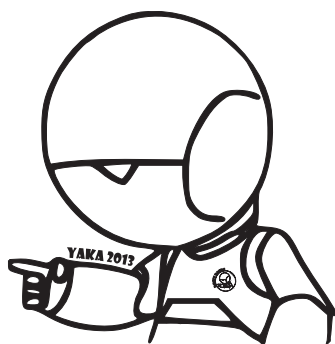


## Hibernate - TP

---

Version 1  
2012



**DON'T PANIC !  
CODE TO FIND THE ANSWER**

YAKA 2013 <[yaka@epita.fr](mailto:yaka@epita.fr)>

---

## Copyright

Ce document est destiné à un usage interne à EPITA <<http://www.epita.fr/>>.

Copyright © 2012/2013 Assistants <[yaka@epita.fr](mailto:yaka@epita.fr)>.

**La copie de ce document est autorisée uniquement sous les conditions suivantes :**

- ▷ Vous avez téléchargé votre copie grâce à l'intranet des assistants <<https://www.acu.epita.fr/intra/>>.
- ▷ Vous devez vous assurer d'être en possession de la dernière version en date du présent document.
- ▷ Il est de votre responsabilité de vous assurer que ce document reste hors de la portée de tiers n'appartenant à votre "promotion".

## Table des matières

<b>1</b>	<b>Hibernate</b>	<b>1</b>
1.1	Présentation . . . . .	1
1.2	La Base de données . . . . .	2
1.3	Fonctionnement . . . . .	4
1.4	Configuration . . . . .	4
1.5	Beans . . . . .	7
1.6	Annotations . . . . .	7
1.6.1	Introduction . . . . .	7
1.7	La session factory . . . . .	10
1.8	Les DAO : Data Access Object . . . . .	13
1.8.1	Introduction . . . . .	13
1.8.2	Criteria . . . . .	13
1.8.3	HQL . . . . .	14
1.8.4	Mon premier DAO . . . . .	15
1.8.5	A votre tour ! . . . . .	17
1.9	Aller plus loin . . . . .	18
1.9.1	Gagner du temps . . . . .	18
1.9.2	C'est l'heure des tests ! . . . . .	18
1.9.3	HQL et Criteria : un peu de pratique . . . . .	18
<b>2</b>	<b>Conclusion</b>	<b>19</b>

# 1 Hibernate

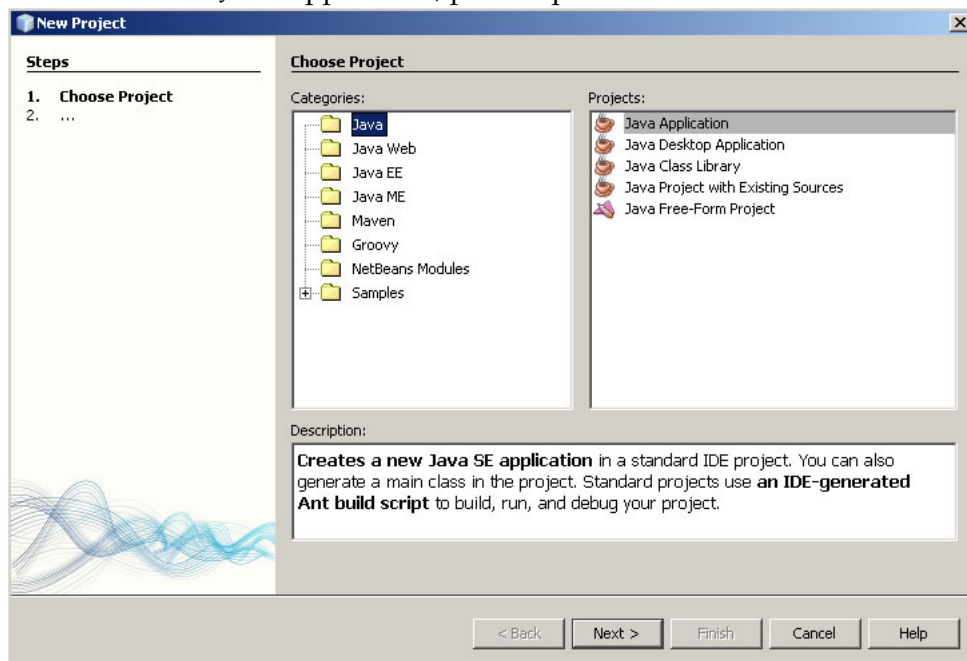
## 1.1 Présentation

Hibernate est un Framework Java permettant de récupérer les informations d'une base de données dans du code Java de façon transparente via l'Object Mapping<sup>1</sup>.

Lors de l'utilisation d'une base de données il n'est pas très pratique de faire des requêtes directement en SQL et de transformer le résultat en un objet Java. C'est la raison pour laquelle on utilise des Frameworks comme Hibernate en Java qui facilitent les accès à la base de données en fournissant un *wrapper* autour de ceux-ci.

Commençons par créer une nouvelle application Java dans NetBeans 'File/New Project'.

