Débuter avec Hibernate sous Eclipse

par Julien DEFAUT

Date de publication :

Dernière mise à jour :

Les applications d'entreprises s'orientent de plus en plus vers des architectures n-tiers. La technologie J2EE et les frameworks qui se sont créés autour offrent beaucoup d'outils pour répondre aux besoins modernes. Nous allons nous intéresser dans cet article, sous la forme d'un tutorial, au framework de mapping objet-relationnel le plus populaire pour J2EE appelé Hibernate. Téléchargez la version pdf.

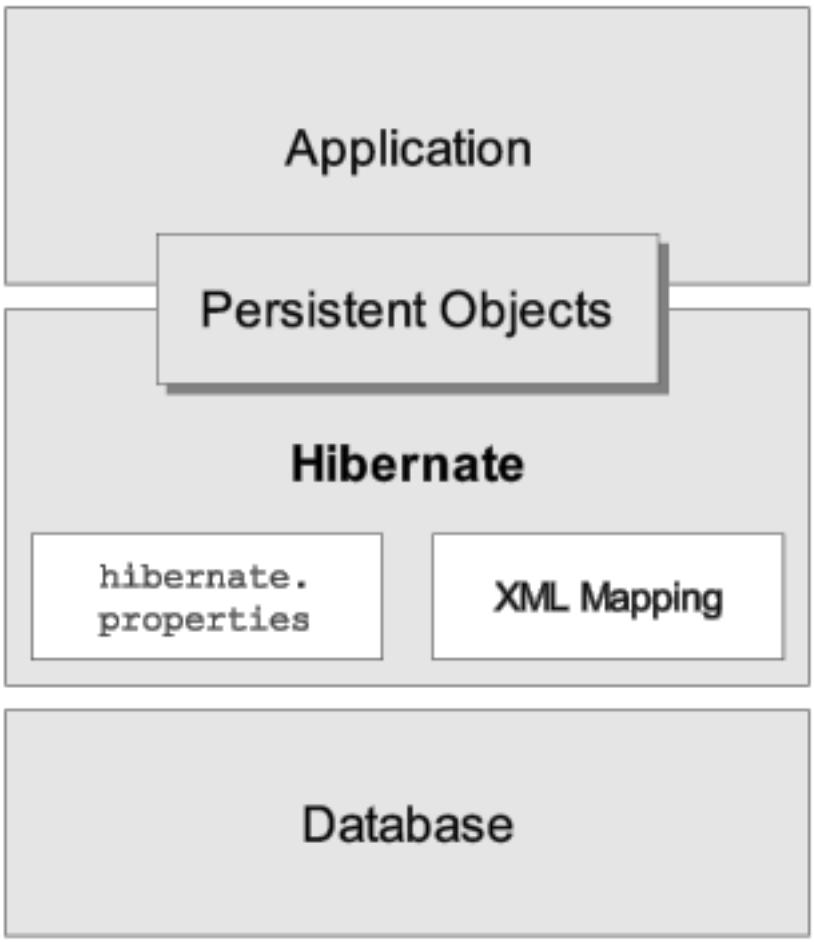
Introduction

Les applications d'entreprises s'orientent de plus en plus vers des architectures n-tiers. La technologie J2EE et les frameworks qui se sont créés autour offrent beaucoup d'outils pour répondre aux besoins modernes. Pour la couche présentation, on trouve, par exemple, le trés populaire Struts respectant le modèle MVC de séparation du code, de l'interface et des données. Pour ce type de couche, on trouve plus classiquement les JSP ou les trés récentes JSF qui permettraient de concurrencer directement le modèle initié par l'ASP.NET.

Nous allons, cependant, nous intéresser à une couche plus basse d'une architecture applicative, la couche d'accés aux données. Celle-ci permet d'interfacer le code métier avec une source de données. L'intérêt est de pouvoir changer de base de données en n'ayant besoin de ne modifier que la couche d'accés.

Il est possible d'écrire soit même les classes qui seront ensuite exposées au code métier mais c'est souvent fastidieux ou même maladroit à réaliser. Il vaut donc mieux utiliser un framework spécialisé dans cette tâche. De ce côté Dotnet a une faiblesse : OpenSpace, la technologie officielle de microsoft ne devrait sortir qu'en 2006. En attendant quelques solutions existent même s'ils ne disposent pas du même retour d'expérience que les outils J2EE. Pour la technologie de Sun, la spécification récente est JDO qui s'avère assez peu utilisée dans les fait. Le framework le plus populaire pour J2EE est sans contexte Hibernate.

Voici comment se présente trés globalement l'architecture d'Hibernate.



Source : hibernate.org

Ce type de technologie peut être appelé framework de **mapping objet-relationnel** ou de **persistance objet des données**.

En effet, la couche applicative voit les données comme des classes dont le contenu reste en mémoire même après la fin d'exécution du programme. D'où **persistance objet des données**. De plus, le lien entre les classes exposées et la source physique des données (souvent une base de données relationnelle) est définie par un fichier xml. D'où **mapping objet-relationnel**.

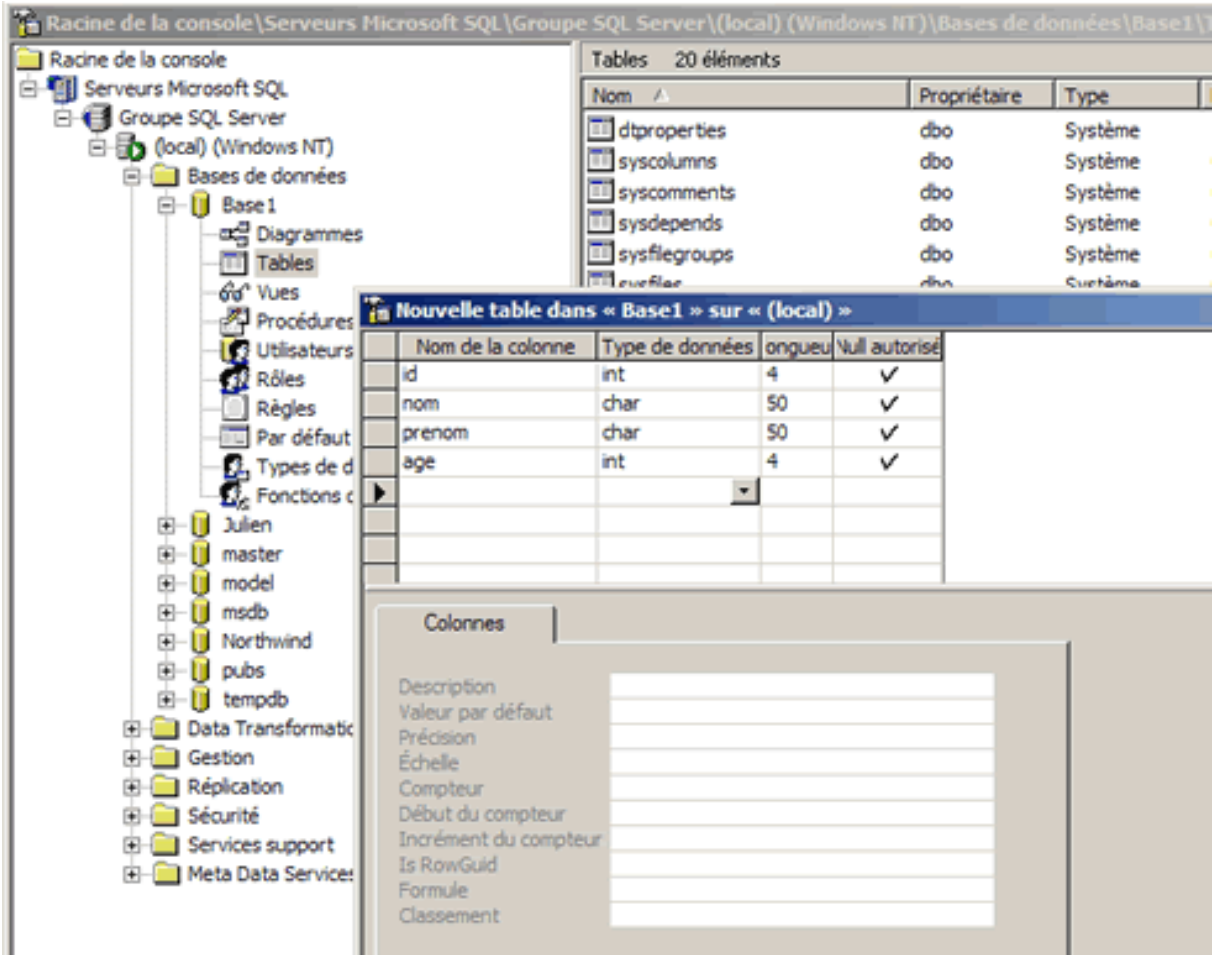
Cet article va présenter comment utiliser Hibernate sous Eclipse grâce à un plug-in appelé Hibernate Synchroniser. La base de données choisie est SQL Server 2000.

