Hibernate

Le mapping

plan

- Types d'éléments mappés
- Mapping des collections
- Mapping des associations
- Mapping de l'héritage

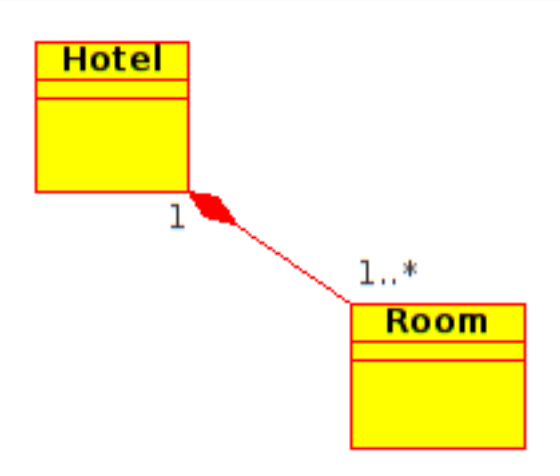
Types d'éléments mappés

Types d'éléments mappés

- A ce stade, il devient important de faire la distinction entre les 2 types d'éléments pouvant être mappés dans Hibernate :
  - Entity :Toute classe Java persistée par Hibernate qui possède:
    - Son propre cycle de vie
    - Une identité en base (clé primaire)
  - Value-type : Toute classe Java persistée par Hibernate qui :
    - Est associée à une et une seule « Entity » (pas de références partagées)
    - Ne dispose pas d'identité en base, son cycle de vie est dépendant de l'« Entity » à laquelle elle est associée (est supprimé si l'Entity associée est supprimée).

Types d'éléments mappés

- Exemples
  - Entity : La classe Agent que l'on a mappée au précédent TP
  - Value-type : Au sens UML c'est une composition :
    - Une chambre ne peut exister que si elle est associée à un Hôtel
    - Il est impossible d'associer plusieurs hôtels à la même chambre
    - Si l'Hôtel est supprimé, toutes les chambres associées le sont également



Mapping des collections

Mapping des collections

- Hibernate supporte la plupart des collections Java :

Java	Hibernate mapping
java.util.Collection	<bag> ou <idbag>
java.util.Set	<set>
java.util.SortedSet	<set sort="..." >
java.util.List	<list>
java.util.Map	<map>
java.util.SortedMap	<map sort="..." >
Object[ ]	<array>
Int[ ], float[ ], double[ ], long[ ], byte[ ], ...	<primitive-array>

Mapping des collections

- Dans tous les cas, il est recommandé par Hibernate d'instancier les attributs qui sont des collections le plus tôt possible (dès la déclaration de l'attribut) :

```
private List<Car> cars = new ArrayList<Car>();
private Set<Driver> drivers = new HashSet<Driver>();
```

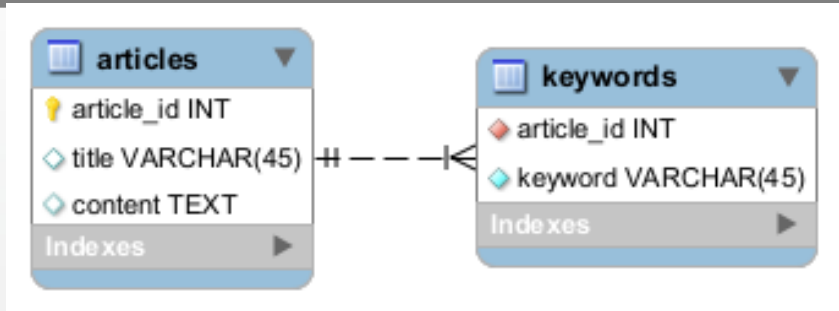
Mapping des collections - Set

- Rappels l'interface java.util.Set
  - Collection non-ordonnée
  - N'autorise pas les doublons

Mapping des collections - Set

- Mapping de l'interface java.util.Set
  - <key> spécifie le nom de la colonne clé étrangère de la table 'keywords' qui référence la table contenant la collection, 'articles'.
  - <element> indique que la collection contient des Value-Type de type String.

```
<set name="keywords" table="keywords">
    <key column="article_id"/>
    <element column="keyword" type="string"/>
</set>
```



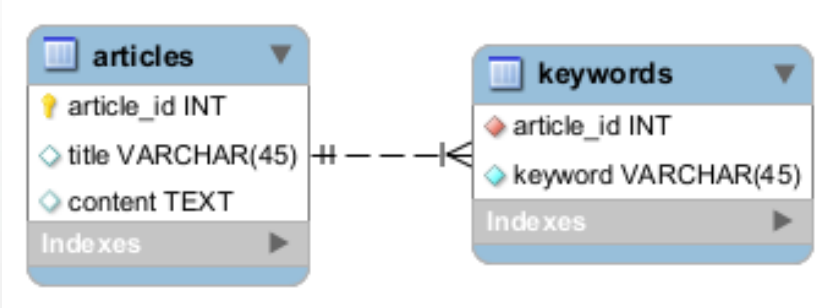
Mapping des collections - Collection

- Rappels l'interface java.util.Collection
  - Collection non-ordonnée
  - Autorise les doublons

Mapping des collections - Collection

- Mapping de l'interface java.util.Collection
  - Avec <bag>, le mapping de Collection est quasiment identique à celui de Set.
  - Contrairement au Set, le même 'keyword' peut apparaître plusieurs fois dans la liste.

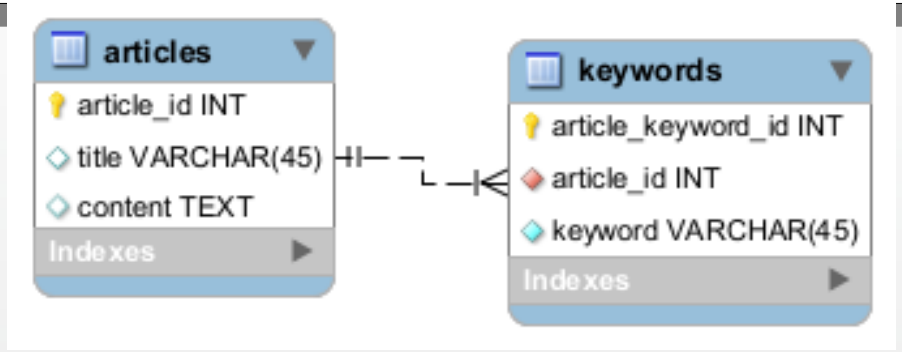
```
<bag name="keywords" table="keywords">
        <key column="article_id" />
        <element type="string" column="keyword" not-null="true" />
</bag>
```



