Hibernate : Mapping Objet Relationnel

Cyril Labbé

LIG, Université de Grenoble, France
first.last@imag.fr

http://membres-lig.imag.fr/labbe/IBD/Hibernate_IBD.pdf
http://membres-lig.imag.fr/labbe/IBD/hibernate_query.pdf
http://membres-lig.imag.fr/labbe/IBD/Zoo_hiber.zip

Table of Contents

Persistance et programmation

Stockage persistant à l'aide d'un SGBD relationnel

- Modèle relationnel solide et répandu
- Langage SQL (définition de données et interrogation)
- Gestion des transactions et concurrences
- Cohérence et reprise après panne
- Performant

Langage de programmation

- Paradigme *objets*
- Java

Faire le pont entre

- le monde objet
- le monde Java

Lien Java - SQL

Chacun son rôle

- SGBD-R : source de persistance
- Java : logique applicative

Pourquoi JDBC ne suffit pas

- Les requêtes sont construites par le code Java (*à la main*)
- Le résultat d'une requête est une *relation*
- Les objets *de l'application* sont reconstruits (*à la main*)

- C'est beaucoup de travail fastidieux,
- sans lien direct avec les fonctionnalités de l'application.

Persistance d'objets

Sérialisation d'objets

- sauvegarde de l'état du graphe d'objets
- accès à tout ou rien

Pour utiliser le modèle relationnel il faut résoudre :

- le problème de l'héritage
- le problème de la granularité
- des problèmes d'identification
- ...

Le cas simple

Les attributs sont atomiques

```
public class Person {
        Long PERSON_ID;
        String FIRST_NAME;
        String LAST_NAME;

        public Person(Long id, String fname, String lname) {...}

        public Long getPERSON_ID(){...}
        public void setPERSON_ID(Long id ) {...}
        public String getFIRST_NAME(){...}
        public void setFIRST_NAME(String na) {...}
        public String getLAST_NAME(){...}
        public void setLAST_NAME (String lna) {...}
}
```

Mapping relationnel

- $Persons(\underline{id}, FName, LName)$ $< id, fn, ln > \in Persons \iff \{...\}$

Exemple : Problème de granularité

Quand un attribut est un object complexe

```
public class Address {
    Long ADDRESS_ID;
    String STREET;
    String CITY;
    String STATE;
    String ZIP;
    Person person; // the person this address belongs to.

    public Person getPerson() {return person;}
    public void setPerson(Person person){this.person = person;}
    ...//rest of get/set methods
}
```

Mapping relationnel

- *Addresses*(*a_id*, *Street*, *City*, *State*, *Zip*, *??*)

  $< id, fn, ln > \in Addresses \iff \{...\}$

- forme normale ?

Exemple : Problème de granularité

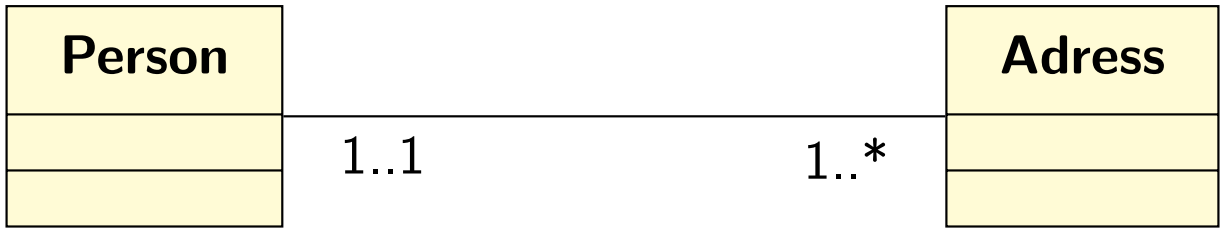
Quand un attribut est un ensemble

```
public class Person {
        Long PERSON_ID;
        String FIRST_NAME;
        String LAST_NAME;
        Set ADDRESSES = new HashSet();
        public Set getADDRESSES() {return ADDRESSES; }
        public void setADDRESSES(Set ADDRESSES) { this.ADDRESSES = ADDRESSES;}
        public void addAdd(Address a)
        {
                a.setperson(this);
                ADRESSES.add(a);
        }
        ...//rest of get/set methods
}
```

Mapping relationnel

- *Persons(id, FName, LName, ??)* $< id, fn, ln > \in Persons \iff \{...\}$
- forme normale ?