1 - Création de la base de données

La source de données choisie pour cet article est SQLServer 2000. Il est évidemment possible d'en choisir une autre comme Oracle, Mysql ou même un simple fichier XML pourvu que vous disposiez des drivers JDBC adaptés.

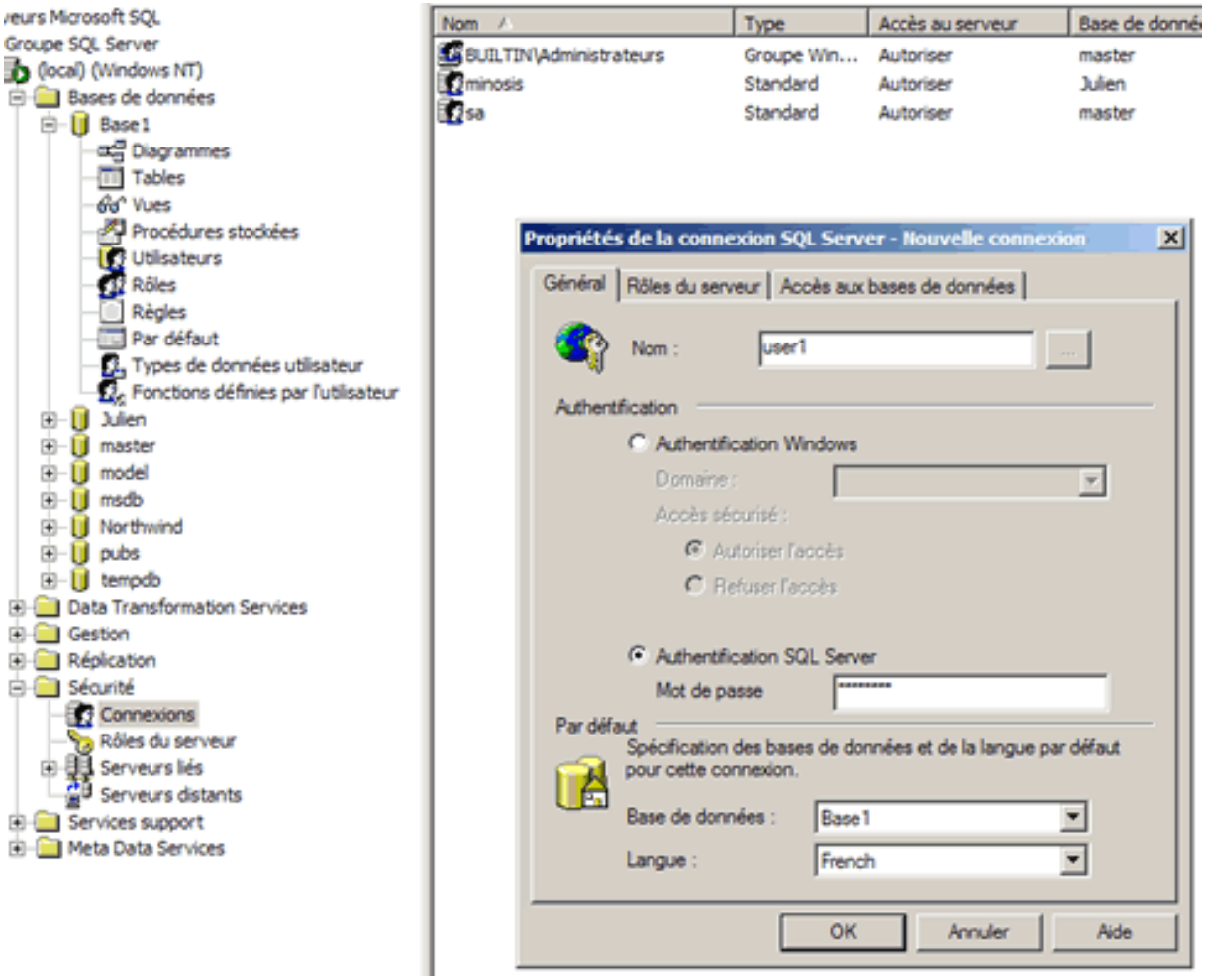
Nous supposons que SQLServer 2000 est installé et que vous disposez des droits nécessaires pour créer une nouvelle base et de nouveaux utilisateurs.

- Ouvrez Entreprise Manager et déroulez l'arborescence jusqu'à obtenir la liste des bases.
- Faites "Bouton droit" sur le répertoire "Bases de données" et cliquez sur "Nouvelle base de données ...". Nommez la base "Base1".
- Cliquez sur "Base1" qui vient d'apparaître dans l'arborescence puis faites "Bouton droit" sur "Tables" et cliquez sur "Nouvelle table ...".
- Insérez les informations suivantes :



Ne définissez pas le champs "id" en clé primaire mais comme un champs normal. En effet, le type Primary Key sera défini dans le fichier xml de mapping plus tard.

- Nommez cette table "t_contact".
- Déroulez le dossier "Sécurité" de l'arborescence et faites "Bouton droit" sur "Connexion" puis "Nouvelle connexion".
- Indiquez dans le premier onglet les informations suivantes :



Le mot de passe choisit ici est "password".

- Cliquez sur l'onglet "Accès aux bases de données", cochez "Base1" et donnez comme rôle "db_datareader" et "db_datawriter" uniquement.
- Double-cliquez sur "Base1 > Tables > t_contact" et cliquez sur "Autorisations". Cochez SELECT, INSERT, UPDATE et DELETE.