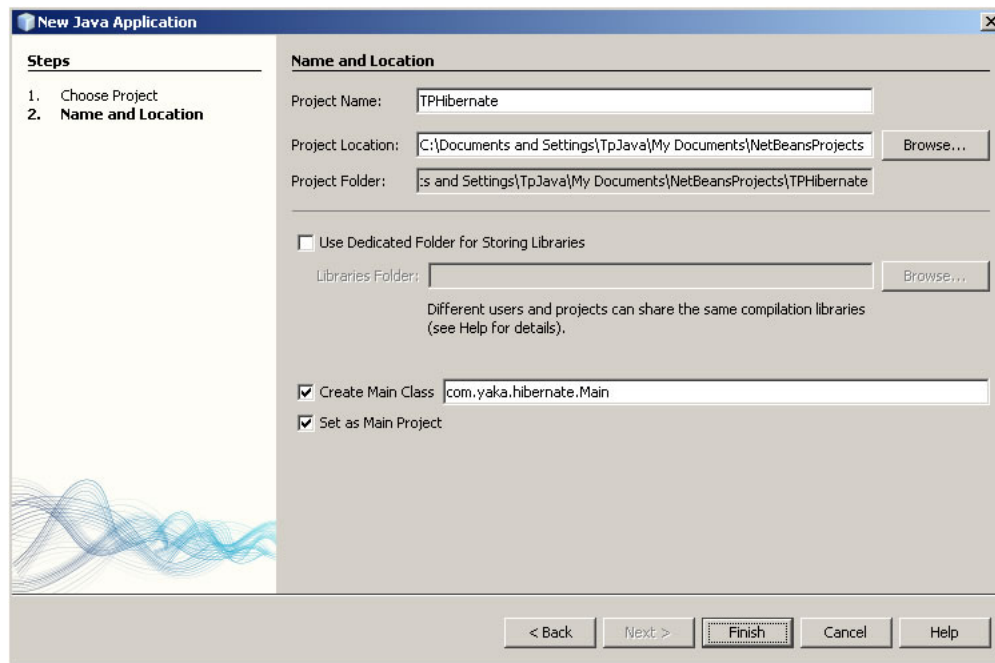
Sélectionnez Java Application, puis cliquez sur Next.



Cliquez sur Finish.

---

1. L'Object Mapping est une méthode permettant de créer une bijection entre des objets au sens POO du terme et les tables d'une base de données relationnelle classique



Vous devriez avoir un nouveau projet avec une classe **Main** dans le package **com.yaka.hibernate**.

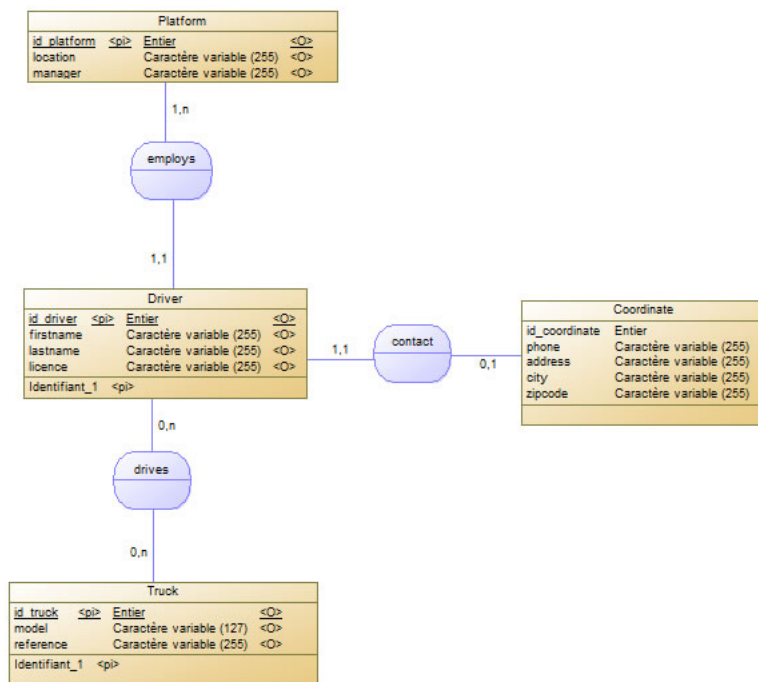
## 1.2 La Base de données

Afin de pouvoir pratiquer Hibernate dans de bonnes conditions, voici une description de notre projet :

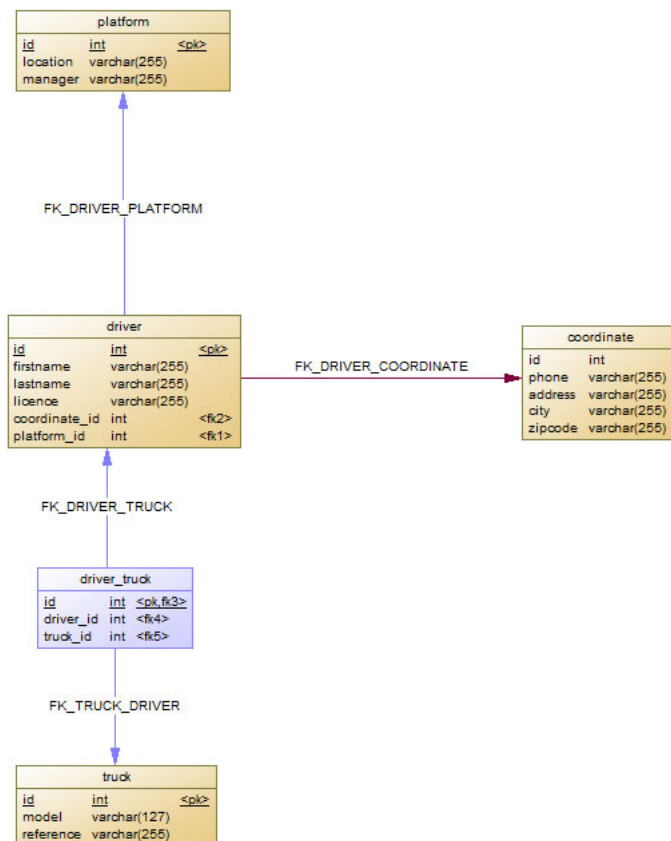
Une grande société aimerait pouvoir gérer mieux ses livraisons, pour cela elle nous explique son fonctionnement :

- Nous avons des conducteurs (Nom, prénom, Numéro de Permis) qui peuvent conduire plusieurs camions (Modèle, Référence).
- Chaque conducteur est associé à une plateforme (Nom du gérant, Lieu).
- Chaque conducteur possède une fiche de coordonnées (Téléphone, Adresse, Ville, Code Postal).

