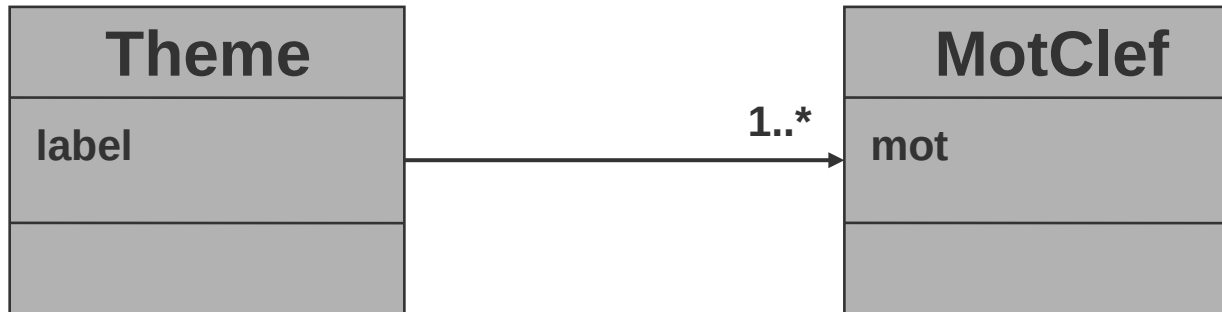


Table TTheme

id	label
1	Aventure
2	SF
3	Informatique


Table Assoc.

idtheme	idnr	id	mot
1	3	1	JavaEE
2	3	2	Software
2	3	3	ORM

Table TMotClef

# Association OneToMany - uni. les classes

```
public class Theme {  
    private Long id;  
    private String label;  
    private Set<MotClef> motclefs = new HashSet<MotClef>();  
  
    public Set<MotClef> getMotclefs() { return motclefs;}  
    public void setMotclefs(Set<MotClef> motclefs) {  
        this.motclefs = motclefs;  
    }  
    public void addMotClef(MotClef mot){  
        motclefs.add(mot);  
    }  
}
```



```
public class MotClef {  
    private Long id;  
    private String mot;  
  
    public MotClef(){}  
  
    public MotClef(String m){  
        mot = m;  
    }  
}
```

# Association OneToMany - uni.

## Table d'association

**@Entity**

```
public class Theme {  
    private Long id;  
    private String label;  
    private Set<MotClef> motclefs = new HashSet<MotClef>();  
    @OneToMany  
    @JoinTable(name="ThemeMot",  
        joinColumns=@JoinColumn(name="IDTHEME"),  
        inverseJoinColumns=@JoinColumn(name="IDMOT"))  
    public Set<MotClef> getMotclefs() { return motclefs;}  
    public void setMotclefs(Set<MotClef> motclefs) {  
        this.motclefs = motclefs;  
    }  
    public void addMotClef(MotClef mot){motclefs.add(mot);}
```

```
public class MotClef {  
    private Long id;  
    private String mot;  
  
    public MotClef(){}  
  
    public MotClef(String m){mot = m ; }
```

# Association OneToMany - uni. Clé étrangère

**@Entity**

```
public class Theme {  
    private Long id;  
    private String label;  
    private Set<MotClef> motclefs = new HashSet<MotClef>();  
    @OneToMany  
    @JoinColumn(name="IDTHEME")  
    public Set<MotClef> getMotclefs() { return motclefs;}  
    public void setMotclefs(Set<MotClef> motclefs) {  
        this.motclefs = motclefs;  
    }  
    public void addMotClef(MotClef mot){motclefs.add(mot);}
```

```
public class MotClef {  
    private Long id;  
    private String mot;  
  
    public MotClef(){}  
  
    public MotClef(String m){mot = m ; }
```

# Association one-to-many l'usage

---

```
em = emf.createEntityManager();  
MotClef m1 = new MotClef();  
session.persist(m1);  
theme = session.get(Theme.class, new Long(1));  
theme.getMotClefs().add(m1);  
em.close()
```

Trace Hibernate

```
Insert into tmotclef(mot_clef,...) values(...)  
Select * from ttheme where id = 1  
Update tmotclef set IDTheme = 1 where ID = ?
```

- Attention ici il faut explicitement rendre le motclef persistant !