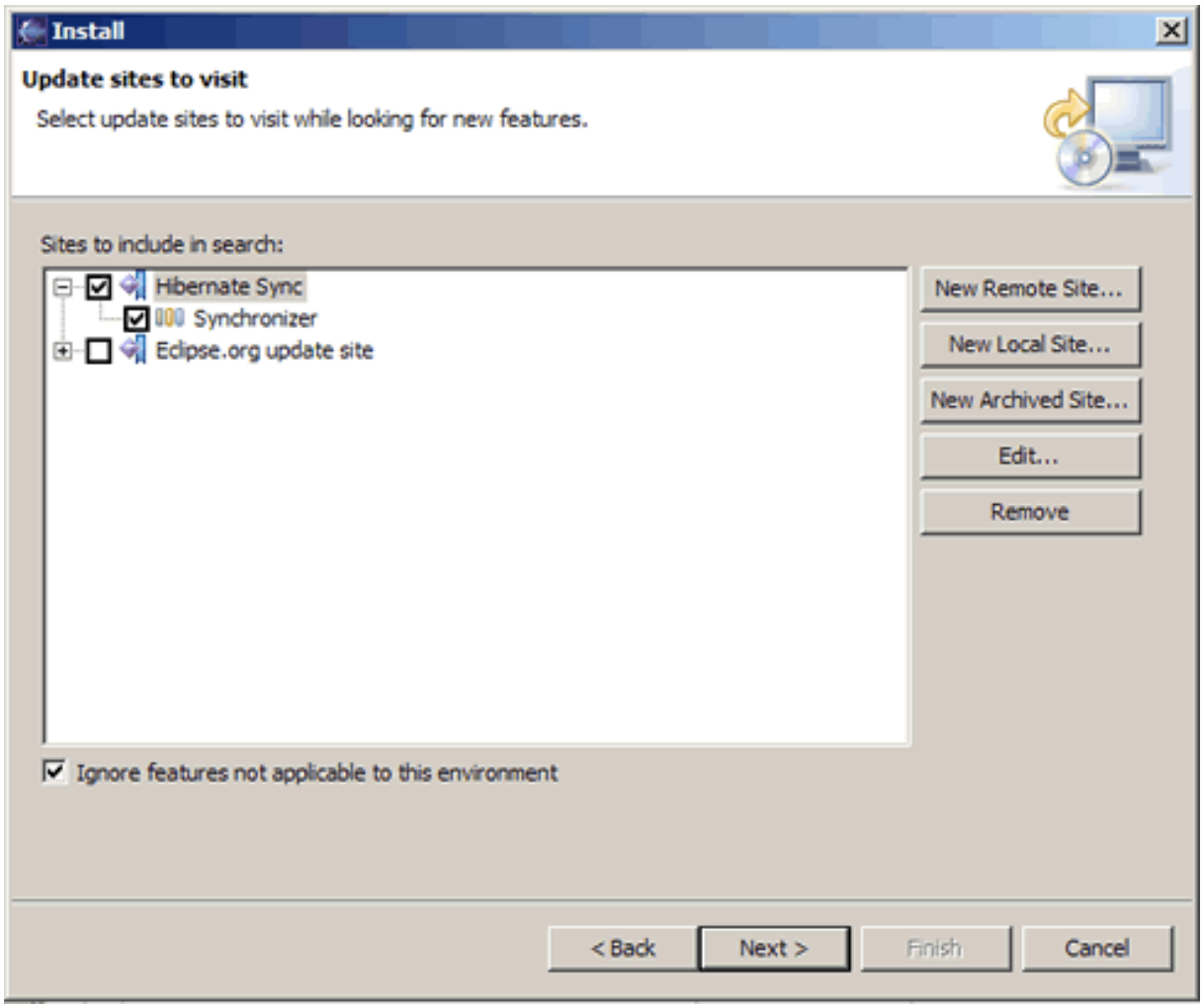
Votre base de données est maintenant prête. Vous pouvez, si vous le désirez, ajouter quelques enregistrements manuellement dans la table pour les futurs tests.

2 - Installation de Hibernate Synchronizer

- Téléchargez et installez J2SE 1.4 SDK.
- Si vous ne disposez pas d'Eclipse, téléchargez-le et installez le.

Si ces deux premiers points vous pose problème, consultez la page suivante : http://www.eclipsetotale.com/articles/installation.html

- Ouvrez Eclipse. Nous allons installer le plugin Hibernate Synchroniser.
- Cliquez sur "Help > Software Updates > Find and Install"
- Sélectionnez "Search for new features to install"
- Cliquez sur "New Remote Site" et entrez l'URL suivante : "http://www.binamics.com/hibernatesync". Donnez comme nom au site "Hibernate Sync".
- Cochez "Synchronizer" :



- Faites "Next", attendez que la recherche se fasse puis cochez "Hibernate Synchronizer". Continuez jusqu'au bout du processus en acceptant toutes les demandes.
- Eclipse devrait télécharger et installer automatiquement le plugin aprés avoir cliqué sur "Installer". Le programme vous demande ensuite de redémarrer. Cliquez sur "Yes".

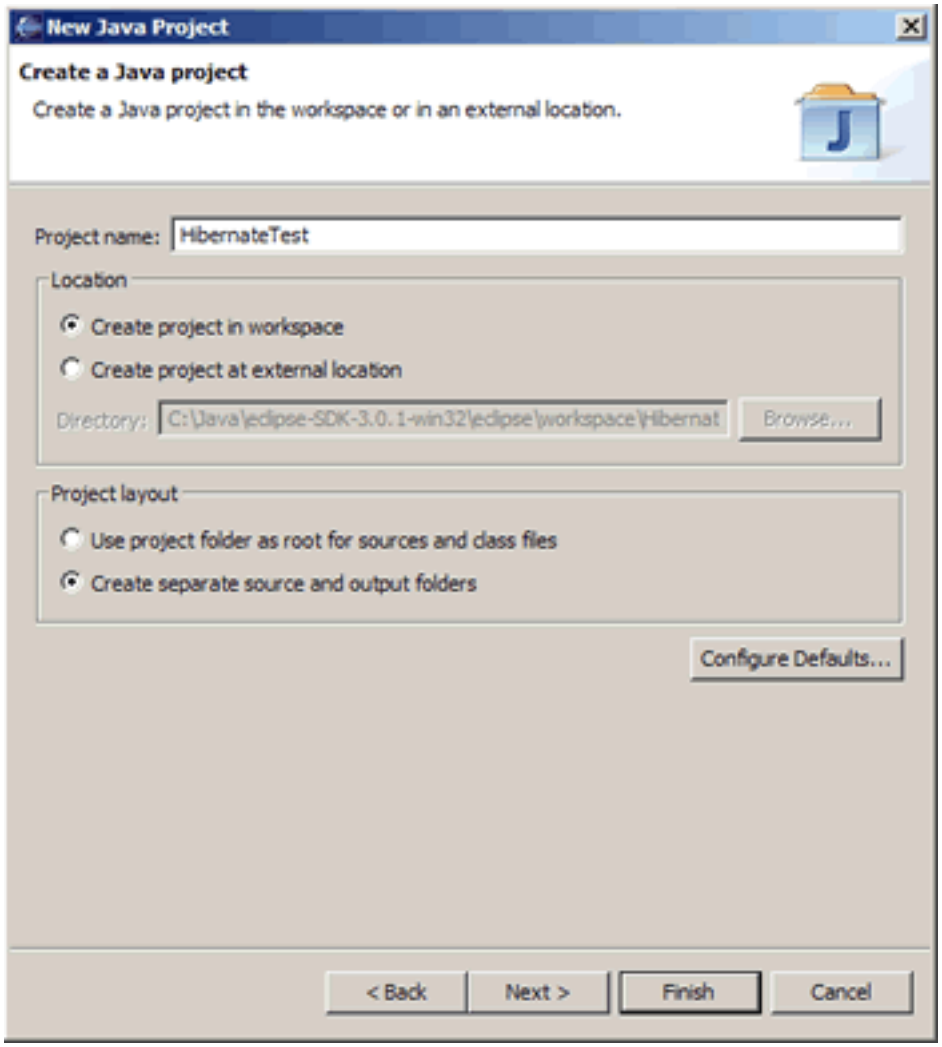
Si tout le processus s'est bien déroulé, Hibernate Synchronizer est installé.