Diagramme UML



Mapping relationnel

- *Persons(id, FName, LName, ??)*
- *Addresses(a_id, Street, City, State, Zip, ??)*

Table of Contents

Hibernate

Déclarer des mapping relationnel

- comment un objet est *traduit* en relationnel
- (fichier xml)

Exécuter des requêtes

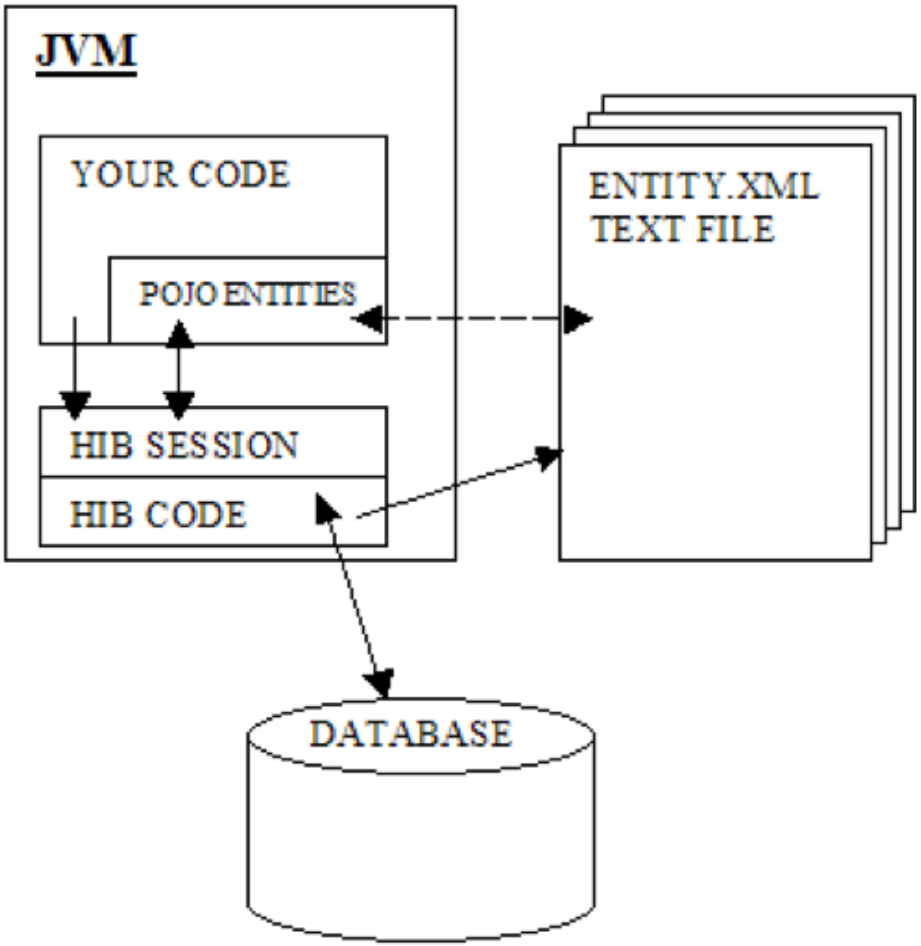
- SQL, HQL, QBE

Transactions

Techniquement

un ensemble de classe java.

Hibernate vue globale



- La couche Hibernate est indépendante du code de *métier*.
- Mapping décrit via des fichier XML
- Hibernate : DML, caching, niveau d'isolation.

Configuration

```
BLA BLA BLA

<hibernate-configuration>

<session-factory>
      <!-- DB connection settings -->
      <property name="hibernate.connection.driver_class"> oracle.jdbc.driver.OracleDriver
           </property>
      <property name="hibernate.connection.url"> jdbc:oracle:thin:@hopper:1521:ufrima</
           property>
      <property name="hibernate.connection.username">BD-Login-Name</property>
      <property name="hibernate.connection.password">BD-Password</property>
      <property name="hibernate.connection.pool_size">10</property>

      <!-- Echo all executed SQL to stdout -->
      <property name="show_sql">true</property>

      <!-- SQL Dialect -->
      <property name="dialect">org.hibernate.dialect.OracleDialect</property>
      <property name="hibernate.hbm2ddl.auto">update</property>

      <!-- Mapping files -->
      <mapping resource="Person.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

Table of Contents

Rendre un objet persistant

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class FirstExample {
  public static void main(String[] args) {
    Session session = null;
    Configuration cf ;

    try{//Create new instance of Person and set values in it.
        Person P = new Person();
        P.setId(3);
        P.setFirstName("Claudia");
        P.setLastName("Roncancio");

        // This step will read hibernate.cfg.xml and prepare hibernate for use.
        SessionFactory sessionFactory = new Configuration().configure("hibernate.cfg.xml"
            ).buildSessionFactory();
        session = sessionFactory.openSession();

        session.save(P);
        System.out.println("Done");

        session.close();

        P.setFirstName("Fabrice");
        P.setLastName("Jouanot");

    }catch(Exception e){
      System.out.println("catch !:"+e.getStackTrace());
    }
```

