Université du littoral Côte d'Opale
Master Info
Module : BDOO
Chapitre : ORM (Hibernate)

TP Hibernate

Partie I : Installation des logiciels nécéssaires au TP

Pour les besoins de ce TP vous aurez besoin des logiciels suivants :

1. *Eclipse (de préférence une version supérieure ou égale à 3.2)*
2. *Hibernate-3.2 : fichier (**hibernate-3.2.5.ga.zip**)*
3. *Mysql*
4. *Hibernate Synchornizer (fichier **HibernateSynchronizer-3.1.9.zip**)*
5. *Mysql Connector J (fichier **mysql-connector-java-3.1.14.tar.gz**)*

Prérequis des installations :
Il faut absolument que Eclipse et MysQL soient déjà installés sur votre machine et que vous ayez les droits vous permettant de configurer ces installations.

Installation de Hibernate-3.2 :

*Aucune manipulation n'est nécessaire (exceptée la décompression du fichier). Décompresser le fichier fourni **hibernate-3.2.5.ga.zip** quelque part dans votre esapce de travail. Le répertoire ainsi obtenu (on l'appellera par la suite **HIBERNATE**) sera utilisé comme un ensemble de librairies. Ce sont notamment les librairies hibernate ainsi que quelques librairies annexes qui seront nécessaires à vos développement Java.*

Installation de MysQl Connector J

*Idem que pour Hibernate-3.2. Ce logiciel se présente également comme une librairie qui sera nécessaire pour qu'une application Java puisse se connecter à MySql. Donc décompresser le fichier **mysql-connector-java-3.1.14.tar.gz** dans un endroit de votre espace de travail qu'on appellera par la suite : **CONNECTOR***

Installation de Hibernate Synchornizer

Hybernate Synchronizer est un des plugins Eclipse simplifiant le développement d'applications Java Hibernate. Son installation est toute simple.

*Reprérer d'abord le répertoire des extensions Eclipse (c'est un peu le répertoire des plugins). Ce répertoire qu'on appellera **EXTENSION** est simple à trouver. C'est un répertoire qui doit s'appeller eclipse et qui contient un sous-répertoire plugins.*
*Alors décompresser le fichier **HibernateSynchronizer-3.1.9.zip** dans ce répertoire (je veux dire EXTENSION).*

A ce stade nous avons tout ce qui nous faut pour commencer à travailler aec Hibernate et Eclipse en utilisant le SGBD MySQL.