3 - Utilisation de Hibernate

3.1 - Création et préparation du projet

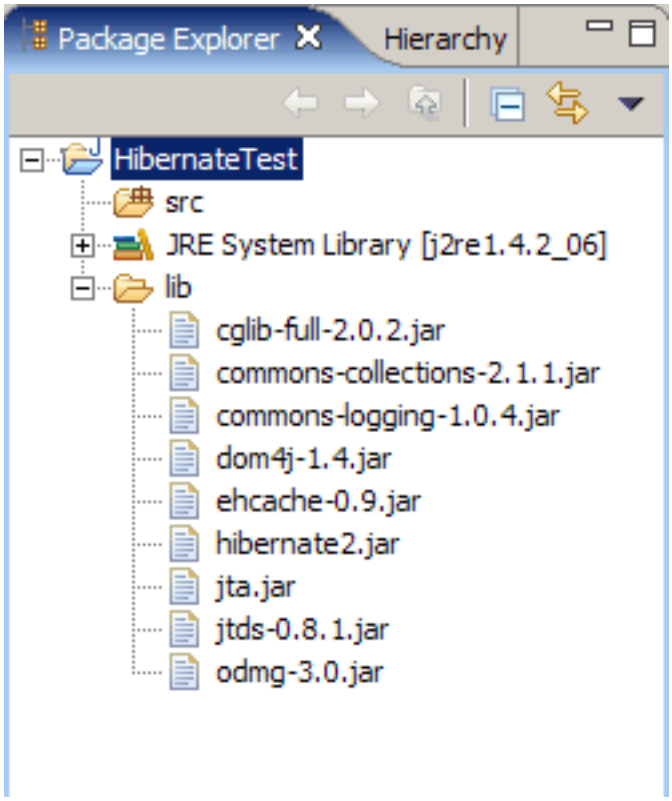
Avant de passer au coeur de l'utilisation d'Hibernate, nous devons créé et configurer un projet Eclipse. Cette partie est essentielle au bon fonctionnement de la suite.

- Faites "Fichier>New>Project"
- Choisissez "Java Project" puis donnez au projet les caractéristiques suivantes :



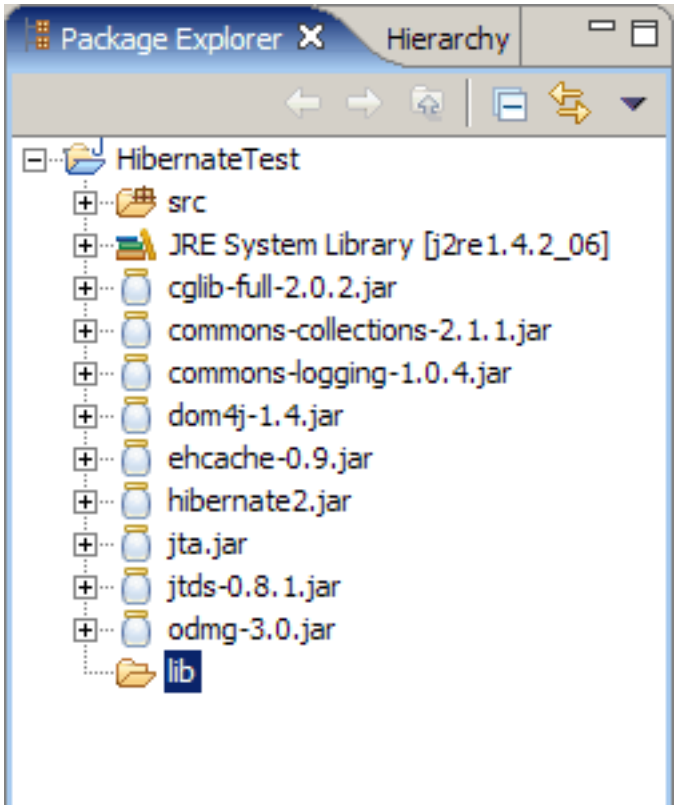
Veillez à bien cocher "Create separate source and output folders".

- Si Eclipse vous demande de passer en Java Perspective, répondez "Yes".
- Créez à partir de l'explorateur windows un répertoire "lib" dans la racine de votre projet : /eclipse/workspace/HibernateTest/lib
- Téléchargez l'archive lib_tuto.zip : ici. Décompressez son contenu dans le répertoire "lib". hibernate2.jar contient les principaux packages nécessaires au bon fonctionnement d'Hibernate. jtds-0.8.1.jar est un driver JDBC opensource pour se connecter à SQLServer 2000. Les autres packages sont indispensables à Hibernate.
- Revenez à Eclipse et faites sur le nom du projet "HibernateTest" : "Bouton droit > Refresh". Le répertoire "lib" a dû apparaître.
- A ce moment précis, si vous déroulez le répertoire "lib", le projet doit ressembler à ça :



Nous allons maintenant ajouter au classpath du projet les librairies :

- Faites sur le projet "HibernateTest" : "Bouton Droit > Properties"
- Cliquez sur "Java Build Path" puis sur l'onglet "Librairies".
- Cliquez sur "Add jars". Une fenêtre apparaît. Déroulez l'arborescence qui soit être "HibernateTest > lib" et sélectionnez tous les jar présents. Puis cliquez sur "OK".
- Tous les packages devraient apparaître dans la liste de "Librairies". Cliquez sur "OK" dans la fenêtre "Properties for HibernateTest".
- Le projet doit être maintenant comme ceci :