# Association many-to-one les modèles

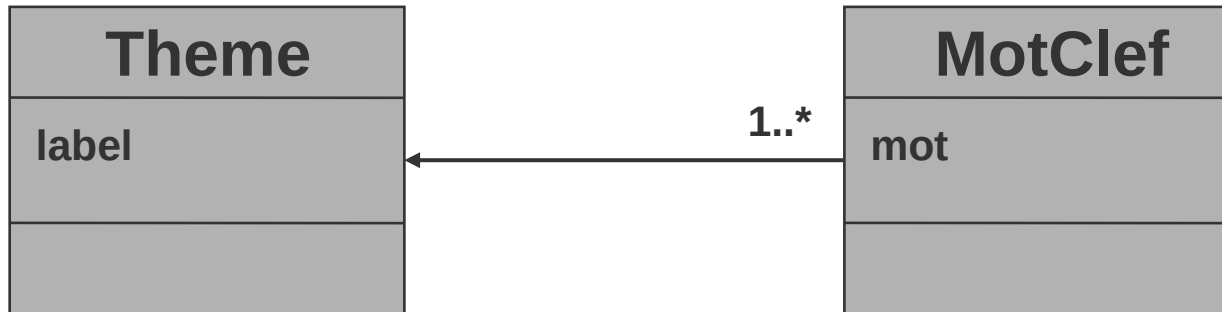


Table TTheme

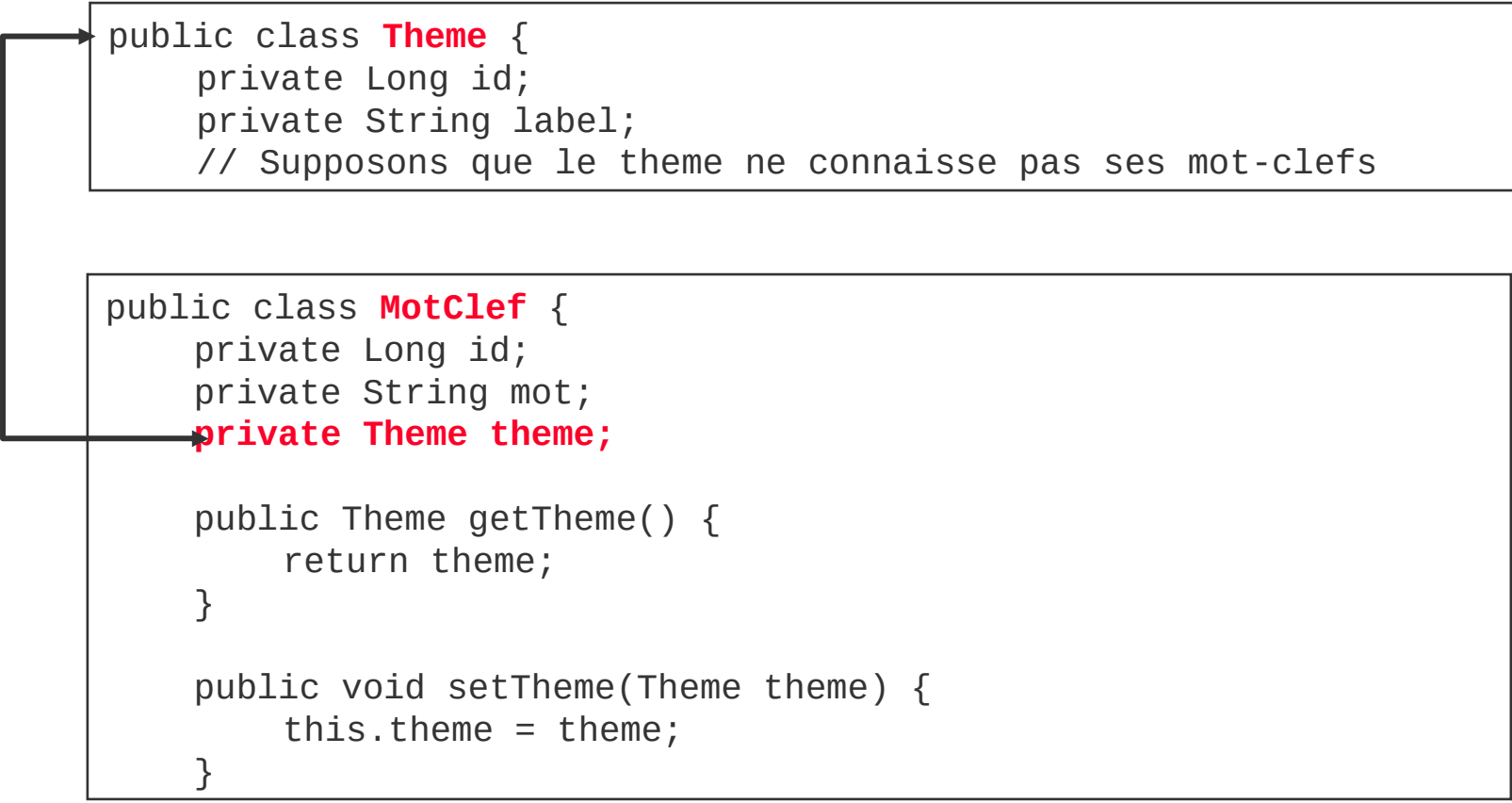
Id	label
1	Aventure
2	Science-fiction
3	Informatique