Partie 2 : Création de la base de données relationnelle

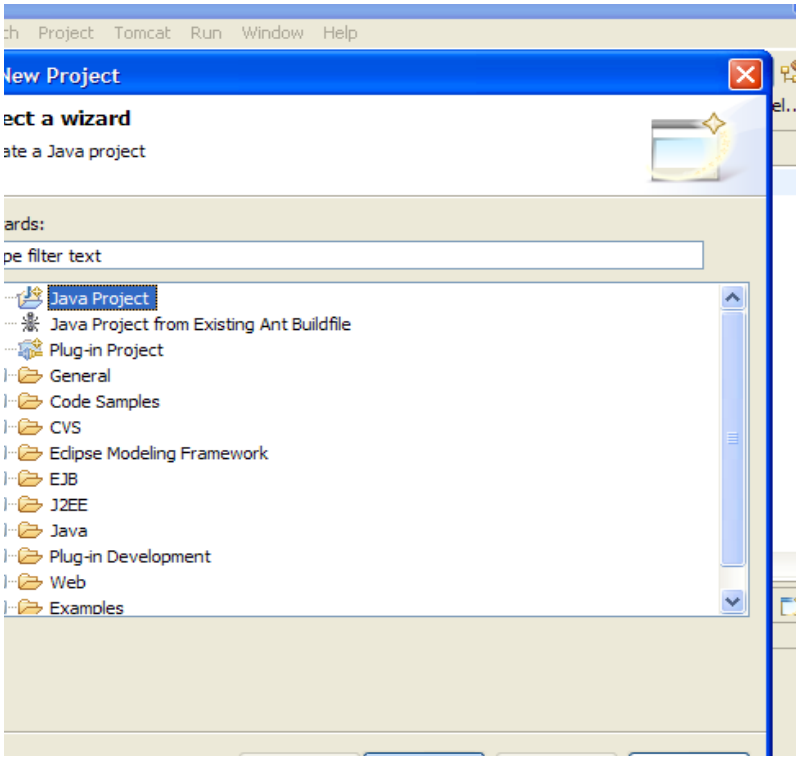
Nous allons créer et instancier une base de données MySql que nous traiterons par la suite à partir de classes Java et en utilisant hibernate.
Créer une base ou utiliser une base courante. Dans cette base et en utilisant un shell MySQL exécuter les commandes SQL suivantes :

```
#
# Structure for the `groupes` table :
#
CREATE TABLE `groupes` (
 `idgroupe` int(4) NOT NULL auto_increment,
 `nomgroupe` varchar(50) default NULL,
 `commentairegroupe` varchar(150) default NULL,
 PRIMARY KEY (`idgroupe`),
 UNIQUE KEY `idgroupe` (`idgroupe`)
) TYPE=InnoDB;

#
# Structure for the `personnes` table :
#
CREATE TABLE `personnes` (
  `idpersonne` int(11) NOT NULL auto_increment,
  `nompersonne` varchar(50) default NULL,
  `prenompersonne` varchar(50) default NULL,
  `datenaisspersonne` datetime default NULL,
  `coeffpersonne` int(11) default NULL,
  PRIMARY KEY  (`idpersonne`),
  UNIQUE KEY `idpersonne` (`idpersonne`)
) TYPE=InnoDB;

#
# Structure for the `grppers` table :
#
CREATE TABLE `grppers` (
  `idgrppers` int(11) NOT NULL auto_increment,
  `idgroupe` int(11) default NULL,
  `idpersonne` int(11) default NULL,
  PRIMARY KEY (`idgrppers`),
  UNIQUE KEY `idgrppers` (`idgrppers`),
  KEY `idgroupe` (`idgroupe`),
  KEY `idpersonne` (`idpersonne`),
  CONSTRAINT `0_48` FOREIGN KEY (`idpersonne`) REFERENCES `personnes`
(`idpersonne`),
  CONSTRAINT `0_45` FOREIGN KEY (`idgroupe`) REFERENCES `groupes` (`idgroupe`)
) TYPE=InnoDB;

INSERT INTO `groupes` (`idgroupe`, `nomgroupe`, `commentairegroupe`) VALUES
  (1,'groupe 1',NULL),
  (2,'groupe 2',NULL);

INSERT INTO `grppers` (`idgrppers`, `idgroupe`, `idpersonne`) VALUES
  (1,1,1),
  (2,2,2),
  (3,2,3),
  (4,1,4),
  (5,1,5);

INSERT INTO `personnes` (`idpersonne`, `nompersonne`, `prenompersonne`,
                         `datenaisspersonne`, `coeffpersonne`) VALUES
  (1,'nom1','prenom1','1967-01-06',123),
  (2,'nom2','prenom2','1973-08-11',34),
  (3,'nom3','prenom3','1956-04-28',145),
  (4,'nom4','prenom4','1980-12-02',23),
  (5,'nom5','prenom5','1966-10-13',119);
```

Partie 3: création du projet Eclipse

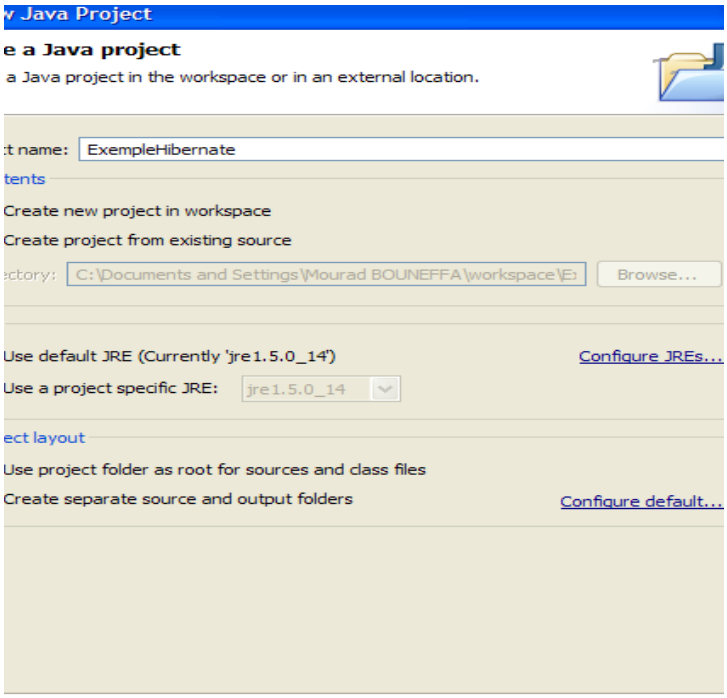
Créer un projet Eclipse de type Java Poojject ayant comme répertoire des sources un répertoire appelé ***src*** *et comme répertoire des binaires un répertoire* ***bin****.*
Pour ce faire il suffit d'exécuter les actions suivantes sous Eclipse :