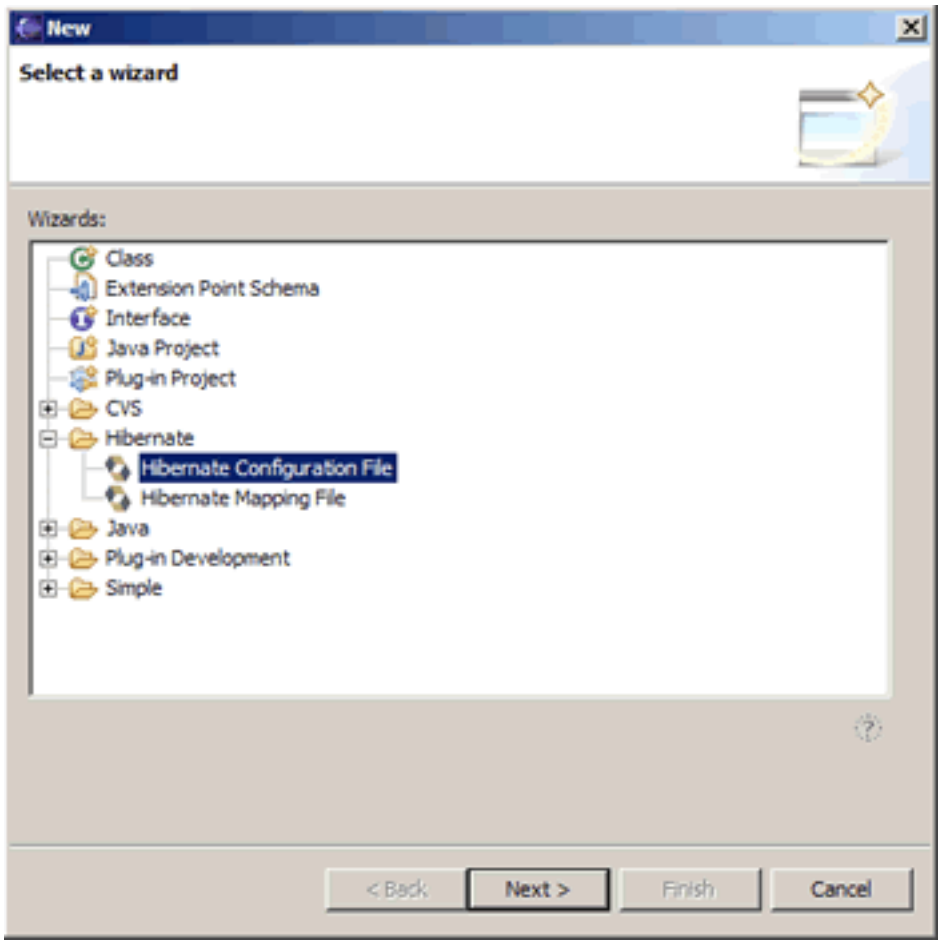
Nous sommes prêt à passer aux choses sérieuses ;-)

3.2 - Création des éléments de Hibernate

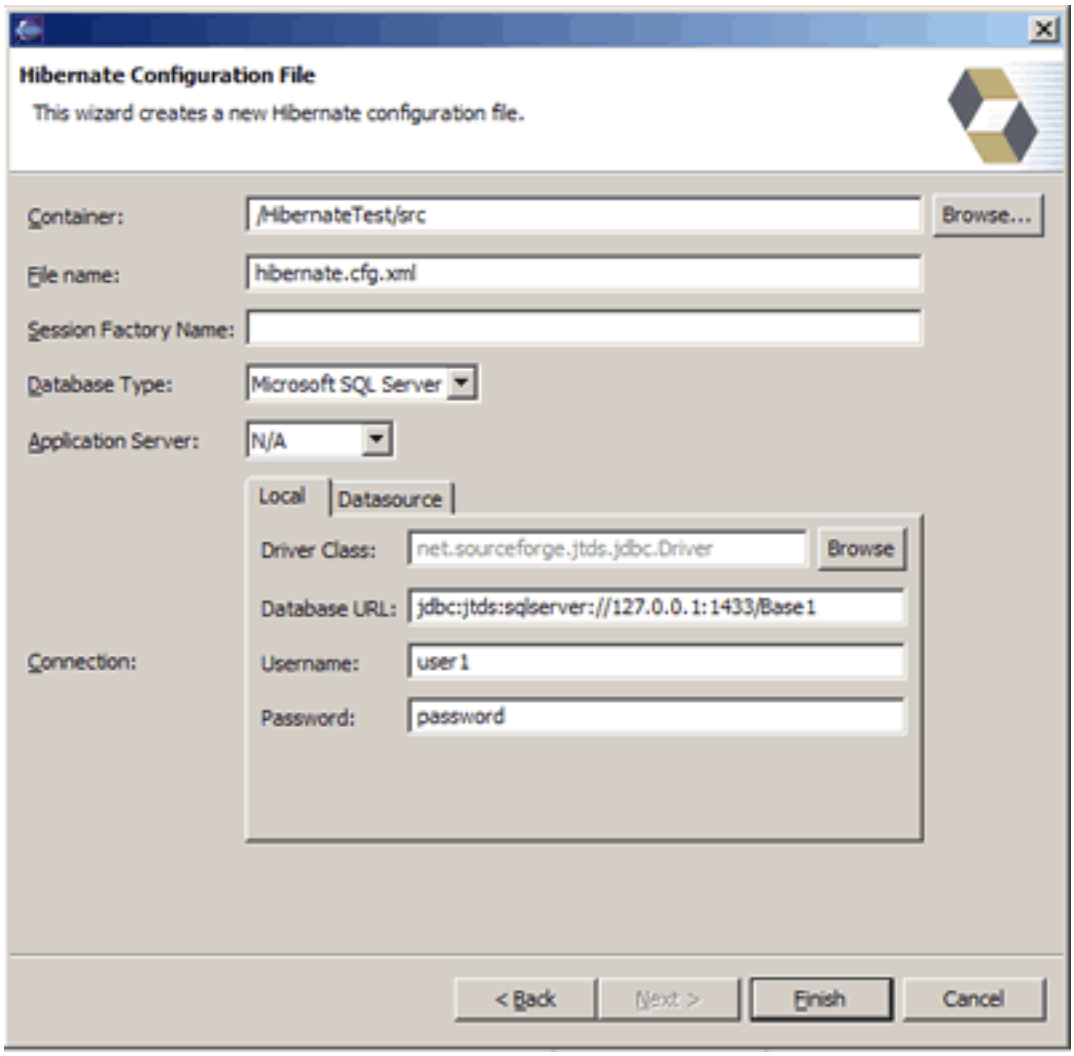
3.2.1 - Création du fichier de configuration xml

Le fichier que nous allons créer va permettre d'indiquer au moteur d'Hibernate les paramètres de connexion à la base de données.

- Sur le dossier "src", faites "Bouton droit > New > Other" et choissez "Hibernate > Hibernate Configuration File"



- Dans la fenêtre qui apparaît ensuite, indiquez les informations suivantes :



Remarque 1 : si plus tard, hors du contexte de ce tutorial, vous deviez utiliser Hibernate dans un serveur d'application J2EE, vous devriez le choisir dans la liste "Application Server".

Remarque 2 : le nom "net.sourceforge.jtds.jdbc.Driver" doit être entré à la main (aprés avoir cliqué sur "Browse"). En effet, le moteur de recherche ne le trouvera pas sinon.

Remarque 3 : on suppose que l'instance SQLServer est sur la même machine (127.0.0.1). Si ce n'est pas le cas, entrez la bonne adresse ip.

- Cliquez sur "Finish". hibernate.cfg.xml est créé.
- Le fichier devrait s'ouvrir dans votre éditeur xml par défaut. Le mieux est de l'ouvrir sous Eclipse : "Bouton droit sur le fichier > Open With > Text Editor".