Table TMotClef

id	mot_clef	IDTheme
1	J2EE	3
2	Ordinateur	3
3	Langage Java	3

# Association many-to-one les classes

---



```
public class Theme {  
    private Long id;  
    private String label;  
    // Supposons que le theme ne connaisse pas ses mot-clefs
```

```
    public class MotClef {  
        private Long id;  
        private String mot;  
        private Theme theme;  
  
        public Theme getTheme() {  
            return theme;  
        }  
  
        public void setTheme(Theme theme) {  
            this.theme = theme;  
        }  
    }
```



# Association many-to-one

## JPA

---

**@Entity**

```
public class Theme {  
    private Long id;  
    private String label;  
    // Supposons que le theme ne connaisse pas ses mot-clefs
```

```
public class MotClef {  
    private Long id;  
    private String mot;  
    private Theme theme;  
    @ManyToOne  
    public Theme getTheme() {  
        return theme;  
    }  
  
    public void setTheme(Theme theme) {  
        this.theme = theme;  
    }  
}
```

# Association many-to-one l'usage

## Contrainte d'intégrité : IDTheme not null

```
public void testCreationMotClef() throws Exception{
    EntityManager em = emf.createEntityManager();

    MotClef m = new MotClef();
    m.setMot("Templier");
    Theme t = (Theme)s.get(Theme.class, new Long(4));
    s.persist(m);

    m.setTheme(t);

    s.persist(m);

    em.close();
}
```

~~1~~

2

# Association bi-directionnelle mise à jour

---

- Il suffit de combiner les deux associations uni-directionnelles.
- Attention à la mise à jour des références !
- Elles doivent être mises à jour des 2 côtés

```
Transaction tx = s.beginTransaction();  
MotClef m = new MotClef();  
m.setMot("Templier");  
Theme t = (Theme)s.get(Theme.class, new Long(4));  
s.persist(m)
```

```
m.setTheme(t);  
t.getMotClefs().add(m);
```

```
tx.commit();  
s.close();
```

```
}
```

# Association bidirectionnelle balise inverse

---

## Mapping Theme

```
<hibernate-mapping package="com.plb.etechno.j12.exemple.metier">
  <class name="Theme" table="TTheme">
    <!-- id et autres propriétés -->

    <set name="motclefs" inverse=" true ">
      <key column="IDTheme"/>
      <one-to-many class="MotClef"/>
    </set>
  </class>
</hibernate-mapping>
```

## Code

```
public class Theme {
    .....
    public void addMotClef(MotClef mot){
        motclefs.add(mot);
        mot.setTheme(this);
    }
}
```

# Association bidirectionnelle

## Propriété *mappedBy*

---

**@Entity**

```
public class Theme {  
    private Long id;  
    private String label ;  
    @OneToMany(mappedBy="theme")  
    private List motClefs;
```

```
public class MotClef {  
    private Long id;  
    private String mot;  
    private Theme theme;  
    @ManyToOne  
    public Theme getTheme() {  
        return theme;  
    }  
  
    public void setTheme(Theme theme) {  
        this.theme = theme;  
    }  
}
```