File -->New --> Project

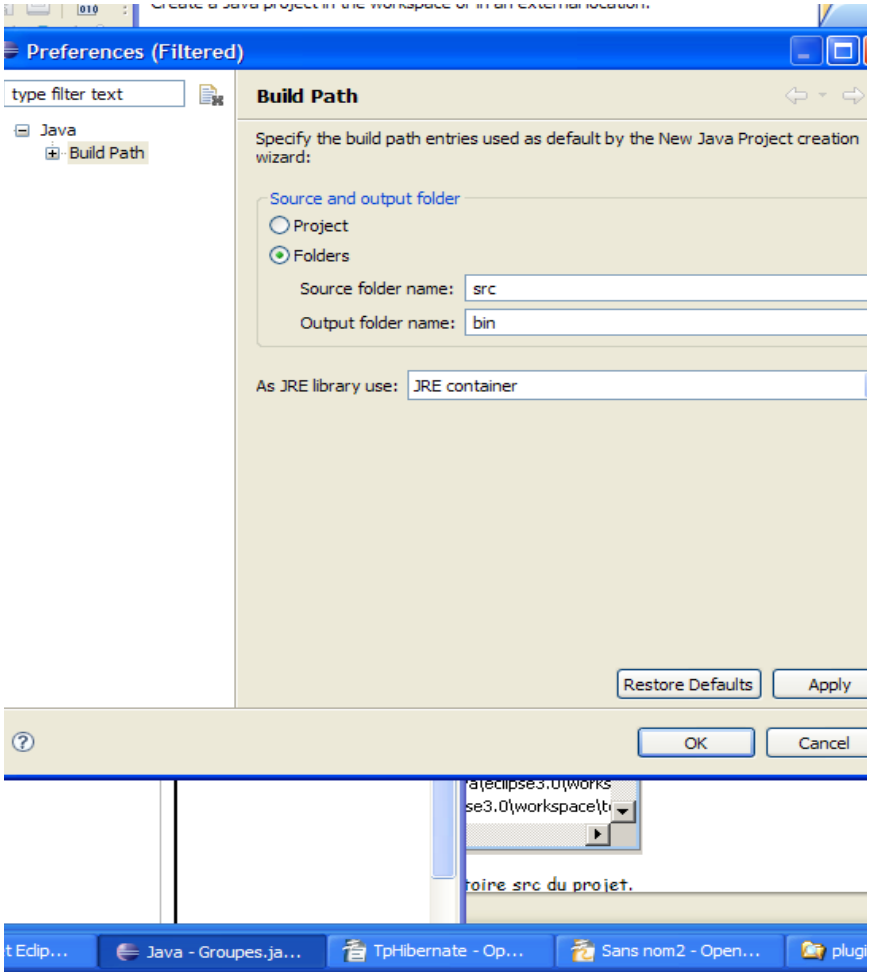


Choisir Java Project puis Next

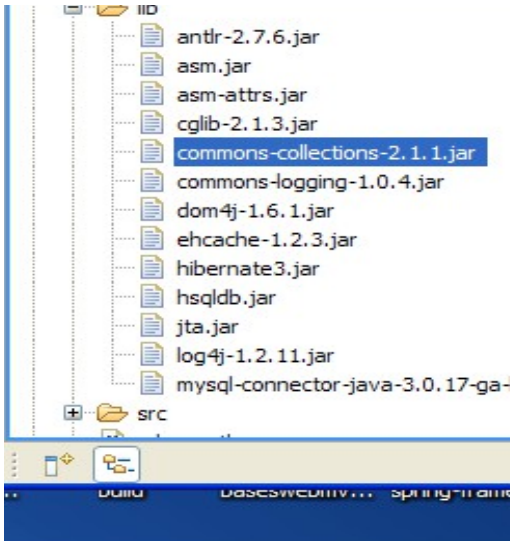
Donner un nom au projet.
Dans Project Layout cliquer sur configure defaut



pour source and output folder sélctionner folders puis src et bin pour les sources et les binaires.