A partir de ces informations on obtient la modélisation Merise suivante :



Et à partir de cette modélisation Merise on obtient la modélisation physique suivante :



Maintenant que nous avons nos deux modélisations, nous pouvons créer notre base MySQL que nous nommerons 'hibernate'.

Pour cela, nous vous fournissons un script de création de base de données, à exécuter .

Le script demande un mot de passe qui est 'ykrn' afin de créer la base. Vous pouvez exécuter à tout moment ce fichier pour réinitialiser votre base.

Notez qu'il est aussi possible de demander à Hibernate de générer la base tout seul !

### 1.3 Fonctionnement

Le framework Hibernate permet de créer une application à partir d'**objets beans** :

- **Une classe bean** représente une entité de la base de données, soit une table.
- **Un objet bean** représente un tuple de la même table.

Hibernate associe automatiquement<sup>2</sup> les **classes beans** à la base de données permettant de récupérer directement sous forme d'objets les données associées. Cela permet de s'abstraire du langage de la base de données.

Il est toutefois malgré tout nécessaire de réaliser certains accès à la base. Cela se fait *via* deux méthodes :

- **Criteria** : C'est une API qui se base sur des critères fournis par le(s) bean(s) voulu(s). La requête sera formée « automagiquement » à partir de ces critères.
- **HQL** : pour Hibernate Query Language. Il est très similaire au SQL mais le nommage utilise les classes beans et non les champs de la table.

Nous reviendrons sur ces méthodes plus tard dans ce document.

### 1.4 Configuration

La configuration d'Hibernate est limitée à un seul fichier XML : `hibernate.cfg.xml`. Voici ses propriétés les plus importantes.

- **hibernate.connection.url** : URL ou le chemin de connexion à la base de données.  
**MySQL** : `jdbc:mysql://localhost:3306/[DB]`
- **hibernate.connection.driver\_class** : Le driver pour se connecter à la base de données.  
**MySQL** : `com.mysql.jdbc.Driver`
- **hibernate.connection.username** : l'utilisateur de la base de données.
- **hibernate.connection.password** : Le mot de passe de l'utilisateur de la base de données.
- **dialect** : La bibliothèque pour générer les requêtes.  
**MySQL** : `org.hibernate.dialect.MySQLDialect`
- **hibernate.current\_session\_context\_class** : Indique la manière dont la session (qui est l'interface entre Hibernate et Java) sera liée au contexte courant. Nous utiliserons ici la valeur « thread » qui indique qu'une session Hibernate sera liée au thread l'ayant ouverte.
- **hibernate.show\_sql** : Utile pour déboguer, montre le code SQL exécuté.

Il existe d'autres propriétés avec des valeurs par défaut que vous pouvez modifier. Pour avoir plus d'informations, nous vous invitons à consulter la documentation Hibernate. Vous pourrez par exemple y trouver comment faire générer ou mettre à jour la base de données par le *framework* au lancement du serveur, ce qui peut être utile lors du développement d'une application pilote<sup>3</sup>. Après les propriétés nous avons les balises *mapping* utilisées pour spécifier le chemin de nos beans dans le projet.

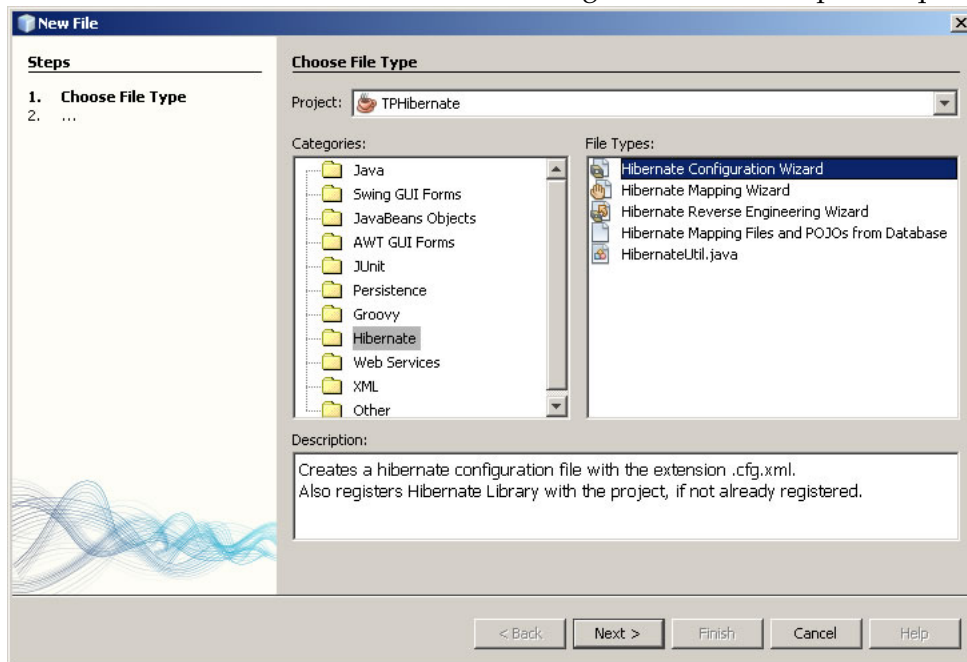
```
<mapping class="com.yaka.hibernate.[Class]" />
```

Nous pouvons maintenant générer le fichier de configuration Hibernate : *File/NewFile*.