# Association OneToOne bidirectionnelle

---

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
  <many-to-one name="address" column="addressId"
    unique="true" not-null="true"/>
</class>
<class name="Address">
  <id name="id" column="addressId">
    <generator class="native"/>
  </id>
  <one-to-one name="person" property-ref="address"/>
</class>
```

# Association OneToOne bidirectionnelle - JPA

---

**@Entity**

```
public class Personne {  
    private Long id;  
    private String nom ;  
    @OneToOne(mappedBy="personne")  
    private Adresse adresse;
```

```
public class Adresse {  
    private Long id;  
  
    private Personne personne;  
    @OneToOne  
    public Personne getPersonne() {  
        return personne;  
    }  
  
    public void setPersonne(Personne personne) {  
        this.personne = personne;  
    }  
}
```

# Association OneToMany bidirectionnelle

---

```
<class name="Theme">
  <id name="id" column="id">
    <generator class="native"/>
  </id>
  <set name="mots" inverse="true">
    <key column="IDTheme"/>
    <one-to-many class="MotClef"/>
  </set>
</class>
<class name="MotClef">
  <id name="id" column="id">
    <generator class="native"/>
  </id>
  <many-to-one name="address" column="IDTheme" not-null="true"/>
</class>
```



# Association OneToMany bidirectionnelle - JPA

---

**@Entity**

```
public class Theme {  
    private Long id;  
    private String label ;  
    @OneToMany(mappedBy="theme")  
    private Set<MotClef> mots;
```

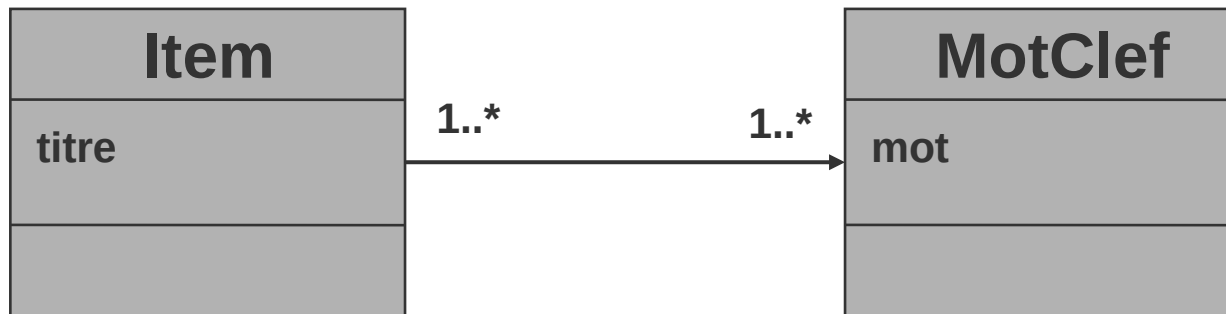
```
public class MotClef {  
    private Long id;  
    private String mot;  
    private Theme theme;  
    @ManyToOne  
    @JoinColumn(name="IDTheme") // Pas du côté du mappedBy  
    public Theme getTheme() {  
        return theme;  
    }  
  
    public void setTheme(Theme theme) {  
        this.theme = theme;  
    }  
}
```

# Association many-to-many

## Modèle objet

---

- Association sans attributs ou comportement
- On navigue d'item vers MotClef



# Association many-to-many

## Modèle relationnel

Table  
Titem

id	titre	texte	duree	nbDvd	chanteur	auteur	isbn	period	type
01	Troie	Fil.	2h20	2	null	null	null	null	DvD
02	Supe	Dis.	1h04	null	SuperT.	null	null	null	CD
03	Tele	Je..	null	null	null	null	null	hebdo	Mag

Table  
d'association !

Table Titem\_motclef

IdItem	IDMotclef
1	1
4	3

Table TMotClef

id	mot_clef	IDTheme
1	J2EE	3
2	Ordinateur	3
3	Langage Java	3

Mapping des associations

# Association many-to-many

## Les Classes

---