Mapping des collections - Collection

- Mapping de l'interface java.util.Collection
  - <idbag>, permet de rajouter une notion de « clé artificielle » avec <collection-id> afin d'identifier de manière unique les doublons, côté base de données.
  - Attention, le generator 'native' n'est pas implémenté pour ce cas particulier

```
<idbag name="keywords" table="keywords">
        <collection-id type="long" column="article_keyword_id">
           <generator class="sequence" />
        </collection-id>
        <key column="article_id" />
        <element type="string" column="keyword" not-null="true" />
</idbag>
```



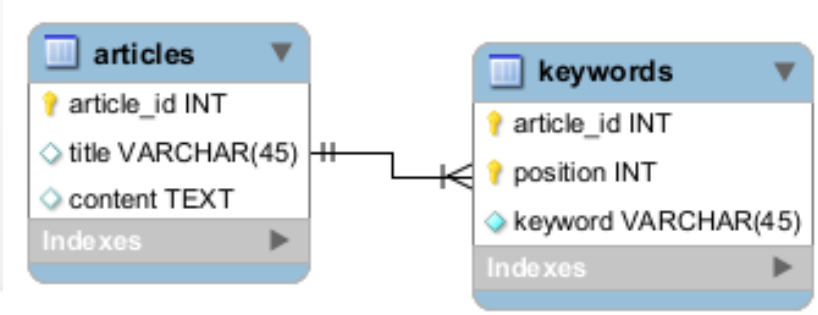
Mapping des collections - List

- Rappels l'interface java.util.List
  - Collection ordonnée
  - Autorise les doublons

Mapping des collections - List

- Mapping de l'interface java.util.List
  - <list>, permet le mapping de List
  - <list-index> est la colonne chargée de stocker la position de l'élément dans la liste.
  - Une clé composée sera établie avec l'index de liste et la clé étrangère vers le conteneur de la collection

```
<list name="keywords" table="keywords">
    <key column="article_id"/>
    <list-index column="position" />
    <element column="keyword" type="string"/>
</list>
```



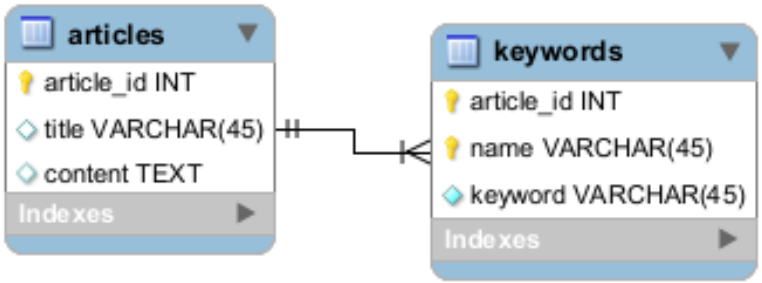
Mapping des collections - Map

- Rappels l'interface java.util.Map
  - Collection non-ordonnée
  - Collection de type clé/valeur (la valeur est indexée par la clé)

Mapping des collections - Map

- Mapping de l'interface java.util.Map
  - <map>, quasiment similaire à liste
  - <map-key> est la colonne chargée de stocker la clé de l'élément dans la liste.
  - Une clé composée sera établie avec la clé de la Map et la clé étrangère vers le conteneur de la collection

```
<map name="keywords" table="keywords">
   <key column="article_id"/>
   <map-key column="name" type="string" />
   <element column="keyword" type="string"/>
</map>
```



Mapping des collections - Tableaux

- <array> permet le mapping d'un tableau d'objet, la configuration est quasiment identique à celle de List avec <list-index> qui est conservé.

```
<array name="keywords" table="keywords">
   <key column="article_id"/>
   <list-index column="position" />
   <element column="keyword" type="string"/>
</array>
```

- De la même manière, <primitive-array> permet le mapping d'un tableau de type de base Java

```
<primitive-array name="numbers" table="numbers">
        <key column="person_id" />
        <list-index column="position" />
        <element type="int" column="number" not-null="true" />
</primitive-array>
```

Mapping des collections - Tableaux

- Il est fortement recommandé par Hibernate d'éviter les tableaux.
- Hibernate met en oeuvre plusieurs mécanismes d' optimisation qui améliorent les performances lors de la persistance des collections. Ces mécanismes ne peuvent être utilisés avec les tableaux.

Mapping des collections - ordonnancement

- Les interfaces java.util.SortedMap et java.util.SortedSet
  - Vont être ordonnées côté Java grâce à l'attribut « sort » qui peut valoir :
    - Unsorted : pas d'ordonnancement
    - Natural : le compareTo() par défaut de l'élément est utilisé
    - Le nom d'une classe implémentant java.util.Comparator

```
<map name="keywords" table="keywords" sort="natural" >
   <key column="article_id"/>
   <map-key column="name" type="string" />
   <element column="keyword" type="string"/>
</map>
```

Mapping des collections - ordonnancement

- Il est également possible d'ordonnancer les collections côté base, lors de la lecture des données, grâce à l'attribut « order-by »
- Il contiendra directement l'instruction SQL qui suit la clause « order by »

```
<set name="keywords" table="keywords"
        order-by="lower(keyword) asc">
    <key column="article_id"/>
    <element column="keyword" type="string"/>
</set>
```

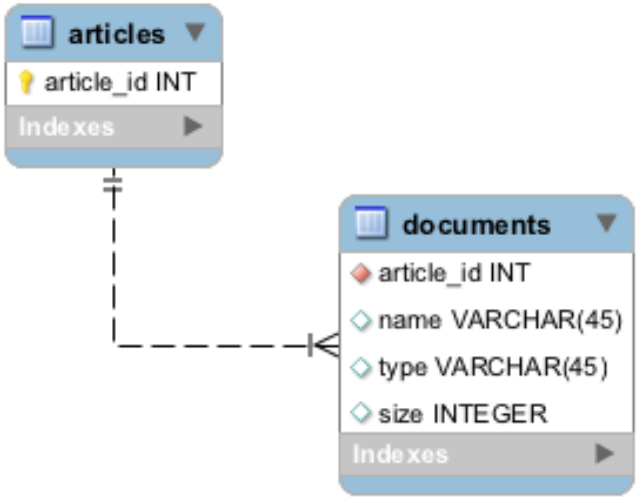
A vous de jouer

- TP 4

Mapping des collections

- Jusqu'à présent, les éléments de nos collections ont été uniquement constitués « Value-Type » sur les types Hibernate de base, globalement les nombres et chaînes de caractères.
- L'utilisation de <composite-element> à la place d' <element> nous permet de travailler toujours en « Value-Type » mais avec des objets Java complets.

```
<set name="documents" table="documents">
    <key column="article_id"/>
    <composite-element class="com.jnesis...Document">
            <property name="name" not-null="true" />
            <property name="type" not-null="true" />
            <property name="size" />
    </composite-element>
</set>
```