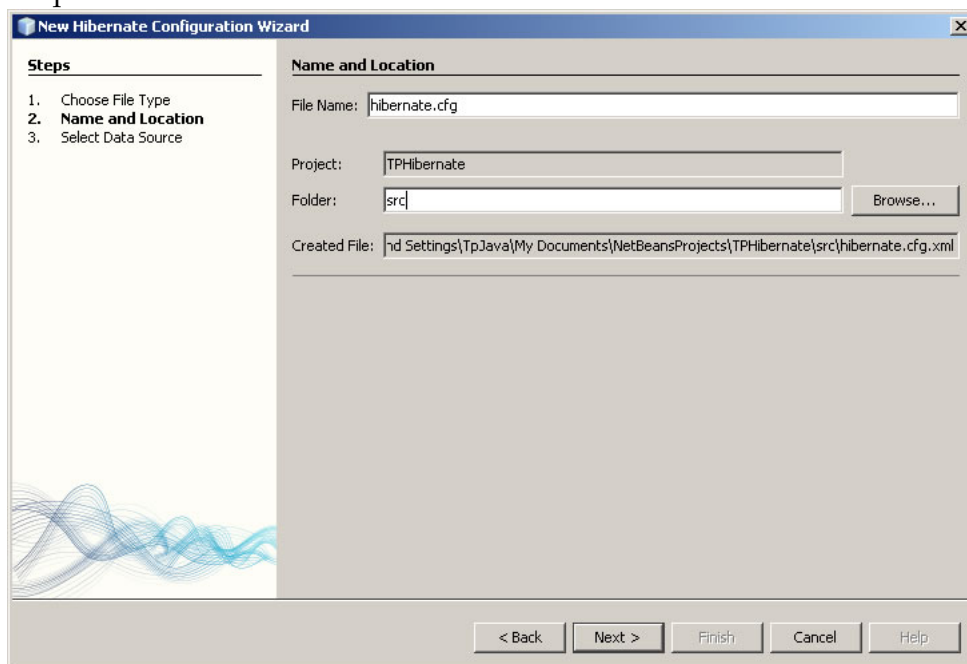
2. Lorsqu'il est bien configuré !

3. Un indice parce que je suis gentil : **hibernate.hbm2ddl.auto**, à vous de trouver les options !

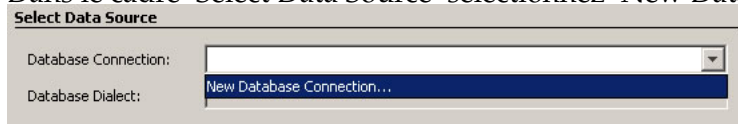
Sélectionnez 'Hibernate/Hibernate Configuration Wizard', puis cliquez sur suivant.



Cliquez sur suivant.

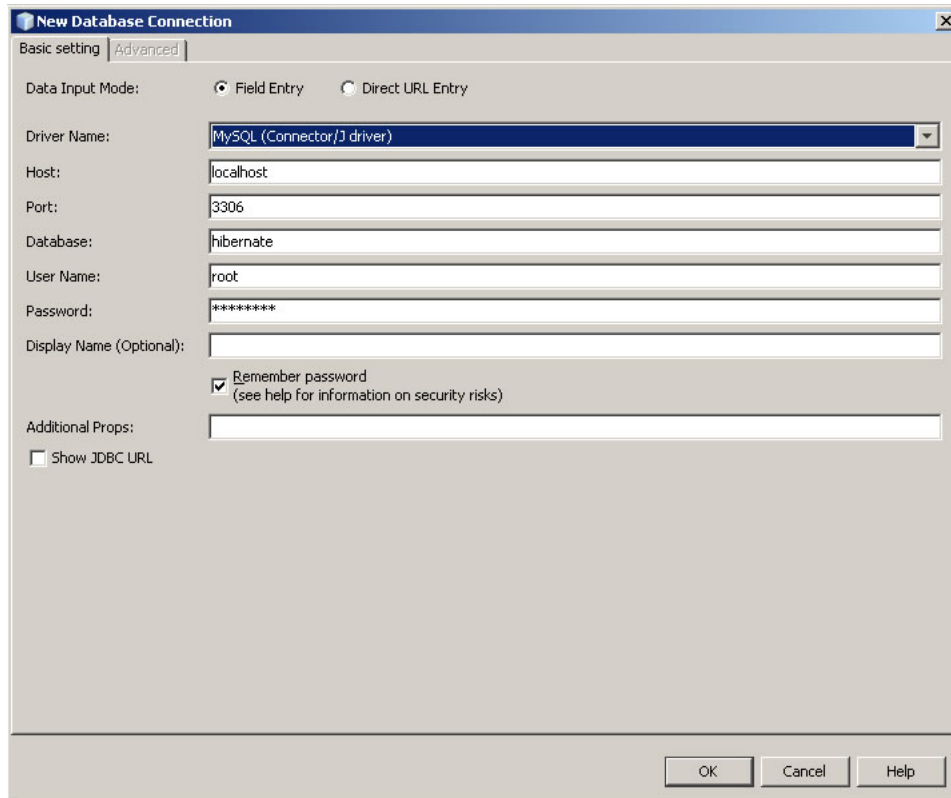


Dans le cadre 'Select Data Source' sélectionnez 'New Data Source'



Recopiez les informations ci-dessous, le mot de passe est 'ykrn'.





The image shows a 'New Database Connection' dialog box with a 'Basic setting' tab selected. It contains fields for Driver Name (MySQL (Connector/J driver)), Host (localhost), Port (3306), Database (hibernate), User Name (root), and Password (masked with asterisks). There are also checkboxes for 'Remember password' and 'Show JDBC URL', and buttons for 'OK', 'Cancel', and 'Help'.

**New Database Connection**

Basic setting | Advanced

Data Input Mode: ☒ Field Entry ☐ Direct URL Entry

Driver Name: MySQL (Connector/J driver)

Host: localhost

Port: 3306

Database: hibernate

User Name: root

Password: \*\*\*\*\*

Display Name (Optional):

☒ Remember password  
(see help for information on security risks)

Additional Props:

☐ Show JDBC URL

OK Cancel Help

## 1.5 Beans

Nous allons maintenant écrire les classes beans pour associer nos tables à nos objets <sup>4</sup>.

Un bean est une classe qui suit certaines règles :

- Un bean doit être sérialisable (implements Serializable);
- Un bean doit implémenter des getters et des setters pour chaque attribut;
- Les getters et setters doivent être au format Sun(Netbeans peut les générer automatiquement);
- Un bean possède un constructeur sans argument;
- Le nom de la classe bean doit être le nom de la table dans la base;
- Les noms des attributs du bean doivent être ceux des colonnes de la table sauf pour les clés étrangères;
- Si la relation est de type 1..N ou 0..N-0..N la clé étrangère est symbolisée par un Set de l'élément de la relation.