Nous allons réaliser un exemple de test au prochain chapitre qui impose des modifications de

hibernate.cfg.xml. En effet, ce fichier est destiné à être utilisé sur un serveur d'application. Nous allons le rendre utilisable dans un contexte plus simple :

- Mettez ces lignes en commentaire :

```
<!--
<property name="hibernate.transaction.factory_class">
	net.sf.hibernate.transaction.JTATransactionFactory
</property>
<property name="jta.UserTransaction">
	java:comp/UserTransaction
</property>
-->
```

- Placez en dessous la nouvelle ligne suivante :

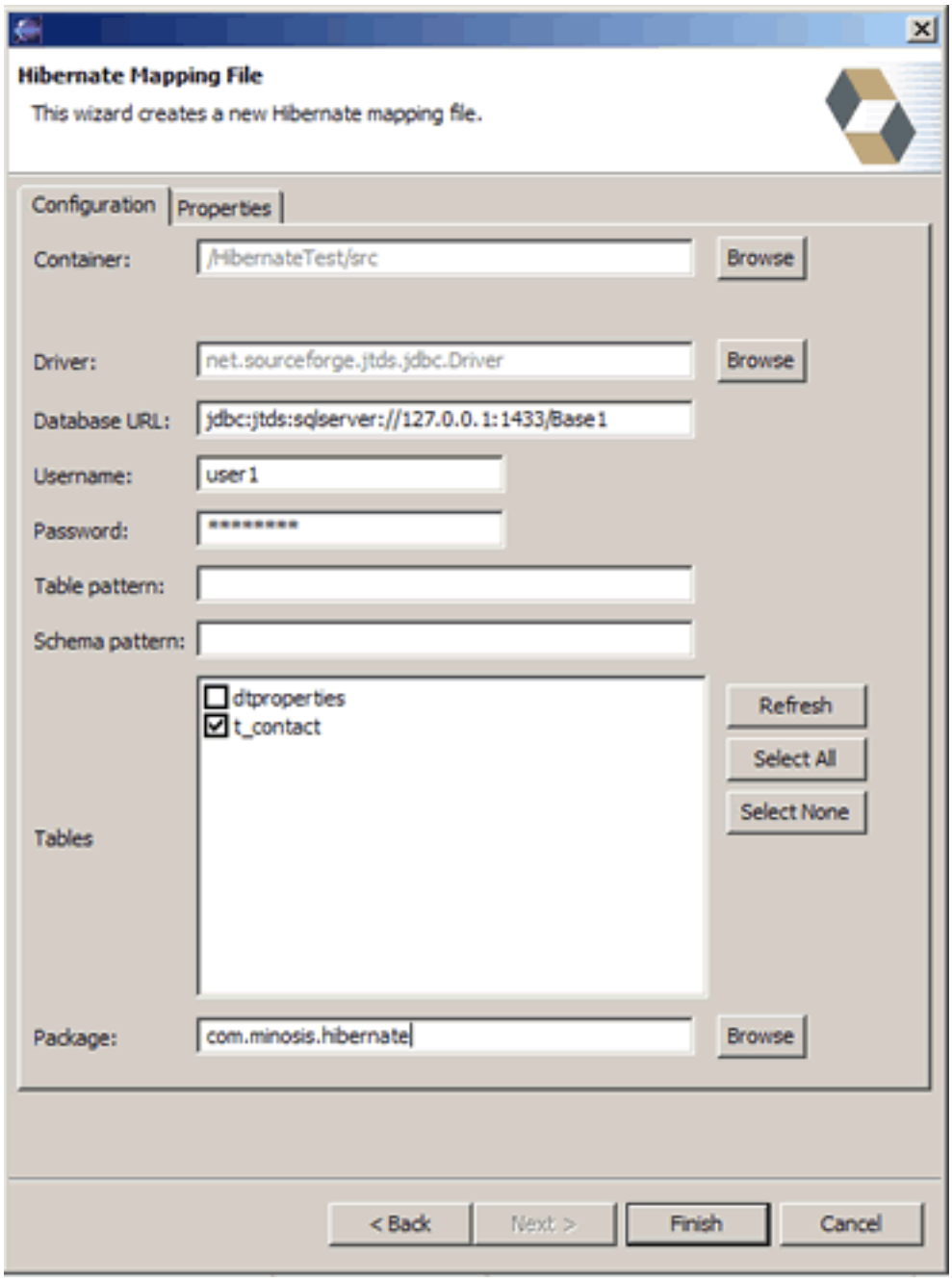
```
<!--
<property name="hibernate.transaction.factory_class">
	net.sf.hibernate.transaction.JTATransactionFactory
</property> <property name="jta.UserTransaction">
	java:comp/UserTransaction
</property>
-->
<property name="hibernate.transaction.factory_class">
  net.sf.hibernate.transaction.JDBCTransactionFactory
</property>
```

Enregistrez le fichier, il est maintenant correct.

3.2.2 - Création du fichier de mapping xml

Ce fichier est un élément majeur puisqu'il va permettre à Hibernate faire le pont entre les classes de persistance et la source de données.

- Faites sur "src" dans le projet : "Bouton droit > New > Package" et donnez comme nom, par exemple : "com.minosis.hibernate".
- Sur le dossier "src", faites "Bouton droit > New > Other" et choissez "Hibernate > Hibernate Mapping File"
- Indiquez les informations suivantes :



Pour obtenir automatiquement la liste des tables de la base, mettez "password" dans Password et cliquez sur "Refresh". Si les tables apparaîssent, c'est que la connexion à la base de données a eu lieu correctement, félicitation ;-)

Cliquez sur "Finish". Notre fichier de TContact.hbm est créé.

A ce moment, on peut supposer le fichier conforme. Il n'en est rien, le plugin Hibernate Synchronizer a généré un fichier non conforme à la DTD, hibernate-mapping-2.0.dtd.

Je vous propose donc de remplacer son contenu par celui-ci :

```
<?xml version="1.0"?>
  <!DOCTYPE hibernate-mapping
 PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

 <class name="com.minosis.hibernate.TContact" table="t_contact">

 <id name="Id" type="integer">
 <column name="id" sql-type="int(4)"/>
 <generator class="increment" />
 </id>

 <property name="Nom" type="string">
 <column name="nom" sql-type="char(50)" not-null="true"/>
 </property>

 <property name="Prenom" type="string">
 <column name="prenom" sql-type="char(50)" not-null="true"/>
 </property>

 <property name="Age" type="integer">
 <column name="age" sql-type="int(4)" not-null="true"/>
 </property>

 </class>

</hibernate-mapping>
```

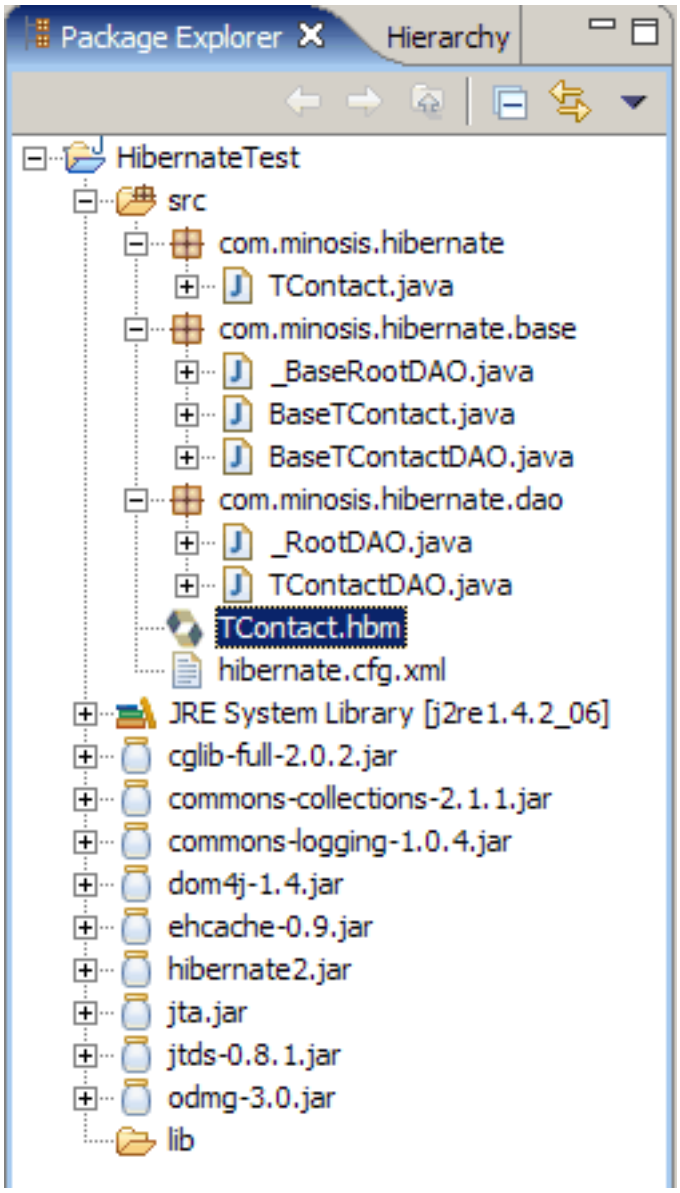
Tout n'était pas faux dans le précédent, mais celui-ci est plus conforme aux préconisations de l'aide officielle d'Hibernate et ne créera pas d'erreur lorsqu'il sera parsé.