A vous de jouer

- TP 5

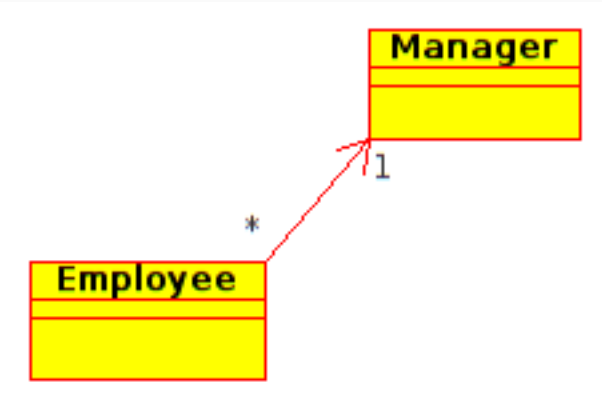
Mapping des associations

Mapping des associations

- Association désigne ici une **relation entre deux Entity** (et donc une relation qui ne constitue pas une composition au sens UML)
- Cette association est qualifiée par son **sens** :
  - Unidirectionnelle
  - Bidirectionnelle
- Sa **cardinalité** :
  - 1 à 1
  - Plusieurs à 1
  - 1 à plusieurs
  - Plusieurs à plusieurs

Mapping des associations unidirectionnelles

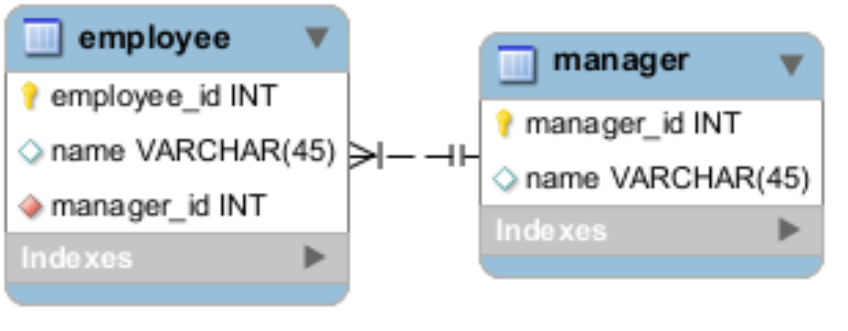
- Many-to-one (plusieurs à 1) est la plus courante des associations unidirectionnelles
- Lorsqu'une instance dispose d'un attribut en référençant une autre et que cette référence n'est pas exclusive
- Dans l'exemple ci-dessous :
  - Employee contient une référence vers Manager.
  - Plusieurs instances d'Employee peuvent référencer la même instance de Manager.
  - Manager n'a aucune connaissance d'Employee (association unidirectionnelle)



Mapping des associations unidirectionnelles

- Le mapping de plusieurs à 1 avec clé étrangère

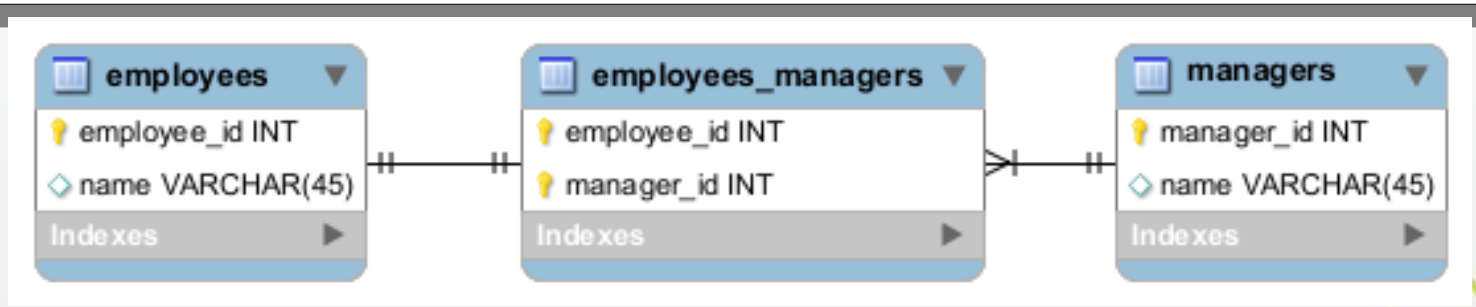
```
<many-to-one name="manager" column="manager_id"  />
```



Mapping des associations unidirectionnelles

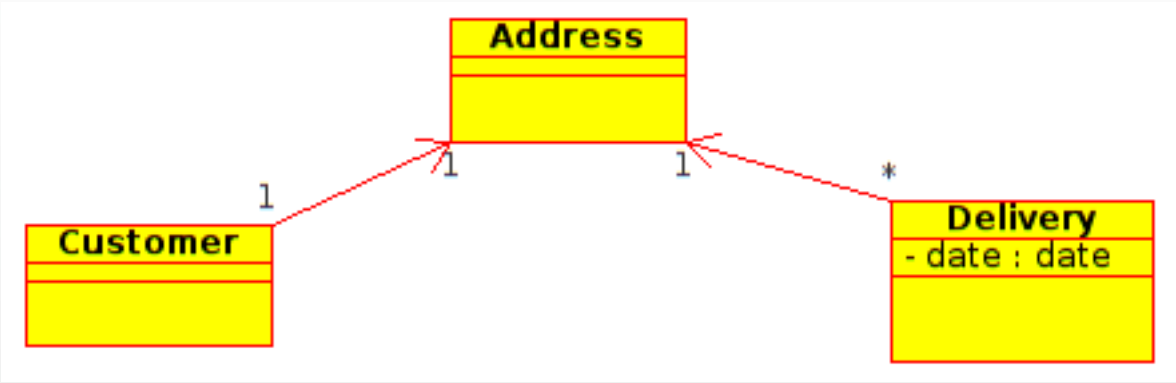
- Association plusieurs à 1 avec table de jointure
  - Une table de jointure est ajoutée avec une contrainte d'unicité pour le côté 1 de l'association
  - Cette version de mapping est à preferer dès lors que l'association est optionnelle (plusieurs à 0..1)

```
<class name="com.jnesis....Employee" table="employees">
      [...]
      <join table="employees_managers" optional="true">
         <key column="employee_id" unique="true"/>
         <many-to-one name="manager"
            column="manager_id" not-null="true"  />
       </join>
</class>
```