Etat d'un object vis à vis de la persistence

Un objet *o* est *Transient*

- *o* créé par `new`, il n'est pas associé à un élément de la BD
- Les objets référencés par des instances transient sont transient
- *o* devient *Persistant* par `save(o)`ou par `saveOrUpdate(o)`.

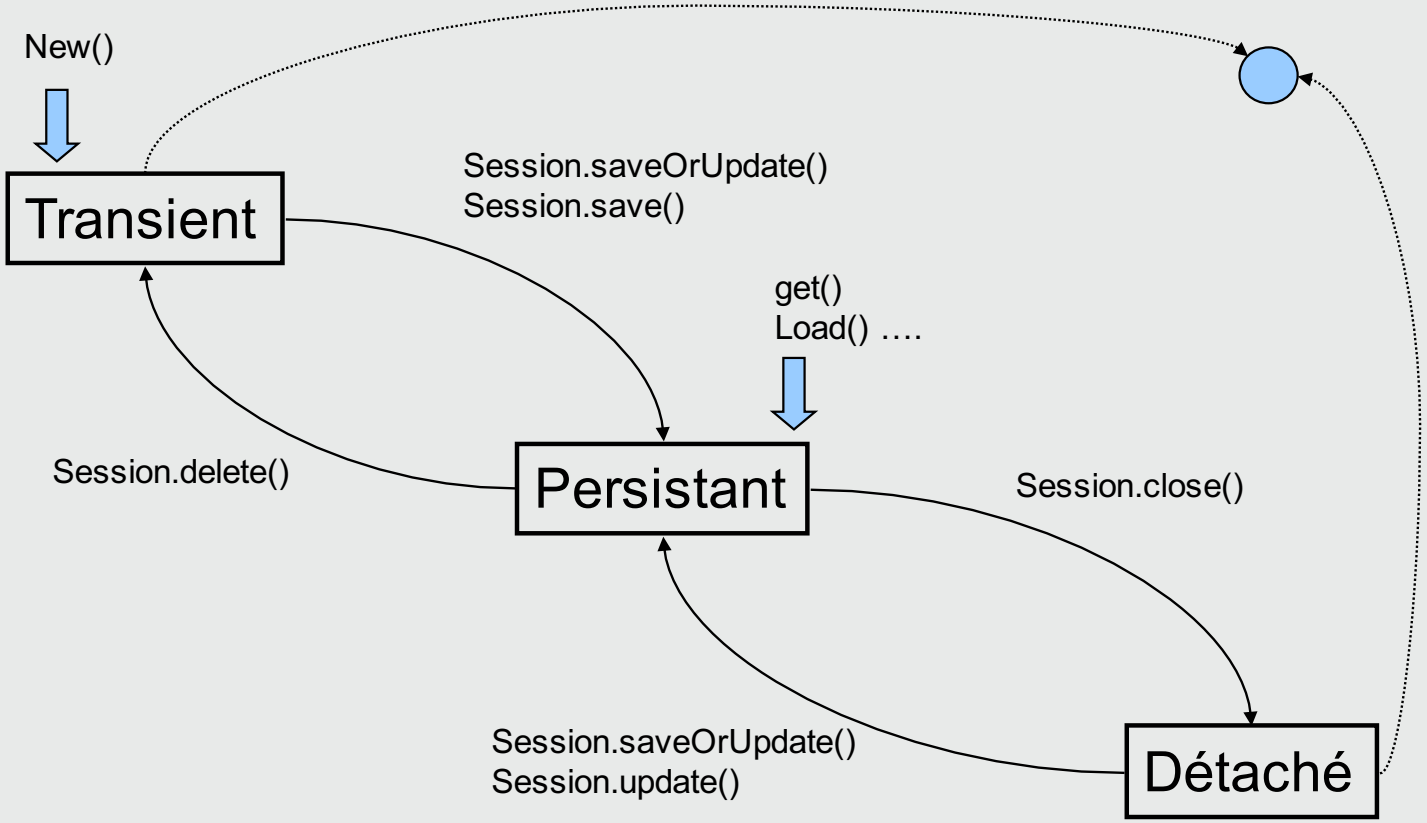
Un objet *o* est *Persistent*

- *o* est associé à un élément de la BD (`save(o)` ou `saveOrUpdate(o)`)
- *o* a été obtenu par exécution d'une requête
- Un objet persistant est toujours associé à une session (aspect *transactionnels*)
- L'état en mémoire est propagé à la BD le plus tard possible (au commit)
- Un objet persistant devient *Transient* après un `delete(o)`
- Un objet persistant devient *Detached* après un `session.close()`

Un objet *o* est *Detached*

- A la fin d'une transaction les objets existent toujours en mémoire
- La synchronisation avec la BD n'est plus assurée
- Ces instances peuvent être réutilisées et réassociées à d'autres session

Cycle de vie d'un objet



Egalité entre objet

Object identity

- Deux objets sont identiques si ils occupent le même espace mémoire
- `(a==b)`

Object equality

- Deux objets différents ont même valeur
- `a.equals(b)`

La persistance ajoute un type d'égalité : database identity

- Deux objets en mémoire peuvent représenter le même object persistant
- Un objet persistant doit être identifié de manière unique : `a.getId().equals(b.getId()`

Rendre un objet persistant

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class FirstExample {
  public static void main(String[] args) {
    Session session = null;
    Configuration cf ;

    try{//Create new instance of Person and set values in it.
        Person P = new Person();
        P.setId(3);
        P.setFirstName("Claudia");
        P.setLastName("Roncancio");

        // This step will read hibernate.cfg.xml and prepare hibernate for use.
        SessionFactory sessionFactory = new Configuration().configure("hibernate.cfg.xml"
            ).buildSessionFactory();
        session = sessionFactory.openSession();

        session.save(P);
        System.out.println("Done");

        session.close();

        P.setFirstName("Fabrice");
        P.setLastName("Jouanot");