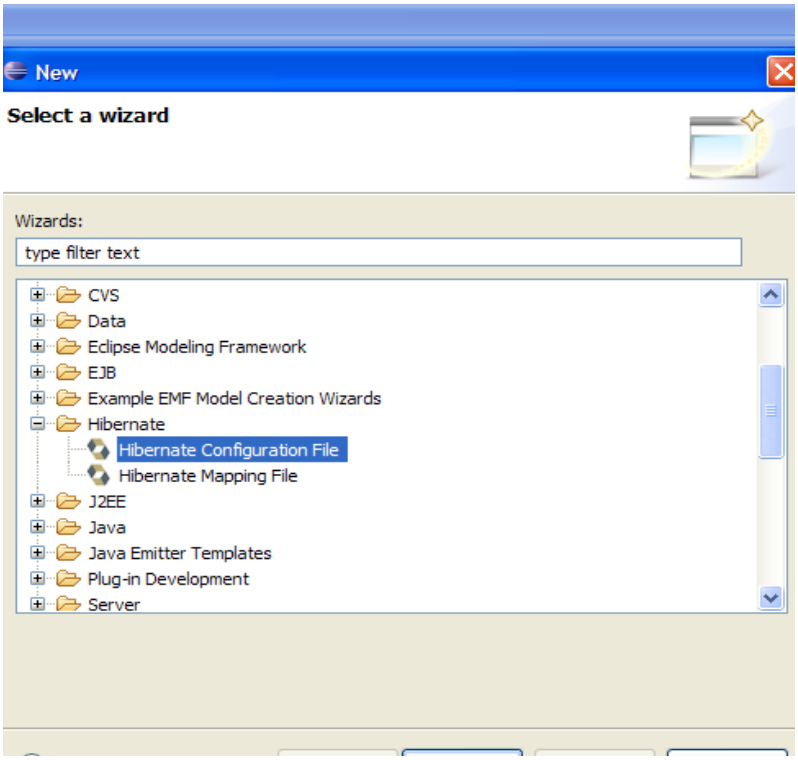
Cliquer sur OK puis finish.
Dans la racine du projet créer un répertoire lib et le remplir avec les fichier suivants :



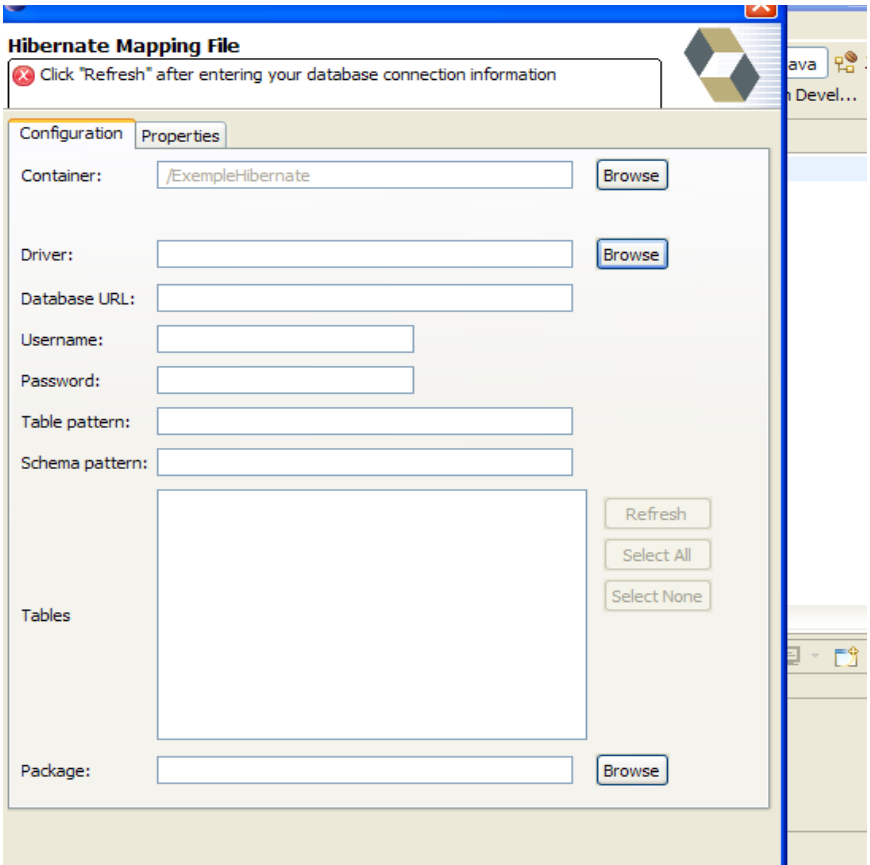
Ces fichiers proviennent tous du répertoire : hibernate-3.2/lib à l'exception de :
hibernate3.jar qui provient de la racine du répertoire hibernate (hibernate-3.2) et mysql-connector-java-3.0.17-ga-bin.jar qui provient de l'archive mysql connector (le driver mysql pour