```
public class Item {
    private Long id;
    private String titre;
    private String texte;
    private Set<Exemplaire> exemplaires = new HashSet<Exemplaire>();

    private Set<MotClef> motclefs = new HashSet<MotClef>();

    public Set<MotClef> getMotclefs() {
        return motclefs;
    }
    public void setMotclefs(Set<MotClef> motclefs) {
        this.motclefs = motclefs;
    }
    public void addMotClef(MotClef m){
        motclefs.add(m);
    }

    public void removeMotClef(MotClef m){
        motclefs.remove(m);
    }
}
```

# Association many-to-many

## Le mapping

---

```
<hibernate-mapping package="com.tsystems.etechno.j12.exemple.metier">
  <class name="Item" table="TItem" discriminator-value="IT">
    <id name="id" column="ID">
      <generator class="native"/>
    </id>
    <!-- autres propriétés -->

    <set name="motclefs" table="titem_motclef" >
      <key column="IDItem"/>
      <many-to-many class="MotClef" column="IDMotClef"/>
    </set>

  </class>
</hibernate-mapping>
```




Table association

# Association many-to-many

## Les Classes

---

### **@Entity**

```
public class Item {  
    private Long id;  
    private String titre;  
    private String texte;  
    private Set<Exemplaire> exemplaires = new HashSet<Exemplaire>();
```

```
    private Set<MotClef> motclefs = new HashSet<MotClef>();
```

### **@ManyToMany**

```
    public Set<MotClef> getMotclefs() {  
        return motclefs;  
    }  
    public void setMotclefs(Set<MotClef> motclefs) {  
        this.motclefs = motclefs;  
    }  
    public void addMotClef(MotClef m){  
        motclefs.add(m);  
    }  
  
    public void removeMotClef(MotClef m){  
        motclefs.remove(m);  
    }  
}
```

# Association many-to-many Usage

---

```
public void testConsultationLMotClefItem() throws Exception{
    Session s = DBHelper.getFactory().openSession();
    Transaction tx = s.beginTransaction();
    Query hqlQuery = s.createQuery("from Item");
    List<Item> lItem = (List<Item>)hqlQuery.list();
    for (Item t : lItem) {
        System.out.print(t.getId() + ") " + t.getTitre() + " nb Ex -> "
            + t.getExemplaires().size() + "["");
        for(MotClef m : t.getMotclefs()){
            System.out.print(m.getMot() + " ");
        }
        System.out.println("]");
    }

    tx.commit();
    s.close();
}
```

# Collections

---

Les *Collection*, *List*, *Map* et *Set* peuvent être mappées via les annotations *@OneToMany* ou *@ManyToMany*

Si on utilise *hbm*, il faut utiliser la balise adéquate (*<set>*, *<list>*, *<map>*, *<bag>*, *<array>*, *<primitive-array>*)

Lorsque la collection est persistée ou chargée, elle est remplacée par une collection de type Hibernate qui respecte l'interface initiale



# Collections Hibernate

---

```
Cat cat = new DomesticCat();
Cat kitten = new DomesticCat();
....
Set kittens = new HashSet();
kittens.add(kitten);
cat.setKittens(kittens);
session.persist(cat);
kittens = cat.getKittens(); // Okay,
(HashSet) cat.getKittens(); // Error!
```

# List

---

Les *List* peuvent être mappées de 2 façons :

Comme liste ordonnée et l'ordre n'est pas matérialisé dans une colonne de la base de données : annotation **@OrderBy** ou attribut **order-by**

Comme liste indexée par une colonne de la base de donnée : annotation **@OrderColumn** ou balise **<list-index>**

# Exemple liste ordonnée

---

```
@Entity
public class Customer {
    @Id @GeneratedValue public Integer getId()
        { return id; }
    public void setId(Integer id) { this.id = id; }
    private Integer id ;

    @OneToMany(mappedBy="customer")
    @OrderBy("number")
    public List<Order> getOrders() { return orders; }
    public void setOrders(List<Order> orders)
        { this.orders = orders; }
    private List<Order> orders;
}
```

# Exemple List indexée

---