Il est relativement simple à comprendre. <id> défini la clé pirmaire et <generator class="increment" />, indique le comportement de génération automatique de la clé primaire (ici incrément simple). Les <property> déclarent les autres champs. La balise <class> informe du nom de l'objet et du nom de la table correspondante.

Enregistrez le fichier. Le plugin génère alors automatiquement des classes. Si ce n'est pas le cas, faites sur le fichier TContact.hbm : "Bouton droit > Hibernate Synchroniser > Synchronize File".

Vous pouvez aller regarder les classes générées, surtout BaseTContact.java qui contient la traduction objet de la table "t_contact". C'est plutôt intéressant.

Voici l'allure du projet avec tous les packages déroulés :



Tous les fichiers nécessaires au mapping sont maintenant créés.

Le chapitre suivant propose de montrer comment les utiliser à partir de ce qui pourrait être la couche applicative.

4 - Test des classes

Maintenant que des classes ont été générées grâce à Hibernate et Hibernate Synchronizer, nous pouvons les tester.

4.1 - Classe HibernateUtil

Une classe Hibernate appellée SessionFactory permet à partir du fichier de configuration (hibernate.cfg.xml) d'être associé à la source de données. Elle fournit des objets Session pour manipuler les données. Cependant, en général, les serveurs d'application et les conteneurs de servlets, exécutent dans plusieurs threads le même code. En utilisation directe, SessionFactory serait donc instancié autant de fois qu'il y a de threads. Chaque instance se base pourtant sur le même fichier de configuration. Il est donc plus adapté de rendre une même instance de SessionFactory accessible par les threads. Nous allons, pour celà, écrire une classe trouvée dans l'aide en ligne d'Hibernate.org.

- Dans votre projet HibernateTest, faites sur "com.minosis.bdd" : "Bouton droit > New > File". Donnez-lui comme nom "HibernateUtil.java".
- Copiez dans ce fichier vide, le contenu suivant :

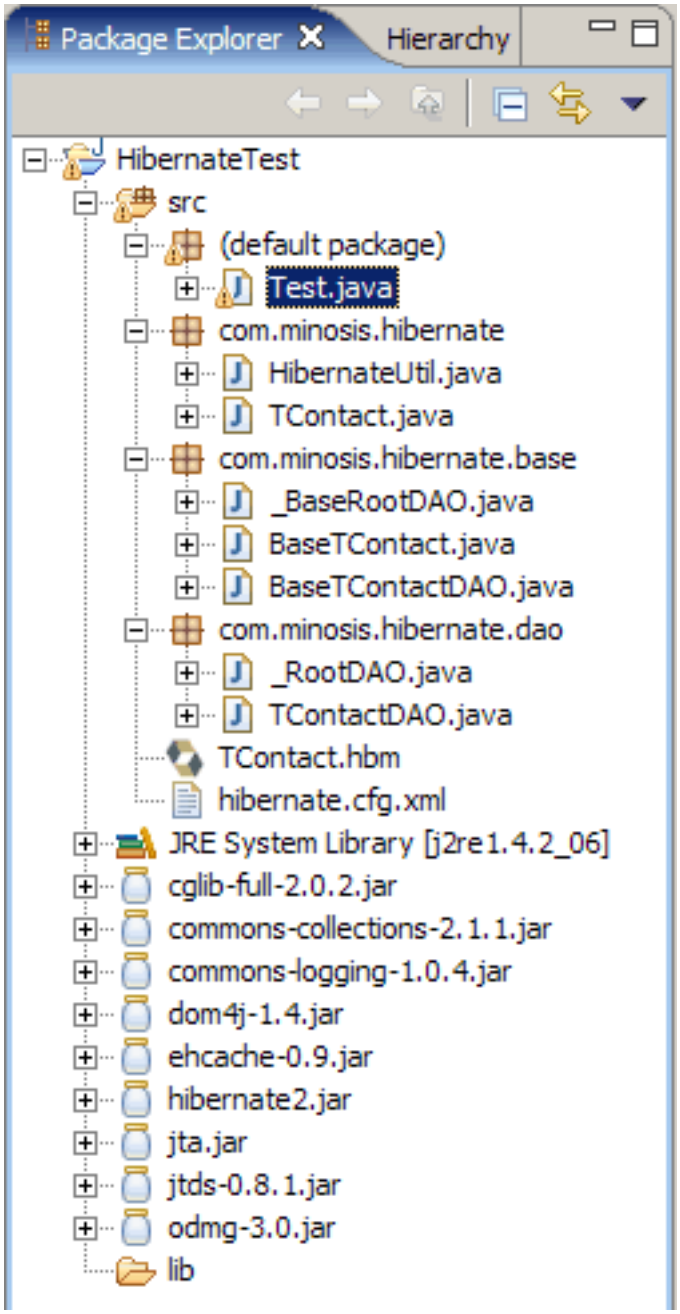
```
package com.minosis.hibernate;

import net.sf.hibernate.*;
import net.sf.hibernate.cfg.*;

public class HibernateUtil {

 private static final SessionFactory sessionFactory;

 static {
  try {
  // Crée la SessionFactory
  sessionFactory =
       new Configuration().configure().buildSessionFactory();
  } catch (HibernateException ex) {
  throw new RuntimeException("Problème de configuration : "
  + ex.getMessage(), ex);
  }
  }

 public static final ThreadLocal session = new ThreadLocal();

 public static Session currentSession()
             throws HibernateException {
  Session s = (Session) session.get();
  // Ouvre une nouvelle Session, si ce Thread n'en a aucune
  if (s == null) {
  s = sessionFactory.openSession();
  session.set(s);
  }
  return s;
  }

 public static void closeSession()
             throws HibernateException {
  Session s = (Session) session.get();
  session.set(null);
```

```
        if (s != null)
        s.close();
        }
    }
```

4.2 - Classe Test

Nous allons, avec cette classe, tester la lecture, la modification et l'insertion de données dans "t_contact".

- Faites sur le répertoire "src" : "Bouton droit > New > File". Nommez le fichier "Test.java".