Mapping des associations unidirectionnelles

- One-to-one (1 à 1)
- Lorsqu'une instance dispose d'un attribut en référençant une autre et que cette référence est exclusive
- Dans l'exemple ci-dessous :
  - Customer contient une référence vers Address.
  - Plusieurs instances de Customer ne peuvent pas référencer la même instance d'Address (par contre rien n'empêche une autre classe de référencer l'instance d'Address)
  - Address n'a aucune connaissance de Customer (association unidirectionnelle)



Mapping des associations unidirectionnelles

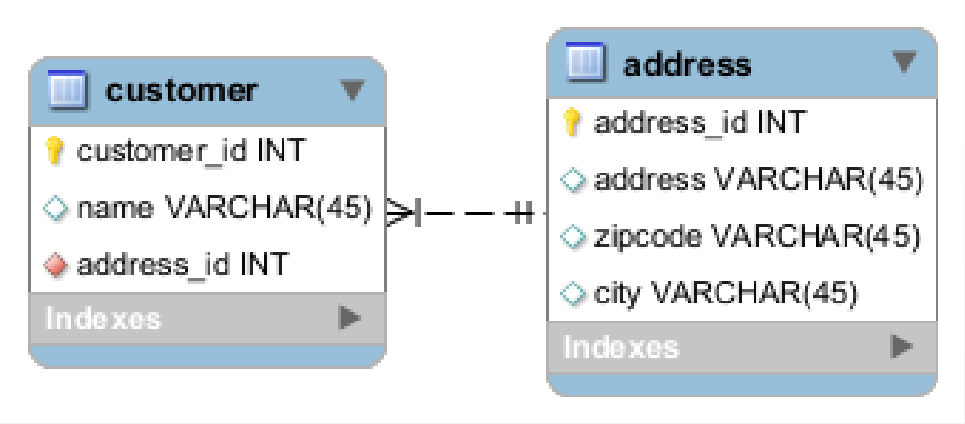
- Il existe deux manières de mapper les associations 1 à 1
  - Par clé étrangère (avec une contrainte d'unicité)
  - Par clé primaire partagée

Mapping des associations unidirectionnelles

Mapping 1 à 1 avec clé étrangère

- Il s'agit simplement d'un mapping many-to-one auquel on rajoute la contrainte d'unicité

```
<many-to-one name="address" column="address_id"  unique="true" />
```



Mapping des associations unidirectionnelles

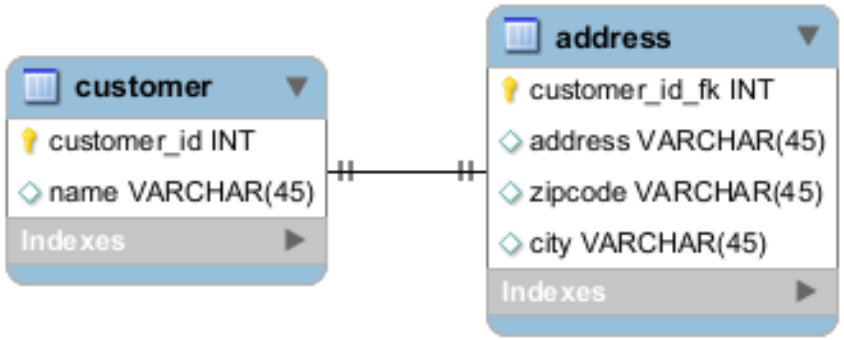
- Mapping 1 à 1 avec clé primaire partagée
  - L'attribut **constrained="true"** viendra garantir que l'attribut customer contiendra un identifiant de customer valide.
  - Un type de **générateur d'identifiant spécifique (foreign)** doit être créé pour que la clé primaire d'Address soit identique à la clé primaire du Customer référencé.

```
<class name="Address">
   <id name="id" column="customer_id">
      <generator class="foreign">
         <param name="property">customer</param>
      </generator>
   </id>
   <one-to-one name="customer" constrained="true"/>
</class>
```

Mapping des associations unidirectionnelles

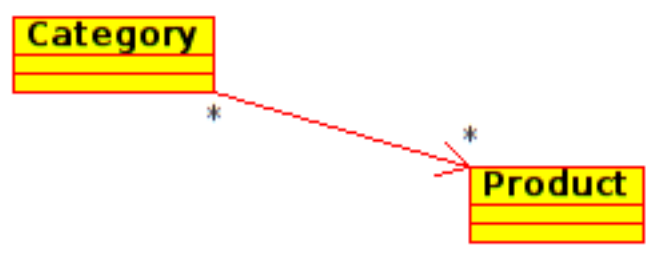
- Mapping 1 à 1 avec clé primaire partagée
  - Notez que pour que ce type de mapping fonctionne, il faut créer une relation bidirectionnelle au niveau objet. (car la valeur de la clé primaire d'Address est lue grâce à l'attribut « customer » contenu dans Address)
  - On ajoute donc une référence vers Customer depuis Address

```
public class Address {
      private Customer customer;
}
```



Mapping des associations unidirectionnelles

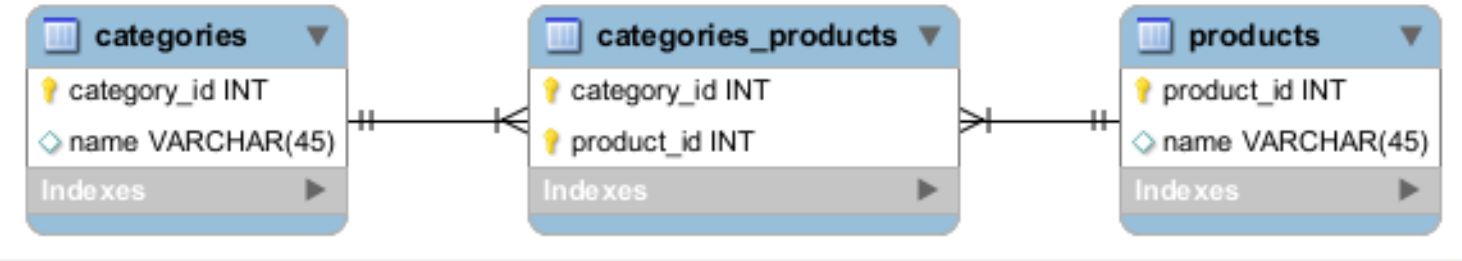
- Association plusieurs à plusieurs
  - Une classe contient une liste d'instances d'une autre classe (Category dispose d'une liste d'instances de Product)
  - Les instances contenues peuvent être partagées entre plusieurs conteneurs (plusieurs instances de Category peuvent référencer la même instance de Product)