    }catch(Exception e){
      System.out.println("catch !:"+e.getStackTrace());
    }
```

Table of Contents

Déclaration d'un mapping simple

```
public class Person {
        Long PERSON_ID;
        String FIRST_NAME;
        String LAST_NAME;

        public Person(Long id, String fname, String lname) {...}

        public Long getPERSON_ID(){...}
        public void setPERSON_ID(Long id ) {...}
        public String getFIRST_NAME(){...}
        public void setFIRST_NAME(String na) {...}
        public String getLAST_NAME(){...}
        public void setLAST_NAME (String lna) {...}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
                "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
                "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="Person" table="Person_Table" >
        <id name="PERSON_ID" column="P_ID" type="Long">
        </id>
        <property name="FIRST_NAME" type="String" column="F_NAME" />
        <property name="L_NAME"/>
</class>
</hibernate-mapping>
```

Persons(P_id, F_Name, L_Name)

Déclaration d'un *One to many*

```
public class Person {
        Long PERSON_ID;
        String FIRST_NAME;
        String LAST_NAME;
        Set ADDRESSES = new HashSet();
        public Set getADDRESSES(){return ADDRESSES;}
        public void setADDRESSES(Set ADDRESSES){this.ADDRESSES = ADDRESSES;}
        public void addAdd(Address a){ a.setperson(this);ADRESSES.add(a);}
...//rest of get/set methods
}
```

```
...
<hibernate-mapping>

<class name="Person" table="Person" >
...
    <set    name="ADDRESSES"
        inverse="true"
        cascade="save-update">
              <key  column="Person_ID"/>
              <one-to-many class="Address"/>
    </set>
...
</class>
</hibernate-mapping>
```

Persons(<u>id</u>, FName, LName)

Déclaration d'un *Many to one*

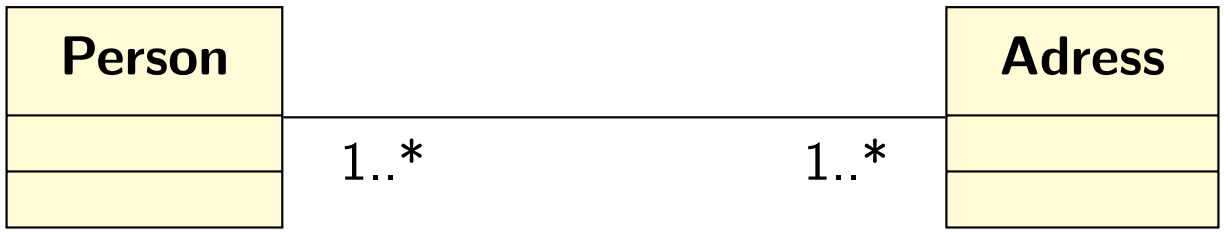
```
public class Address {
    Long ADDRESS_ID;
    String STREET;
    String CITY;
    String STATE;
    String ZIP;
    Person person; // the person this address belongs to.
    public Person getPerson(){return person;}
    public void setPerson(Person person){this.person = person;}
...//rest of get/set methods
}
```