java). Après avoir créer et rempli le répertoire lib intégrer les librairies qu'il contient au projet Eclipse.
Il suffit d'accéder aux propriétés du projet (Menu contextuel puis propriétés ou Properties puis Java Build Path puis Add jar et ajouter les jar du répertoire lib).

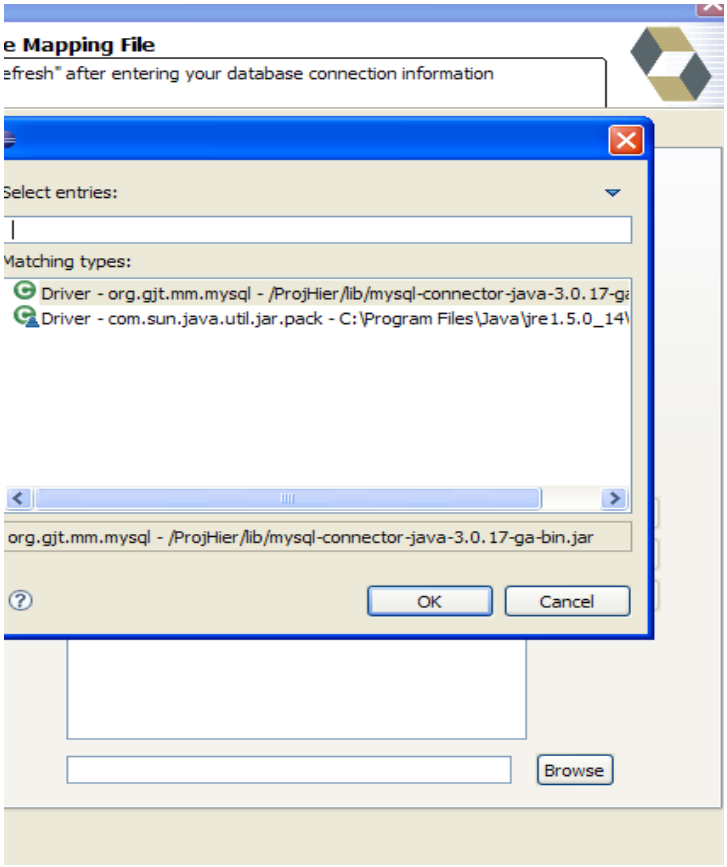
Partie 4: Création des fichiers de mappings hibernate
D'abord créer un package nommé domaine destiné à contenir les fichiers de mappings et les classes java générées par hibernate.
Positionnez-vous à la racine du projet puis ***Fichier-->New-->other-->Hibernate-->Hibernate Mapping File*** *(Voir figure) puis* ***next***