Mapping des associations unidirectionnelles

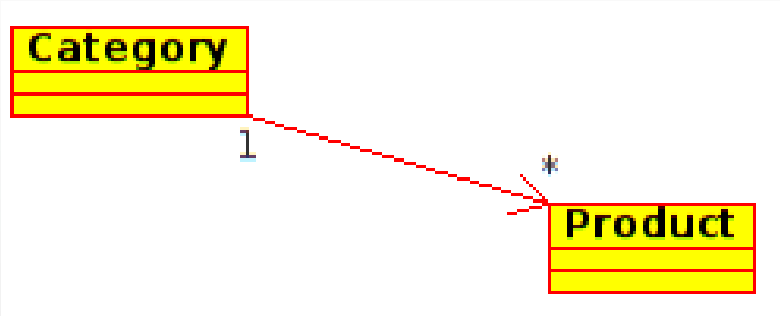
- Mapping plusieurs à plusieurs
  - Une table de jointure intermédiaire sera créée en base

```
<class name="com.jnesis....Category" table="categories">
        <id name="id" column="category_id">
            <generator class="native" />
        </id>
        <set name="products" table="categories_products">
            <key column="category_id" />
            <many-to-many class="com.jnesis....Product "
                                column="product_id" />
        </set>
</class>
```



Mapping des associations unidirectionnelles

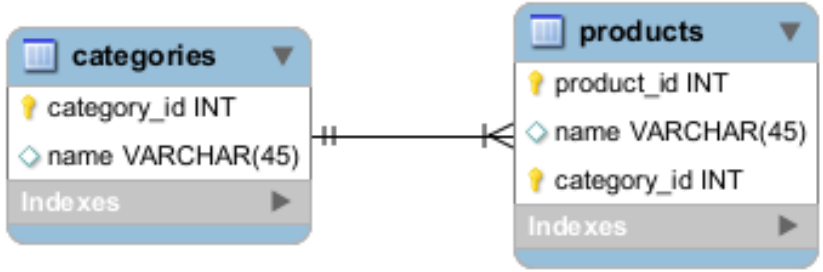
- Association 1 à plusieurs
  - Une classe contient une liste d'instances d'une autre classe (Category dispose d'une liste d'instances de Product)
  - Les instances contenues ne peuvent être liées qu'à un seul conteneur (une seule instance de Category par Product)
  - Il existe deux manières de mapper ces associations
    - Par clé étrangère
    - Avec une table de jointures



Mapping des associations unidirectionnelles

- Association 1 à plusieurs avec clé étrangère
  - Chaque élément contenu dans la liste dispose alors d'une clé étrangère vers l'élément conteneur
  - Ce type de mapping est déconseillé par Hibernate

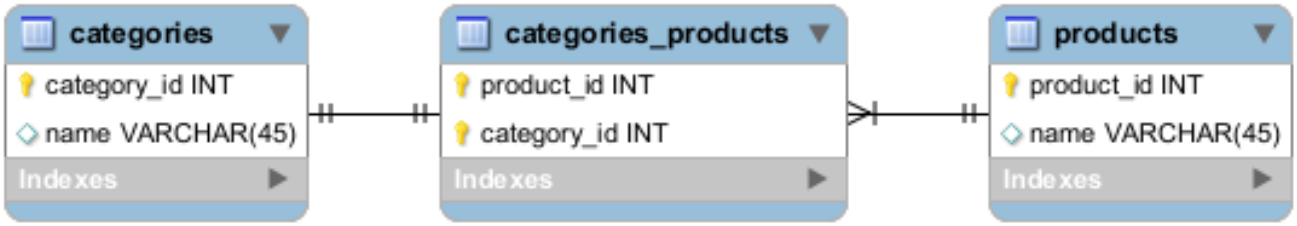
```
<class name="com.jnesis....Category" table="categories">
        <id name="id" column="category_id">
            <generator class="native" />
        </id>
        <set name="products">
            <key column="category_id" not-null="true" />
            <one-to-many class="com.jnesis....Product" />
        </set>
</class>
```



Mapping des associations unidirectionnelles

- Association 1 à plusieurs avec table de jointure
  - Il s'agit en fait d'un mapping plusieurs à plusieurs auquel on rajoute une contrainte d'unicité sur la colonne de la table de jointure représentant le côté 1 de l'association

```
<class name="com.jnesis....Category" table="categories">
        <id name="id" column="category_id">
            <generator class="native" />
        </id>
        <set name="products" table="categories_products">
            <key column="category_id" unique="true" />
            <many-to-many class="com.jnesis....Product    "
                                column="product_id" />
        </set>
</class>
```



A vous de jouer

- TP 6

Mapping des associations bidirectionnelles

- La constitution d'associations bidirectionnelles s'effectue simplement en combinant les mappings de chaque côté de l'association

Association	Implémentation	Côté 1	Côté 2
1 à n	Clé étrangère	<one-to-many>	<many-to-one>
1 à n	Table jointure	<many-to-many unique="true">	<join> avec <many-to-one>
1 à 1	Clé partagée	<one-to-one>	<one-to-one>
1 à 1	Clé étrangère	<many-to-one unique="true">	<one-to-one>
n à n	Table jointure	<many-to-many>	<many-to-many>

Mapping des associations bidirectionnelles

- C'est au développeur de s'assurer de la bidirectionnalité des associations au niveau des instances Java
- Généralement l'un des deux côté sera choisi pour gérér l'aspect bidirectionnel de l'association

```
public class Category {
	[...]
	public void addProduct(Product product)
	{
		this.products.add(product);
		product.setCategory(this);
	}
}
```

Mapping des associations bidirectionnelles

- Les associations bidirectionnelles entraînent un **phénomène de double mise à jour des données**
  - En effet lorsque l'association Product-Category est mise à jour :
    - Une requête d'update sera executée pour l'ajout d'un produit à la categorie
    - Une autre requête d'update sera executée pour l'ajout de la categorie au produit
    - Les deux requêtes effectuent la même mise à jour !