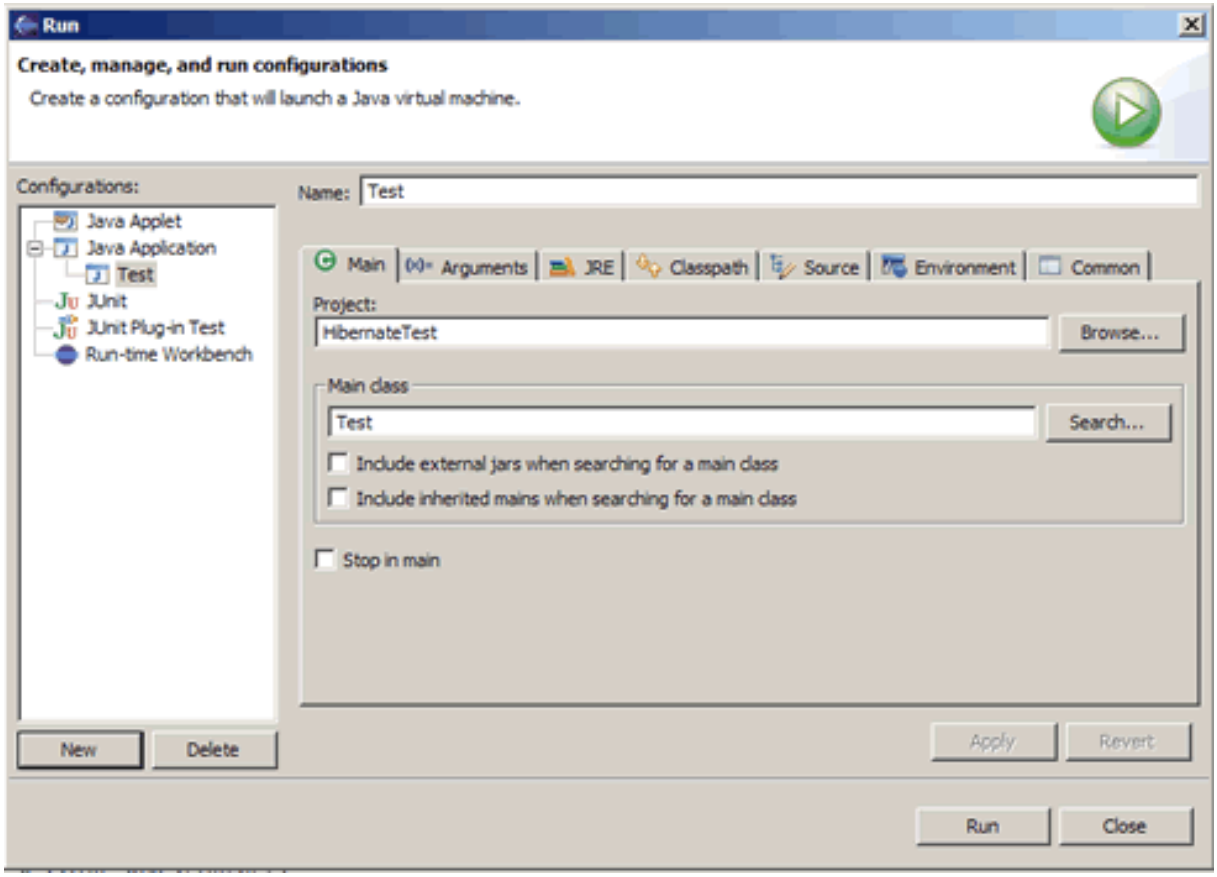


- Placez dans Test.java, le code suivant :

```
import java.util.*;
import net.sf.hibernate.*;
import com.minosis.hibernate.*;

public class Test {
```

```
    public static void main(String[] args)
        throws HibernateException {

    }
}
```

- Pour exécuter le programme (qui ne fait rien pour le moment), cliquez sur "Run > Run...". Puis sélectionnez "Java Application" et cliquez sur le bouton "New".



Si vous désirez démarrer de suite le programme, cliquez sur "Run" sinon vous pourrez le faire plus tard, cliquez sur "Close".

4.2.1 - Insertion

- Placez le code suivant dans la méthode main() :

```
Session session = HibernateUtil.currentSession();

Transaction tx = session.beginTransaction();
```

```
TContact contact = new TContact();
contact.setNom("Dupont");
contact.setPrenom("Jean");
contact.setAge(new Integer(44));
session.save(contact);

contact = new TContact();
contact.setNom("Lambert");
contact.setPrenom("Julie");
contact.setAge(new Integer(27));
session.save(contact);

tx.commit();

HibernateUtil.closeSession();
```

On crée une nouvelle Session à partir du SessionFactory static donné par HibernateUtil.

Une transaction est démarrée pour pallier à d'éventuels problèmes lors de l'insertion.

Jean Dupont est inséré ensuite, grâce à la classe TContact. Puis la transaction est commitée avant la fermeture de la session.

4.2.2 - Mise à jour

- Placez le code suivant dans la méthode main() (on la suppose vide) :

```
Session session = HibernateUtil.currentSession();

Transaction tx = session.beginTransaction();


TContact contact =
(TContact) session.load(TContact.class, new Integer(1));

contact.setPrenom("Jacques");
session.save(contact);

tx.commit();

HibernateUtil.closeSession();
```

Aprés le démarrage de la transaction, la classe persistante TContact correspondant à l'id=1 est chargée par session.load().

Le premier enregistrement est donc chargé et modifié. Si vous avez déjà exécuté le code d'insertion, "Jean" sera remplacé par "Jacques".

La méthode de sélection de l'enregistrement n'est pas vraiment parfaite. Rassurez vous, l'objet session fournit beaucoup de possibilité de filtre. Nous en verrons une dans l'exemple de lecture.

4.2.3 - Lecture

- Placez le code suivant dans la méthode main() (on la suppose vide) :

```
List list = session.find("from TContact where nom like '%t'");
Iterator it = list.iterator();
while(it.hasNext())
{
  TContact contact = (TContact)it.next();
  System.out.println(contact.getNom());
}

HibernateUtil.closeSession();
```

session.find() nous retourne une liste des enregistrements trouvés par la requête HQL. On lui demande ici de retourner tous les enregistrements trouvés dans un type List.

Il ne reste plus qu'à afficher les informations que l'on veut.

- Cliquez sur l'icône "Run" pour tester ces exemples.

Les exemples ici sont trés simplistes mais permettent au moins d'avoir une vue globale du fonctionnement d'Hibernate. Le framework est évidemment beaucoup plus puissant.

Conclusion

Cet article vous a essentiellement montré comment démarrer avec Hibernate, comment ne pas rencontrer les premiers problèmes de compilation ou de parsing xml.

Cependant l'exemple se base sur une unique table de 4 champs et ne permet pas du tout de réellement connaître Hibernate. Le framework est trés puissant et finalement peu complexe grâce à des plugins comme Hibernate Synchroniser. Hibernate est capable de gérer finement les jointures entre les tables, la génération des clés primaires, la conversion de types. Ceci se fait simplement dans les fichiers de mapping xml. De même, Hibernate fournit un langage de requétage efficace, le HQL. Je vous invite à consulter la documentation officielle en français sur le site Hibernate.org. Elle est trés bien faite et vraiment fournie.

En bref, ce type d'outil optimise le temps de développement du programmeur et permet de réaliser des applications plus homogènes, plus facilement migrables aussi (pensez que pour changer de base de données, il suffit juste de toucher à hibernate.cfg.xml).

Téléchargez le projet Eclipse de ce tutorial. Pour l'utiliser, il vous faudra au moins réaliser les deux premiers chapitres.

Pour information, il existe un début de migration de Hibernate pour Dotnet, appelé nHibernate.