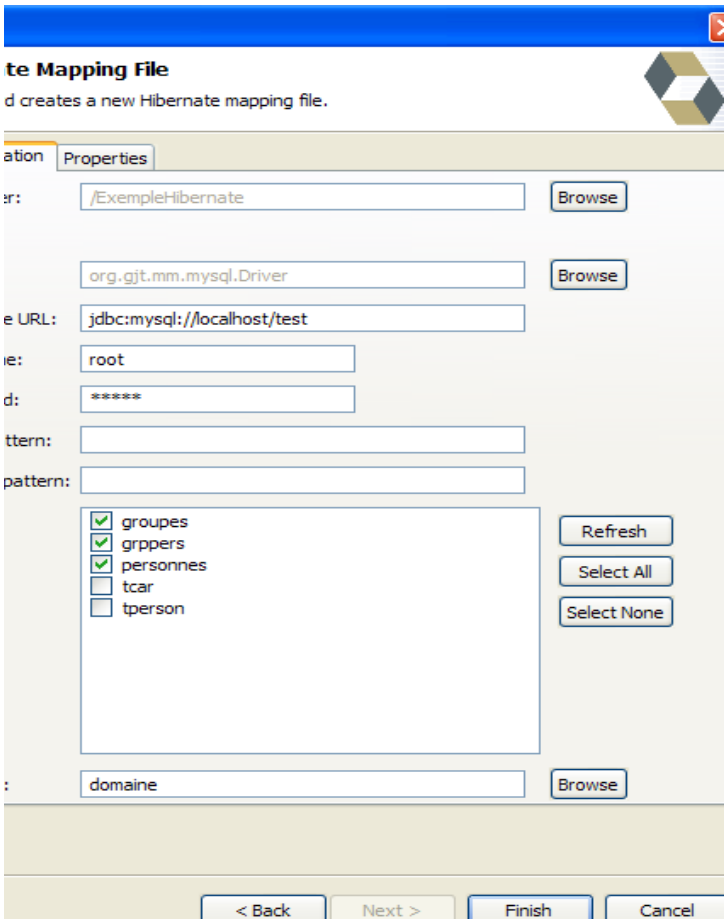
Vous obtenez alors la fenêtre suivante :



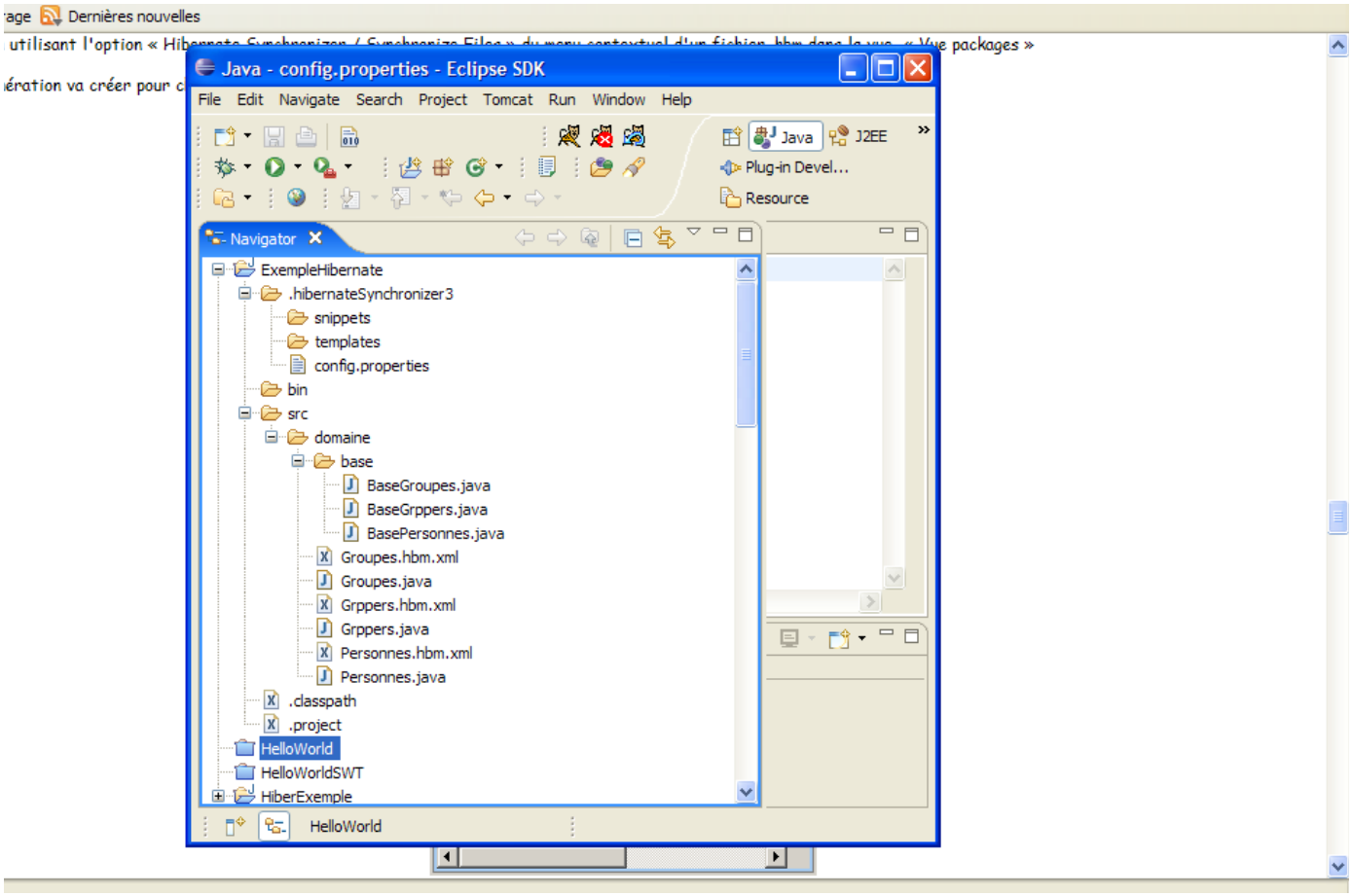
Cliquez d'abord sur browse du côté du driver pour visulaiser le driver à utiliser (driver jdbc). Vous obtenez la fenêtre suivante :