Exemple :

Si un Toto a plusieurs Titi, le bean de Toto aura un attribut 'titis' du type Set<Titi>.

- Si la relation est de type N..1 ou 1..1 la clé étrangère est symbolisée par un attribut de type de l'autre extrémité.

Exemple :

Si un Toto a un Titi, le bean de Toto aura un attribut 'titi' du type Titi.

Exemple :

```
// Table 'toto'
public class Toto implements Serializable
{
    //Bons getter/setter
    private String name; // colonne 'name'
    Private Titi titi; // colonne titi_id

    public String getName() {...};
    public void setName(String n) {...};

    public void setTiti(Titi titi) {...};
    public Titi getTiti(){...};

    //mauvais getter
    public String getname() {...};
    public String get_Name() {...};
}
```

Créez maintenant les beans pour les 4 tables de notre modélisation.

Il faut ensuite penser à ajouter les relations de mapping dans le fichier de configuration XML d'Hibernate.

## 1.6 Annotations

### 1.6.1 Introduction

Pour chaque bean que nous avons écrit il va falloir indiquer les associations liées à la base de données. Pour cela Hibernate propose deux mécanismes :

---

4. Et réciproquement.

- Le fichier de mapping XML d'Hibernate ;
- Un mapping à l'aide d'annotations.

Nous ne détaillerons ici que la méthode utilisant les annotations. Toutefois, si vous désirez avoir la main sur le fonctionnement précis du mapping au prix d'une écriture moins compacte, vous pouvez trouver un rapide résumé du mapping XML ici :

<http://ndpsoftware.com/HibernateMappingCheatSheet.html>

N'oubliez pas, si vous utilisez cette méthode, d'ajouter un nœud mapping à la configuration XML pour indiquer l'emplacement de votre fichier de mapping s'il n'existe pas déjà. Son fonctionnement est similaire à celui des annotations sur le principe mais en utilisant un fichier XML descriptif. Cette méthode est en grande partie *legacy*, mais à un avantage : vous pouvez faire générer à la fois la base *et* les beans Java à partir du XML !

Voyons donc les annotations de plus près avec le bean Platform.

```
@Entity // Specifie que la classe correspond a une table.
public class Platform implements Serializable
{
    private Integer id;
    private String location;
    private String manager;
    // La liste des Drivers associes a une platform
    private Set<Driver> drivers;

    @Id // Specifie que Id est la cle primaire de l'entite
        Platform.
    @GeneratedValue(strategy=GenerationType.AUTO)
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }
    // Le constructeur vide indispensable au bean
    public Platform() {
    }

    // Un autre constructeur
    public Platform(String location, String manager) {
        this.location = location;
        this.manager = manager;
    }

    // Getter pour manager
    public String getManager(){
        return manager;
    }

    // Setter pour manager
    public void setManager(String manager){
        this.manager = manager;
    }
}
```

```
// Getter pour location
public String getLocation() {
    return location;
}

// Setter pour location
public void setLocation(String location) {
    this.location = location;
}

// Defini une relation de type "Une Platform pour Plusieurs Drivers"
@OneToMany
// Permet de repercuter les actions lors des insertions et suppressions
@Cascade(org.hibernate.annotations.CascadeType.ALL)
public Set<Driver> getDrivers() {
    return drivers;
}

public void setDrivers(Set<Driver> drivers) {
    this.drivers = drivers;
}
}
```

#### Tips :

- Vérifiez vos imports ! ils doivent être du type **javax.persistence.\*** et **org.hibernate.\*** pour la plupart, pas autre chose, plusieurs classes ou packages ont le même nom, ne vous trompez pas !;
- N'oubliez pas d'ajouter les bibliothèques nécessaires à votre projet.

Dans ce fichier on peut voir plusieurs annotations intéressantes :

- **@Entity** : Cette annotation est indispensable pour chaque bean, elle permet l'association entre la classe bean et la table MySQL ;
- **@Id** : Elle se place sur le getter de la clé primaire, l'annotation **@GeneratedValue** permet de spécifier l'auto-incrémentation ;
- **@OneToMany** : Permet de créer une relation de type 1..N, le getter doit renvoyer un Set de l'entité associée ;
- **@Cascade(CascadeType)** : Spécifie qu'une cascade doit être effectuée lors d'un ajout, modification ou suppression sur l'entité ;
- **@OneToOne(mappedBy="l'entity")** : Permet de créer une relation de type 1..1, le getter doit renvoyer un élément de l'entité associée. L'argument **mappedBy** permet de spécifier l'entité associée à la relation ;
- **@ManyToOne** : Permet de créer une relation de type N..1, le getter doit renvoyer un élément de l'entité associée ;
- **@ManyToMany** : Permet de créer une relation de type 0..N - 0..N, le getter doit renvoyer un Set de l'entité associée. Pour une relation de ce type la syntaxe complète est la suivante pour une relation 0..N - 0..N entre Toto et Titi :

```
@ManyToMany(targetEntity = com.yaka.hibernate.bean.Toto.class,
            cascade = {CascadeType.PERSIST, CascadeType.MERGE},
```

```

        fetch=FetchType.EAGER)
// Le fetchtype eager permet d'éviter les problèmes de
// laziness au moment de la récupération d'objets.
@JoinTable(name = "titi_toto", // Nom de la table de
liaison
// Nom des clés de jointure
joinColumns = @JoinColumn(name = "titi_id"),
inverseJoinColumns = @JoinColumn(name = "toto_id"))
@Cascade(org.hibernate.annotations.CascadeType.ALL)
public Set<Toto> getTotos()
{
    return this.totos;
}

```

Pour plus d'informations sur les annotations la documentation est disponible sur le lien suivant :