Mapping des associations bidirectionnelles

- Pour éviter cette double mise à jour, on définit, au niveau du mapping, quel est le côté de l'association qui ne sera pas persisté par Hibernate
  - Avec « inverse="true" » pour les collections ou la directive join
  - Avec « insert="false" update="false" » sur un many-to-one

```
<class name="com.jnesis....Category" table="categories">
        <id name="id" column="category_id">
            <generator class="native" />
        </id>
        <set name="products" inverse="true">
            <key column="category_id" not-null="true" />
            <one-to-many class="com.jnesis....Product"  />
        </set>
</class>
```

A vous de jouer

- TP 7

Mapping de l'héritage

Mapping de l'héritage

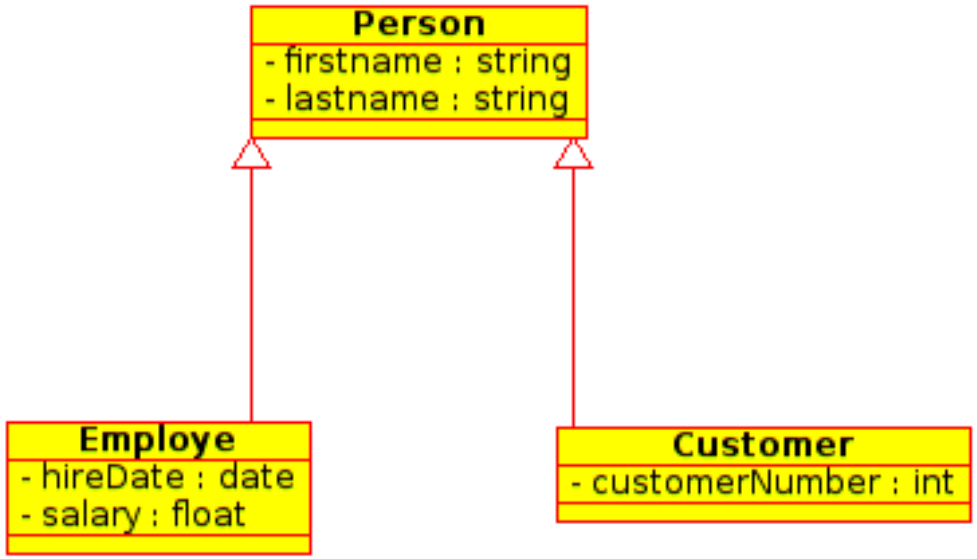
- Les bases de données relationnelles ne disposent généralement pas de la notion d'héritage.
- Il va falloir traduire cette notion sous forme d'associations entre les tables. Différentes approches sont possibles :
  - Une table par classe concrète en utilisant le polymorphisme de manière implicite
  - Une table par classe concrète
  - Une table par hiérarchie de classes
  - Une table par classe

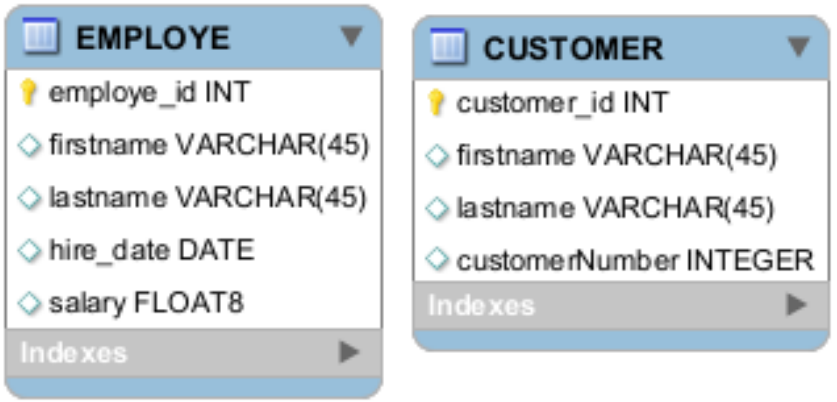
Mapping de l'héritage – polymorphisme implicite

- Une table par classe concrète en utilisant le polymorphisme de manière implicite
  - Dans le cas de l'implémentation d'une interface ou de l'héritage d'une classe abstraite
  - Il s'agit de ne mapper que les classes concrètes

Mapping de l'héritage – polymorphisme implicite

- Seules les classes Employe et Customer seront mappés
- Chacune contiendra le mapping des attributs communs firstname et lastname





Mapping de l'héritage – polymorphisme implicite

```
<class name="com.jnesis.examples.hibernate.model.Employe"
table="employe">
        <id column="employe_id" name="id">
            <generator class="native"/>
        </id>
        <property name="firstname" />
        <property name="lastname" />
        <property name="hireDate" column="hire_date" />
        <property name="salary" />
    </class>
```

```
<class name="com.jnesis.examples.hibernate.model.Customer"
table="customer">
        <id column="customer_id" name="id">
            <generator class="native"/>
        </id>
        <property name="firstname" />
        <property name="lastname" />
        <property name="customerNumber" />
    </class>
```

Mapping de l'héritage – polymorphisme implicite

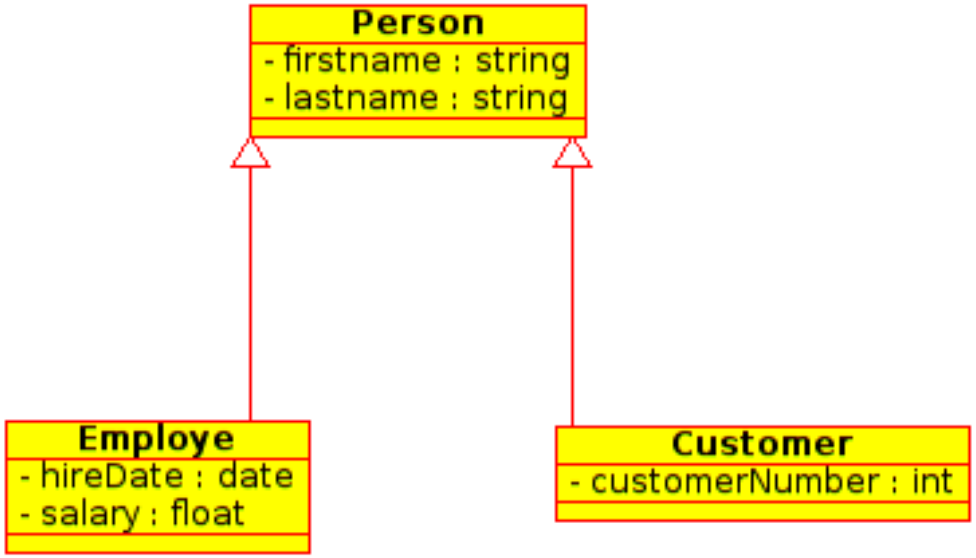
- Avantages
  - Il reste possible de faire avec Hibernate des **requêtes polymorphiques** (récupérer des listes de Person alors même que cette classe abstraite n'apparaît nulle part dans le mapping !).
- Inconvénients
  - Il n'est **pas possible de créer des associations polymorphiques,** c'est à dire vers la classe abstraite ou l'interface (pas d'association vers Person)
  - **Les propriétés communes sont mappées dans chaque classe**, il faut que le mapping reste cohérent en mappant ses propriétés de manière strictement identiques
  - **Les requêtes polymorphiques sont peu performantes** car elles vont engendrer autant de requêtes SQL qu'il y a de sous-classes et ce sans utiliser d'union.