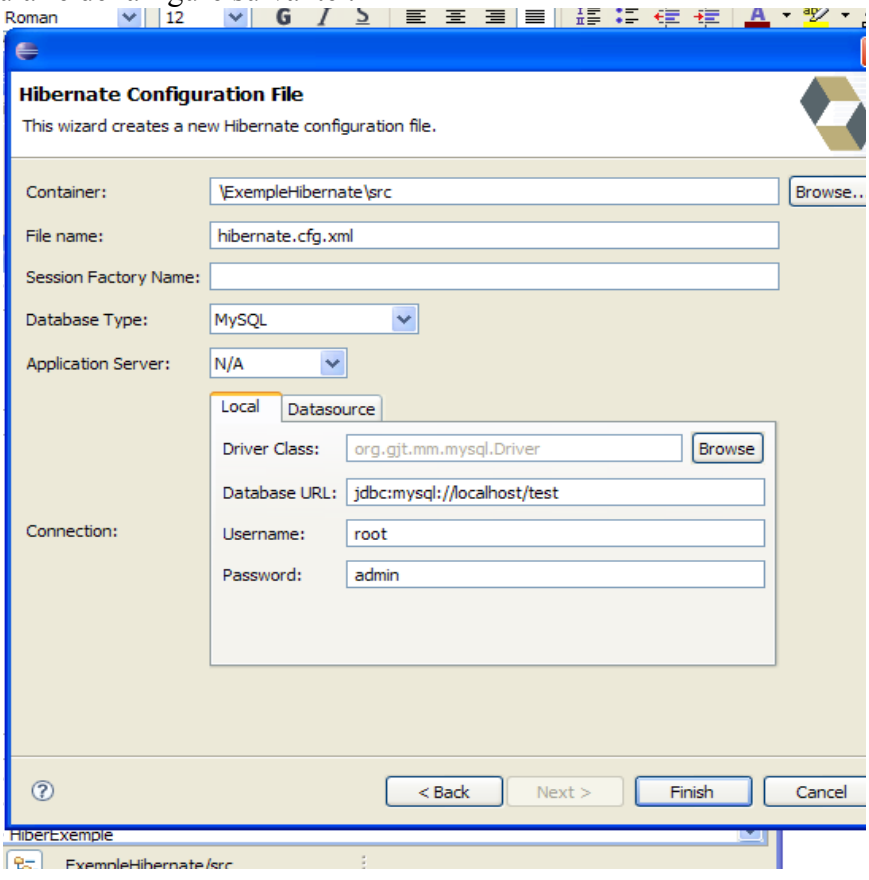
Choisir le driver mysql (par exemple : org.gjt.mm.mysql) puis ok.
Compléter le reste du formulaire de sorte qu'on puisse accéder à la base de données Mysql (voir la figure ci-dessous). Il suffit dans cette fenêtre de positionner l'URL de la base de données, l'utilisateur et son mot de passe, le nom du package java où vous voulez voir apparaître vos fichiers de mappings et les classes java générées puis cliquer sur refresh pour vous connecter à la base de données et sélectionner les tables pour lesquelles vous voulez réaliser le mapping.