[http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html\\_single/](http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/)

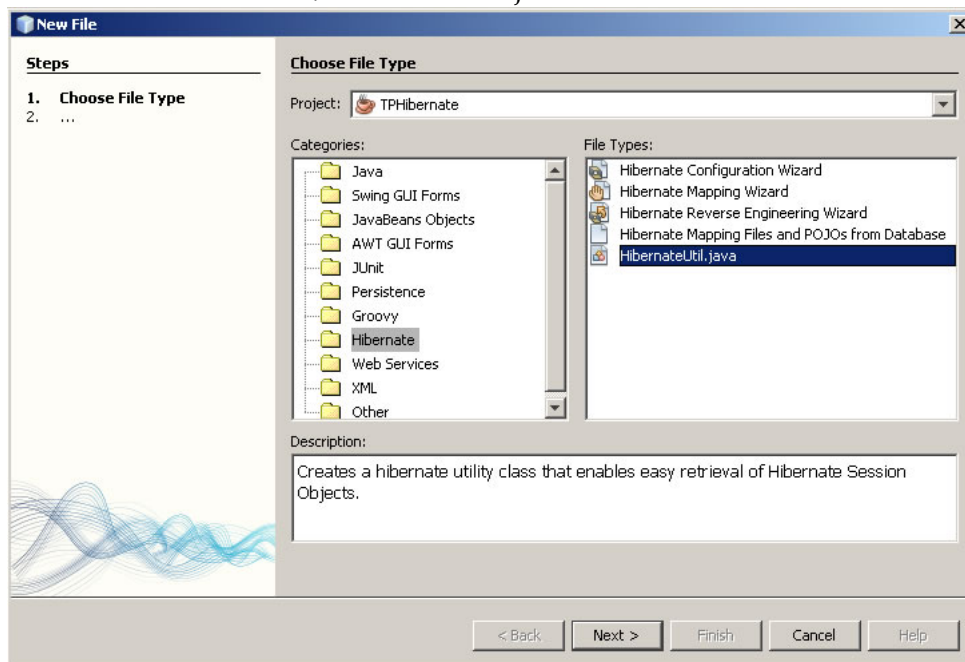
Rajoutez toutes les annotations nécessaires aux autres beans de votre projet.

Note avancée : pour les relations ManyToMany, une table intermédiaire est nécessaire. Il vous est toutefois possible de mapper cette table elle aussi et de la traiter comme n'importe quelle autre entité si vous voulez contrôler finement les associations. Ce n'est bien sûr utile que si vous voulez optimiser votre modèle.

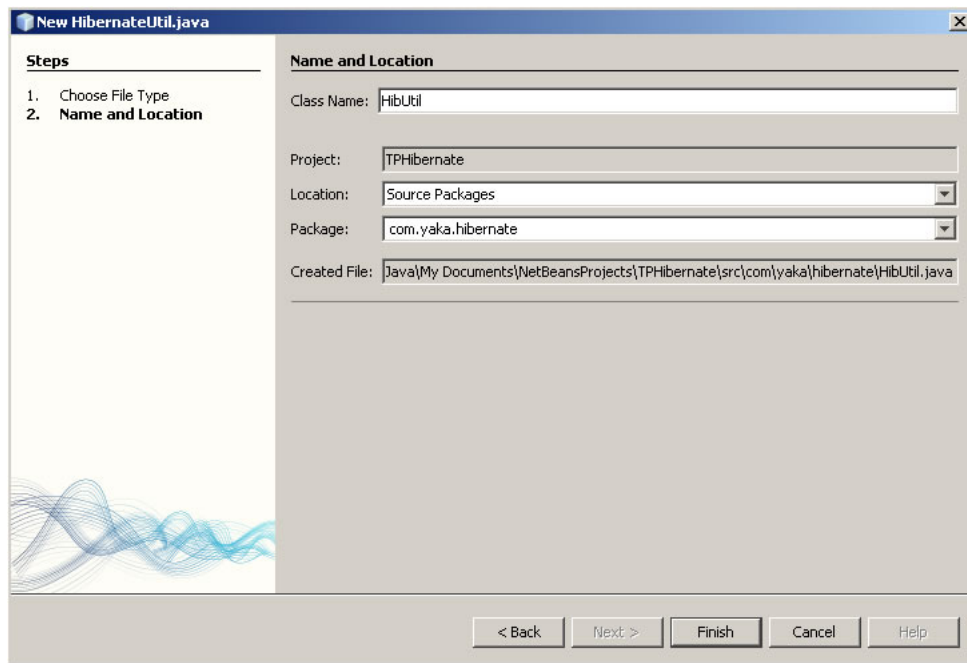
## 1.7 La session factory

La session factory est un élément central dans l'utilisation d'Hibernate. C'est un singleton qui va lancer les connexions avec la base de données. Pour la créer avec NetBeans il faut aller dans le menu "New File"

Sélectionnez 'Hibernate/HibernateUtil.java'.



Renommez la classe 'HibUtil' et validez.



The image shows a 'New HibernateUtil.java' dialog box in an IDE. It has a 'Steps' pane on the left and a 'Name and Location' pane on the right. The 'Steps' pane shows two steps: '1. Choose File Type' and '2. Name and Location', with the second step being the active one. The 'Name and Location' pane contains several input fields: 'Class Name' (HibUtil), 'Project' (TPHibernate), 'Location' (Source Packages), 'Package' (com.yaka.hibernate), and 'Created File' (Java\My Documents\NetBeansProjects\TPHibernate\src\com\yaka\hibernate\HibUtil.java). At the bottom, there are five buttons: '< Back', 'Next >', 'Finish', 'Cancel', and 'Help'.