```
...
<hibernate-mapping>

<class name="Address"   table="Addresses" >
    <id name="ADDRESS_ID"  column="ADDRESS_ID">
    </id>
    <many-to-one
           name="person"
           class="Person"
           column="Person_ID"
     />
...rest of properties defined
</class>
</hibernate-mapping>
```

Addresses(a_id, Street, City, State, Zip, #P_ID)

Relation *Many to many*



Mapping relationnel

- *Persons(p_id, FName, LName)*
- *Addresses(a_id, Street, City, State, Zip)*
- *PERSON_ADDRESS(#a_id,#p_id*

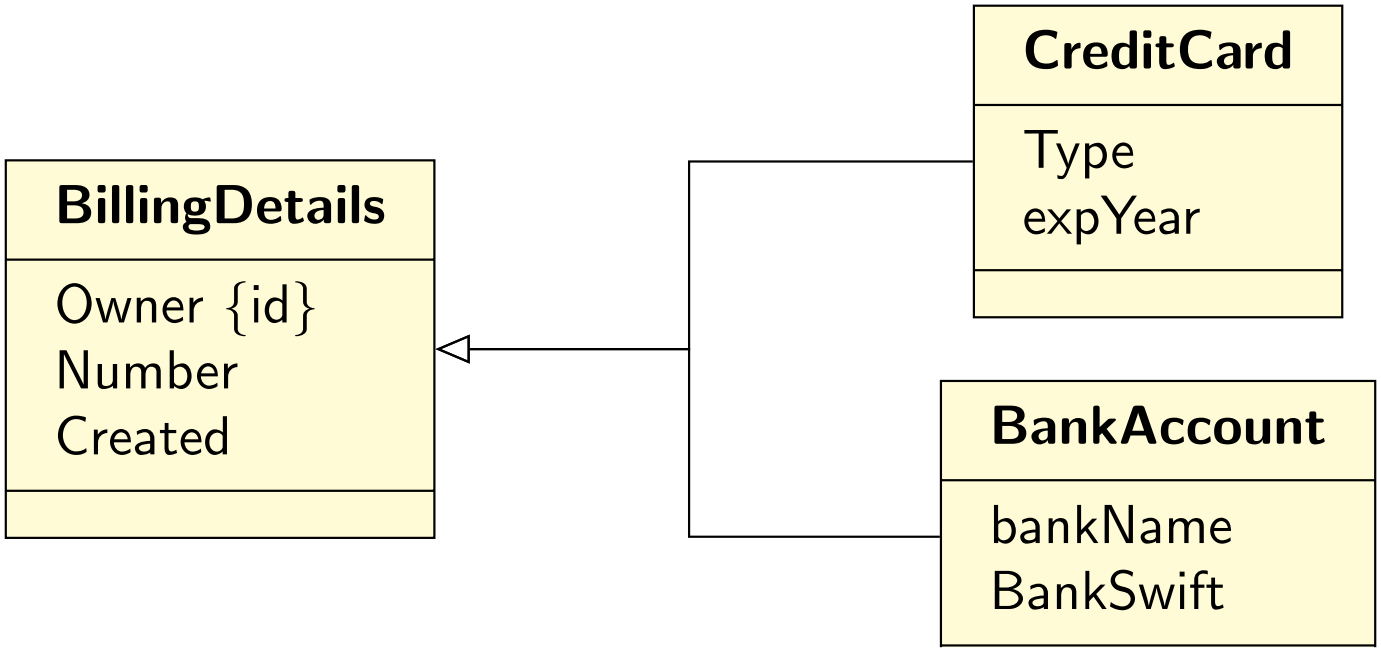
Déclaration d'un *Many to many*