En cliquant sur finish vous obtenez des fichiers de typ nomdetable.hbm.xml qui sont les fichiers de mappings entre les tables et les classes. Pour obtenir les classes correspondantes dans le bon package utiliser l'option Hibernate synchonizer-->synchronize files pour chaque fichier de mappings et en utilisant le menu contextuel. Vous obtenez alors dans le package domaine des classes java ainsi qu'un sous répertoire « base » contenant également des classes java utilisées par hibernate. Déplcacer alors les fichiers de mappings dans le package domaine en utilisant l'instruction move d'eclipse. Vous obtenez alors la strcture du projet suivante :



Après avoir générer les fichiers de mappings, vous allez générer un fichier de configuration hibernate (le fichier hibernate.cfg.xml). Pour cela, utiliser le plugin hibernate et choisir la création et renseigner le formulaire de la figure suivante :



Dans le fichier obtenu il faut ajouter la partie soulignée ci-dessous :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory >

            <!-- local connection properties -->
            <property name="hibernate.connection.url">jdbc:mysql://localhost/test</property>
            <property name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
            <property name="hibernate.connection.username">root</property>
            <property name="hibernate.connection.password">admin</property>
            <!-- property name="hibernate.connection.pool_size"></property -->

            <!-- dialect for MySQL -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.show_sql">false</property>
        <property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</propert
y>
    <mapping resource="domaine/Personnes.hbm.xml"/>
    <mapping resource="domaine/Grppers.hbm.xml"/>
    <mapping resource="domaine/Groupes.hbm.xml"/>

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

Partie 5 : Création des classes utilisant hibernate

Dans Ce qui suit, nous allons développer des classes jouant le rôle d'applications utilisant hibernate pour le stockage des données. Dans un premier temps, nous allons écrire une classe qui jouera le rôle de classe de service permettant d'utiliser hibernate de façon simple. Cette classe est la suivante (il faudra la créer dans le répertoire src du projet).

```
import org.apache.log4j.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
public class HibernateUtil {

  private static final SessionFactory sessionFactory;

  static {
    try {
      sessionFactory = new Configuration().configure()
                                .buildSessionFactory();
    } catch (HibernateException ex) {
      throw new RuntimeException("Exception building SessionFactory: "
            + ex.getMessage(), ex);
    }
  }

  public static final ThreadLocal session = new ThreadLocal();

  public static Session currentSession() throws HibernateException {
    Session s = (Session) session.get();
    // Open a new Session, if this Thread has none yet
    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();
  }
}
```

Maintenant on crée une classe qui va consister à afficher les personnes :

```
import java.util.*;
import org.apache.log4j.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import domaine.*;

public class Test1 {

  public static void main(String[] args) {
    try {
      System.out.println("je suis la");
      Session session = HibernateUtil.currentSession();
      System.out.println("je suis la 2");
      List list = session.createQuery("from Personnes ").list();
      Iterator it = list.iterator();

      while(it.hasNext())
      {
        Personnes personne = (Personnes)it.next();
        System.out.println(personne.getNompersonne());
      }

      HibernateUtil.closeSession();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
}
```

La deuxième classe de test consiste à créer de nouvelles instances

```
import java.util.*;
import org.apache.log4j.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import domaine.*;

public class Test2 {
  public static void main(String[] args) {
    try {
      Session session = HibernateUtil.currentSession();
      Transaction tx = session.beginTransaction();

      Personnes personne = new Personnes();
```

```
      personne.setNompersonne("nom6");
      personne.setPrenompersonne("prenom6");
      personne.setCoeffpersonne(new Integer(46));
      personne.setDatenaisspersonne(new Date());
      session.save(personne);

      Groupes groupe = (Groupes) session.load(Groupes.class, new Integer(1));
      Grppers grppres = new Grppers();
      grppres.setIdpersonne(personne);
      grppres.setIdgroupe(groupe);
      session.save(grppres);

      tx.commit();
      HibernateUtil.closeSession();
    } catch (HibernateException e) {
      e.printStackTrace();
    }
  }
}
```