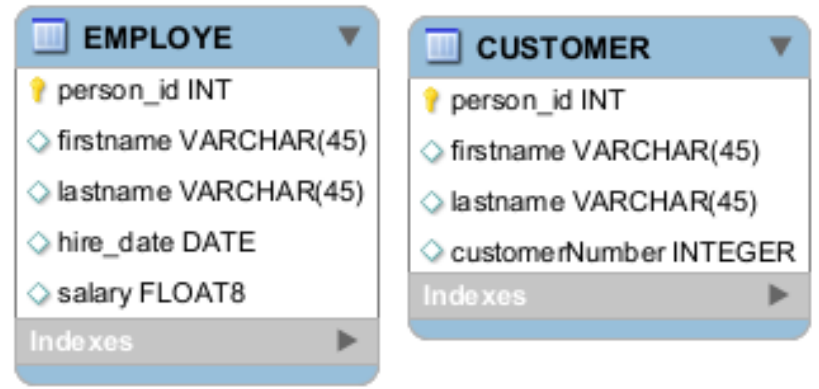
Mapping de l'héritage – une table par classe concrète

- Une table par classe concrète
  - Une table sera utilisée pour chaque classe concrète
  - Pour tous les types d'héritage :
    - L'implémentation d'une interface
    - L'héritage d'une classe abstraite
    - L'héritage d'une classe concrète

Mapping de l'héritage – une table par classe concrète

- Les trois classes sont mappées
- Les attributs communs sont déplacés dans le mapping de Person et seront répétés dans chaque table.
- Les données d'une classe concrète ne sont stockées que dans une seule table dédiée



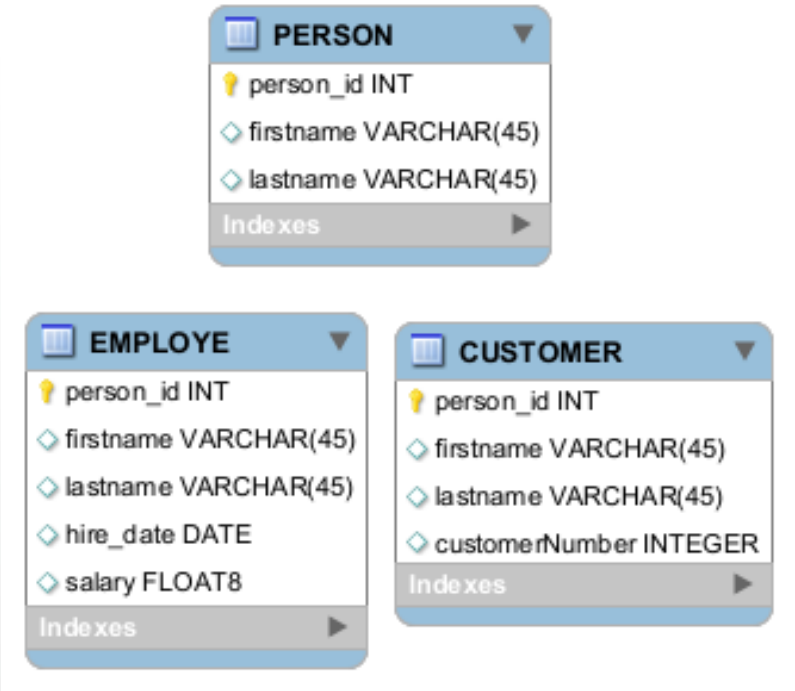


Mapping de l'héritage – une table par classe concrète

```
<hibernate-mapping>
   <class name="com.jnesis....Person" abstract="true">
      <id column="person_id" name="id">
         <generator class="increment"/>
      </id>
      <property name="firstname" />
      <property name="lastname" />

      <union-subclass name="com.jnesis...Employe" table="employe">
         <property name="hireDate" column="hire_date" />
         <property name="salary" />
      </union-subclass>
      <union-subclass name="com.jnesis...Customer" table="customer">
          <property name="customerNumber" />
      </union-subclass>
   </class>
</hibernate-mapping>
```

Mapping de l'héritage – une table par classe concrète

- Dans le mapping de la classe parente :
  - **abstract="true"** , vient spécifier que l'on travaille sur une classe abstraite ou une interface.
  - **abstract="false"** génèrerait la table correspondant à la Person puisqu'elle devient concrète.

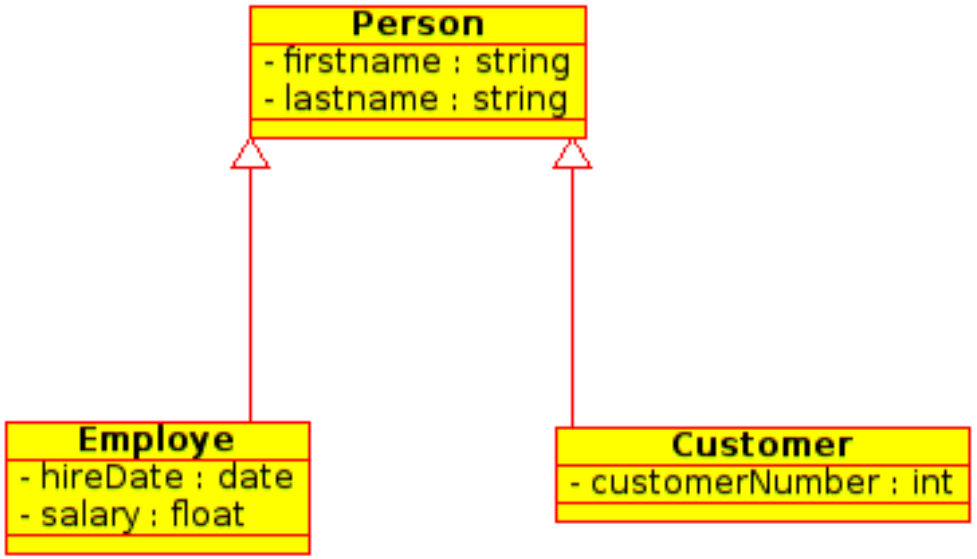


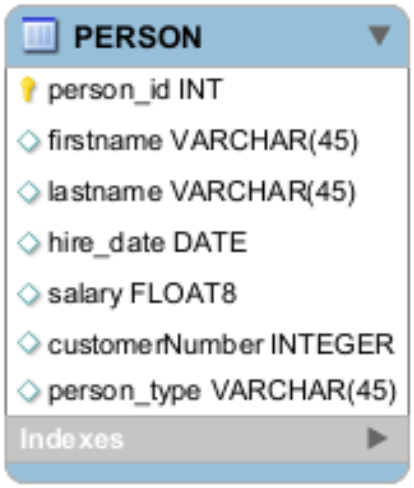
Mapping de l'héritage – une table par classe concrète

- Avantages
  - **Les attributs communs ne sont déclarés qu'à un seul endroit du mapping**, la cohérence est donc assurée
  - Il devient **possible de créer des associations polymorphiques**, c'est à dire vers la classe parente (même si cette dernière est abstraite ou une interface)
  - **Les requêtes polymorphiques utilisent 'union', les performances sont meilleures** qu'avec le polymorphisme implicite
- Inconvénients
  - La clé primaire disposant d'une valeur unique pour l'ensemble des tables impliquées dans l'héritage, il devient **impossible d'utiliser un générateur d'identifiant de type 'identity'**

Mapping de l'héritage – une table par hiérarchie

- Une table par hiérarchie de classes
  - L'ensemble des classes d'une hiérarchie (classes et sous classes) sont stockées dans une seule et même table.
  - Afin d'identifier la classe concrète de chaque enregistrement de la table, une colonne discriminante devra obligatoirement être utilisée.