```
...
<hibernate-mapping>
<class name="Person" table="Person" >
...
   <set  name="ADDRESSES"  table = "PERSON_ADDRESS"  cascade="save-update">
            <key  column="P_ID"/>
            <many-to-many class="Address"  colums="ADDRESS_ID"
            />
    </set>
...
</class>
</hibernate-mapping>
```

```
...
<hibernate-mapping>
<class name="Address"   table="Addresses" >
    <id name="ADDRESS_ID"  column="ADDRESS_ID">
    </id>
    <set name="persons"  table="PERSON_ADDRESS"     inverse="true"   cascade="save-update
        ">
        <key column="ADDRESS_ID"/>
        <many-to-many
                class="Person"     column="Person_ID"/>
    </set>
...rest of properties defined
</class>
</hibernate-mapping>
```

PERSON_ADDRESS(#a_id,#p_id)

Héritage



3 possibilités :

- Une table par classe (concrète)
- Une table par hiérarchie
- une table pas sous-classe

Une table par classe (concrète)

- Une table par classe (concrète). Pas de table pour la classe abstraite
- Toutes les propriétés d'une classes sont représentées par un attribut y compris les propriétés héritées
- Une requête sur la classe abstraite doit être traduite par plusieurs requêtes (une pour chaque classe concrète)
- Les requêtes sur les classes concrètes sont simples

Mapping classique

```
<hibernate-mapping>
...
<class name="CreditCard" table="CREDIT_CARD_T" >
        <id name="CreditCard_id" column="Credit_card_id" type="Long">
        </id>
        <property
            name="Owner" type="String" column="Owner"
        />
        ... //
</class>

...
</hibernate-mapping>
```

Une table par hiérarchie

- Dénormalisation
- La classes concrète est données par le discriminateur
- Performance et simplicité (à défaut de propreté)
- Pas d'union ou de jointure à faire... Les requêtes sur les classes abstraites et concrète sont simples
- Problèmes majeurs : les propriétés des sous-classes doivent pouvoir être *null* et forte redondance qui nuit au mises à jour

Mapping *Une table par hiérarchie*

```
...
<hibernate-mapping>
...
<class name="BillingDetails" >
        <id name="id" column="Billing_detail_id" type="Long">
        </id>
     <discriminator column="Billing_Details_Type" type="string" />
        <property
            name="Owner" type="String" column="Owner"
        />
        ... / les autres prop /
        <subclass
                        name="CreditCard"
                        discriminator-value="CC" >
                <property name="type" column="Credit_Card_Type"/>
                 ...
       </subclass>
       ... / les autres sous-classes /
</class>
...
</hibernate-mapping>
```

Une table par sous-classe

- Une table pour chaque classe (concrète ou abstraite)
- Un attribut pour chaque propriété *non hérité*.
- Une clé primaire qui est aussi une clé étrangère de la table représentant la superclass
- Les instances d'une classes sont retrouvées par jointure entre la table de la classe et la table de la superclasse.
- Les tables sont normalisées
- A ne pas utiliser avec des hiérarchies de classes (et/ou des requêtes) complexes

Déclaration mapping *une table par sous-classe*

```
...
<hibernate-mapping>
...
<class name="BillingDetails"  table="BILLING_DETAILS_T">
        <id name="id" column="Billing_detail_id" type="Long">
        </id>
        <property
            name="Owner" type="String" column="Owner"
        />
        ... / les autres prop. abstraites /
        <joined-subclass
                        name="CreditCard"
                        table="CREDIT_CARD_T" >
                    <key column="Credit_Card_id">
                    <property name="type" column="Credit_Card_Type"/>
                    ... / les autres prop. de la sous-classes /
       </joined-subclass>
       ... / les autres sous-classes /
</class>
...
</hibernate-mapping>
```

Table of Contents

Différents langages

- Hibernate Query Language HQL : orienté objet
- Query By Example (QBE)/ Query By Criteria (QBC) : Interrogation par critères
- SQL natif : écrire des requêtes dans un dialecte particulier, procédure stockées...

Interrogation de classes persistantes

```
List results = session.createQuery("from Person").list();

        Iterator iter = results.iterator();
        // Parcours de la liste et affichage des Elements
        while (iter.hasNext()) {
                Person cc = iter.next();
                System.out.println(cc.getFirst_Name());
        }

List results2 = session.createQuery("from Person").list();
```