```
@Entity
public class Customer {
    @Id @GeneratedValue public Integer getId()
        { return id; }
    public void setId(Integer id) { this.id = id; }
    private Integer id ;

    @OneToMany(mappedBy="customer")
    @OrderColumn(name="orders_index")
    public List<Order> getOrders() { return orders; }
    public void setOrders(List<Order> orders)
        { this.orders = orders; }
    private List<Order> orders;
}
```

# Map

---

Lors du mapping d'une Map, la valeur de la clé peut provenir de :

Une propriété de l'entité associée : annotation **@MapKey** ou **<map-key>** avec sous-élément **<node>**

D'une colonne de la base : annotation **@MapKeyColumn** ou **<map-key>** avec sous-élément **<column>**

# Exemple *MapKey*

---

```
@Entity
public class Customer {
    @Id @GeneratedValue public Integer getId()
        { return id; }
    public void setId(Integer id) { this.id = id; }
    private Integer id ;
```

```
@OneToMany(mappedBy="customer")
@MapKey(name="number")
public Map<String,Order> getOrders() { return
    orders; }
public void setOrders(Map<String,Order> order)
    { this.orders = orders; }
private Map<String,Order> orders;
}
```

# Exemple *MapKeyColumn*

---

```
@Entity
public class Customer {
    @Id @GeneratedValue public Integer getId()
        { return id; }
    public void setId(Integer id) { this.id = id; }
    private Integer id ;

    @OneToMany @JoinTable(name="Cust_Order")
    @MapKeyColumn(name="orders_number")
    public Map<String,Order> getOrders() { return
        orders; }
    public void setOrders(Map<String,Order> orders) {
        this.orders = orders; }
    private Map<String,Order> orders;
}
```

# Mapping de Collection

## <set> et <bag>

```
<set name="motclefs" >  
  <key column="IDTheme"/>  
  <one-to-many class="MotClef"/>  
</set>
```

```
<bag name="motclefs">  
  <key column="IDTheme"/>  
  <one-to-many class="MotClef"/>  
</bag>
```

```
Set motclefs = new HashSet<MotClef>();
```

```
[ MotClef@76, MotClef@43 ]
```

{ Non ordonné a priori }

```
List motclefs = new ArrayList<MotClef>();
```

```
[ MotClef@43, MotClef@76 ]
```

{ Ordonné suivant le chargement }



# Mapping de Collection

## <list> et <map>

### Tag Hibernate

```
<list name="motclefs" >  
  <key column="Idtheme"/>  
  <list-index column="id"/>  
  <one-to-many class="MotClef"/>  
</list>
```

```
<map name="motclefs">  
  <key column="IDTheme"/>  
  <map-key column="mot_clef"  
    type="string« />  
  <one-to-many class="MotClef"/>  
</map>
```

### Collection Java

```
List motclefs = new ArrayList<MotClef>();
```

```
[ null,null,..., null,fantôme, gothique ]
```

```
Map<String,MotClef> motclefs =  
  new HashMap<String,MotClef>();
```

```
[ {« fantome », Motclef@43},  
  {« gothique »,MotClef@76} ]
```

# Collections de type basique

---

Une collection de types basiques (ou d'objets embarqués) peut être mappée via l'annotation **@ElementCollection** ou **<element>**

# Exemple Collection Type basique

---

```
@Entity
public class User {
    [...]
    public String getLastName() { ...}
    @ElementCollection
    @CollectionTable(name="Nicknames",
        joinColumns=@JoinColumn(name="user_id"))
    @Column(name="nickname")
    public Set<String> getNicknames() { ... }
}
```

# SortedMap, SortedSet

---

Hibernate supporte le mapping vers des collections triées via l'annotation **@Sort** ou l'attribut **sort**

Il faut alors préciser si l'on désire utiliser une classe *Comparator* ou le tri naturel

# Exemple SortedSet

---

```
@OneToMany(cascade=CascadeType.ALL,  
    fetch=FetchType.EAGER)  
@JoinColumn(name="CUST_ID")  
@Sort(type = SortType.COMPARATOR, comparator =  
    TicketComparator.class)  
public SortedSet<Ticket> getTickets() {  
    return tickets;  
}
```

# Exercice

---

- Exercice 6 : Mapping des associations