Mapping de l'héritage – une table par hiérarchie

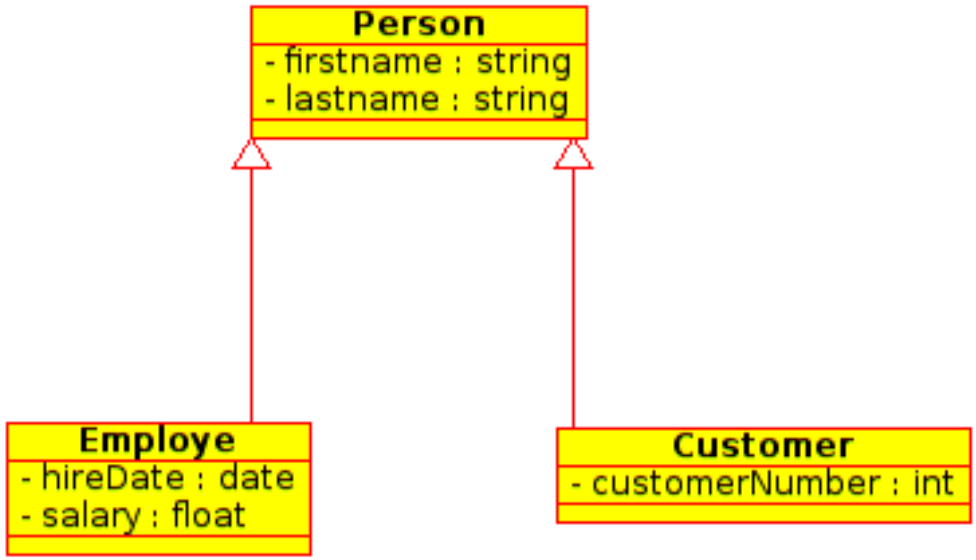
```
<hibernate-mapping>
   <class name="com.jnesis...Person" abstract="true">
      <id column="person_id" name="id">
         <generator class="native"/>
      </id>
      <discriminator column="person_type" type="string" />
      <property name="firstname" />
      <property name="lastname" />

      <subclass name="com.jnesis...Employe" discriminator-value="EMP">
         <property name="hireDate" column="hire_date" />
         <property name="salary" />
      </subclass>
      <subclass name="com.jnesis...Customer" discriminator-value="CUST">
          <property name="customerNumber" />
      </subclass>
   </class>
</hibernate-mapping>
```

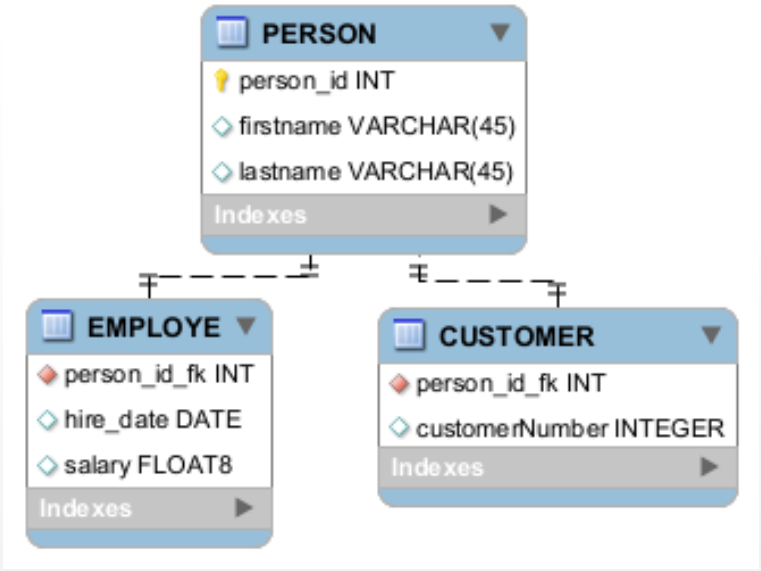
Mapping de l'héritage – une table par hiérarchie

- Avantages
  - **Excellente performance en terme de requêtage polymorphique** (pas besoin de jointures ni d'unions)
  - **Simplicité**
- Inconvénients
  - **Tous les attributs des sous-classes doivent impérativement autoriser la valeur NULL** interdisant ainsi l'usage de contrainte d'intégrité de type « not-null ».
  - **Dénormalisation du schéma** car il y a une dépendance fonctionnelle entre des colonnes qui ne sont pas liées à un identifiant.

Mapping de l'héritage – une table par classe

- Une table par classe
  - Chaque classe d'une hiérarchie (classes et sous classes abstraites ou non) dispose de sa table.
  - Chaque table ne contiendra que les colonnes des attributs qui ne sont pas hérités.
  - Chaque sous-classe contiendra une clé étrangère référençant la classe parente.





Mapping de l'héritage – une table par classe

```
<hibernate-mapping>
   <class name="com.jnesis.examples.hibernate.model.Person" abstract="true">
      <id column="person_id" name="id">
         <generator class="native"/>
      </id>
      <property name="firstname" />
      <property name="lastname" />

      <joined-subclass name="com.jnesis.examples.hibernate.model.Employe">
         <key column="person_id_fk" />
         <property name="hireDate" column="hire_date" />
         <property name="salary" />
      </joined-subclass>
      <joined-subclass name="com.jnesis.examples.hibernate.model.Customer">
         <key column="person_id_fk" />
         <property name="customerNumber" />
      </joined-subclass>
   </class>
</hibernate-mapping>
```

Mapping de l'héritage – une table par classe

- Avantages
  - **Le schéma reste normalisé**, les contraintes de références peuvent s'appliquer sans limitations particulières.
- Inconvénients
  - **Les performances peuvent être fortement dégradées sur des hiérarchies d'héritage complexe** (multiples jointures entre les tables)

A vous de jouer

- TP 8