**New HibernateUtil.java**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name:

Project:

Location:

Package:

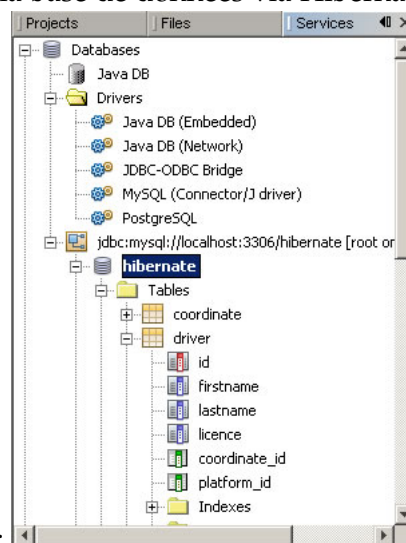
Created File:

< Back   Next >   Finish   Cancel   Help

Afin de pouvoir vérifier si vos beans ont bien été édités voici un petit bout de code à placer dans le main.

```
public static void main(String[] args)
{
    // On cree une plateforme
    Platform platform = new Platform("Paris","Jean");
    // On cree des coordonnees
    Coordinate coord = new Coordinate("0102030405", "42 avenue
        des JSP", "Paris", "Jean");
    // On cree un conducteur
    Driver driver = new Driver("Jean", "Nemmard", "122-64552");
    // Ajout d'un camion
    Truck truck = new Truck("Volvol7M", "abcd");
    // Ajout d'un autre camion
    Truck truck2 = new Truck("Volvol8M", "dbca");
    Set<Truck> trucks = new HashSet<Truck>();
    trucks.add(truck);
    trucks.add(truck2);
    // Ajout de la plateforme au conducteur
    driver.setPlatform(platform);
    // Ajout des coordonnees au conducteur
    driver.setCoordinate(coord);
    // Ajout des camions au conducteur
    driver.setTrucks(trucks);
    // On recupere la session liee a la SessionFactory
    Session session = HibUtil.getSessionFactory().
        getCurrentSession();
    // On commence une transaction avec la base
    Transaction t = session.beginTransaction();
    // On sauvegarde le conducteur et les entites associes
    // Necessite un Cascade sur les sauvegardes
    session.save(driver);
    // Ferme la transaction
    t.commit();
}
```

Lancez votre projet, les tuples suivants devraient être insérés dans la base :  
Si vous voulez voir les informations de la base de données via Hibernate, cliquez sur l'onglet



'Server' et déroulez la partie 'Database' :

- **platform** : un tuple
- **cordinate** : un tuple
- **driver** : un tuple
- **driver\_truck** : 2 tuples
- **truck** : 1 tuple

## 1.8 Les DAO : Data Access Object

### 1.8.1 Introduction

Le *design pattern* DAO (Data Access Object) a pour but de réunir dans une seule entité les opérations sur une source de données (base, fichier, ...). Les DAO sont des classes qui vont permettre de manipuler nos beans plus facilement. Ce sont ces classes qui vont faire les requêtes. Elles utilisent la session factory pour récupérer les informations dans la base de données. Les DAO implémentent en général au minimum le CRUD, qui sont les 4 opérations basiques sur une table (Create, Retrieve, Update, Delete) ainsi qu'une méthode pour récupérer la liste des éléments d'une table. Libre à vous bien entendu d'y ajouter ce dont vous avez besoin. Il est d'usage de créer d'abord une interface pour chaque DAO qu'on implémentera ensuite sous forme de singleton.

Par exemple le DAO pour la classe Platform sera :

**com.yaka.hibernate.dao.PlatformDAO**

Son implémentation sera :

**com.yaka.hibernate.dao.impl.PlatformDAOImpl.**

### 1.8.2 Criteria

Afin de récupérer nos données, il est possible d'utiliser Criteria. Voici un exemple d'utilisation de cette API :

```
/*
 * Recupere la liste des coordonnees qui respectent l'identifiant
 * ou qui ont pour ville Paris
 */
public List<Coordinate> getCriteria(int id)
{
```



```
//On recupere la session
Session sess = HibUtil.getSessionFactory().getCurrentSession
();
Transaction t = sess.beginTransaction();
// On cree un critere sur l'entite coordinate.
Criteria crit = session.createCriteria(Driver.class);
// On restreint selon l'id
crit.add(Restrictions.eq("id", id));
/*
 * On cree un second critere suivant l'attribut coordinate
 * de la classe Driver
 */
Criteria coordcrit = crit.createCriteria('coordinate');
// On restreint selon la ville
coordcrit.add(Restrictions.eq("city", "Paris"));
// On recupere la liste des resultats
List<Coordinate> result = crit.list();
// On finalise la transaction
t.commit();
return result;
}
```

Comme son nom le laisse deviner, Criteria permet de définir des critères de sélection qui seront automatiquement transformés en requêtes sans que vous n'ayez à intervenir.

Pour plus d'informations sur les Criteria :

<http://docs.jboss.org/hibernate/core/3.3/reference/en/html/querycriteria.html>

### 1.8.3 HQL

Afin de récupérer nos données, il est aussi possible d'utiliser HQL, voici le même exemple d'utilisation avec HQL :

```
/*
 * Recupere la liste des drivers qui respectent l'identifiant
 * ou qui ont pour ville Paris
 */
public List<Coordinate> getHQL(int id)
{
    //On recupere la session
    Session sess = HibUtil.getSessionFactory().getCurrentSession
();
    //Notre requete HQL
    String hql = "select d from Driver d join d.coordinate as
        coord
            where d.id = :id and coord.city = :city";
    Transaction t = sess.beginTransaction();
    // On cree la requete
    Query query = sess.createQuery(hql);
    // On specifie la valeur :id
    query.setInteger("id", id);
    query.setString("city", "Paris");
    // On specifie la valeur :city
    List<Coordinate> coords = query.list();
}
```

```
t.commit();
return result;
}
```

HQL permet de remplacer SQL pour avoir accès à des valeurs Java. Pour plus d'informations sur le langage HQL :

<http://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html>

#### 1.8.4 Mon premier DAO

Voici l'interface PlatformDAO :

```
public interface PlatformDAO
{
    public List<Platform> list();
    public Platform get(Integer id);
    public Integer save(Platform platform);
    public Boolean update(Platform platform);
    public Boolean delete(Integer id);
}
```

Et le code pour la classe PlatformDAOImpl :

```
public class PlatformDAOImpl implements PlatformDAO
{
    // Instance de notre singleton
    private static final PlatformDAOImpl instance = new
        PlatformDAOImpl();
    // Constructeur prive du singleton
    private PlatformDAOImpl ()
    {
    }

    // La methode pour recuperer le singleton
    static public PlatformDAO getInstance()
    {
        return instance;
    }
    /*
    * La methode pour recuperer une plate-forme selon l'ID avec
    Criteria
    */
    public Platform get(Integer id)
    {
        // On recupere la session
        Session sess = HibUtil.getSessionFactory().
            getCurrentSession();
        // On commence une transaction avec la base
        Transaction t = sess.beginTransaction();
        // On creer un critere suivant la classe Platform
        Criteria crit = sess.createCriteria(Platform.class);
        // On restreint ce critere suivant la valeur de l'id
        voulu
        crit.add(Restrictions.eq("id", id));
    }
}
```

```
        // On ne veut et doit recuperer qu'un element
        Platform plat = (Platform) crit.uniqueResult();
        // On finalise la transaction
        t.commit();
        return plat;
    }

    /**
     * Recupere la liste des plates-formes
     */
    public List<Platform> list() {
        Session sess = HibUtil.getSessionFactory().
            getCurrentSession();
        Transaction t = sess.beginTransaction();
        Criteria crit = sess.createCriteria(Platform.class);
        // On recupere la liste des platform
        List<Platform> list = crit.list();
        t.commit();
        return list;
    }

    /**
     * Sauve une plate-forme
     */
    public Integer save(Platform platform)
    {
        Session sess = HibUtil.getSessionFactory().
            getCurrentSession();
        Transaction t = sess.beginTransaction();

        // On sauvegarde la platform
        Integer id = (Integer) sess.save(platform);
        t.commit();
        return id;
    }

    public Boolean update(Platform platform)
    {
        try
        {
            Session sess = HibUtil.getSessionFactory().
                getCurrentSession();
            Transaction t = sess.beginTransaction();
            // On met a jour la plateforme
            sess.update(platform);

            t.commit();
            return true;
        }
        catch (HibernateException e)
        {
            e.printStackTrace();
        }
    }
}
```

```
        return false;
    }
}

/*
 * Exemple d'utilisation avec le HQL pour supprimer une plate-
 * -form
 */
public Boolean delete(Integer id)
{
    try
    {
        Session sess = HibUtil.getSessionFactory().
            getCurrentSession();
        String req = "Delete from Platform where id=:id";
        Transaction t = sess.beginTransaction();
        Query q = sess.createQuery(req);
        q.setInteger("id", id.intValue());
        int rowcount = q.executeUpdate();

        t.commit();
        if (rowcount != 0)
            return true;
        return false;
    }
    catch (HibernateException e)
    {
        e.printStackTrace();
        return false;
    }
}
```

Comme vous pouvez le voir, votre DAO peut insérer, supprimer, mettre à jour, récupérer et lister les éléments de la table platform.

### 1.8.5 A votre tour !

Beaucoup de points de détail et de mécaniques ne sont pas décrits dans ces lignes car c'est à vous de découvrir et maîtriser les outils que nous vous présentons. Il ne vous reste qu'à coder la suite et à poser des questions pertinentes !

## 1.9 Aller plus loin

### 1.9.1 Gagner du temps

Maintenant que nous avons tout nos DAO il serait intéressant d'avoir une classe qui permet un accès facile à ceux-ci. On appelle cette classe une Factory<sup>5</sup>, c'est un singleton qui contient une instance de chaque DAO. Ceci permet un accès rapide aux DAO et donc aux données. Créer la Factory suivante : *DAOFactory*.

### 1.9.2 C'est l'heure des tests !

Maintenant que tout a été fait, vous pouvez écrire dans votre classe Main les méthodes suivantes afin de tester votre code.

- **public void displayAll()** : Affiche tout le contenu de toutes les tables.
- **public void addDriver(Coordinate coor, Driver driver, Platform platform)** : Ajouter un tuple dans la table driver et affiche *"Added driver FIRSTNAME LASTNAME whose license number is LICENSE"*.
- **public void addPlatform(Platform platform)** : Ajoute un tuple à la table platform et affiche *"Added Platform which location is LOCATION and manager is MANAGER"*.
- **public void addTruck(Truck truck)** : Ajoute un tuple à la table truck *"Added truck which reference is REFERENCE"*

Ca ne marche pas ? N'hésitez pas à déboguer votre code !

### 1.9.3 HQL et Criteria : un peu de pratique

Voici quelques requêtes que vous devez ajouter à votre code, à vous de choisir le HQL ou Criteria suivant vos préférences.

- Easy
  - + Récupérer tous les conducteurs de *Volvo 50M*
  - + Récupérer tous les conducteurs qui vivent *Paris*
- Medium
  - + Récupérer tous les conducteurs de *Volvo 50M* et dont le manager est *Sabrine*
  - + Récupérer tous les conducteurs qui s'appellent *Gregoire* et qui vivent à *Villejuif*
- Hard
  - + Récupérer toutes les platforms qui ont au moins un conducteur qui conduit un *Opel 17M* et qui vit à *Villejuif*
  - + Récupérer tous les conducteurs dont le manager est *Jerome*, qui vivent au *Le Kremlin Bicetre* et qui conduisent une *Volvo 50M*

---

5. Ce *design pattern* devrait vous dire quelque chose... Si ce n'est pas le cas, vous savez qui est votre ami ;)

## 2 Conclusion

Voilà, vous avez toutes les clés en main pour bien utiliser Hibernate !

*Don't worry...Code to find the answer !...*