Atelier 5 A

Le Framework Hibernate

Introduction

Hibernate est un Framework Java de persistance qui permet de faire correspondre des tables de base de données relationnelles avec des objets java simples (POJO ou «Plain Old Java Object»). Une fois la correspondance entre les deux mondes définie, le programme Java peut manipuler toutes les données en utilisant que des JavaBean, masquant alors totalement la base de données sous-jacente et ses spécificités. Le Framework assure le remplissage de ces objets et la mise à jour de la base en se basant sur leur contenu.

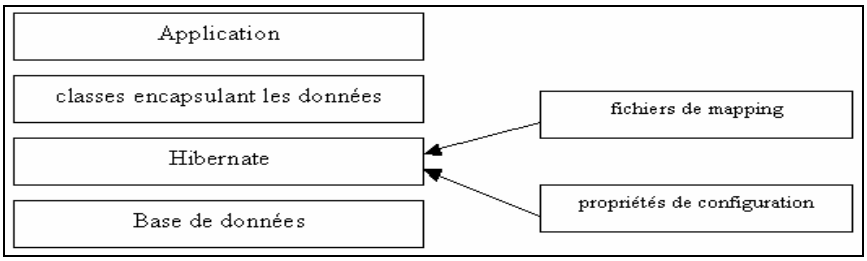
Hibernate 3 franchit un pas supplémentaire. De la même façon que les POJO permettent de manipuler des objets dont les données sont stockées dans une base de données, cette nouvelle fonctionnalité permet de s'affranchir des solutions XML propriétaires. En effet, chaque fournisseur, se devant de se préoccuper de l'importation/exportation des données XML, fournit une solution qui lui est propre à défaut de se baser sur un standard.

Le framework Hibernate permet d'obtenir une représentation XML du résultat d'une requête. Il permet aussi de rendre persistant, c'est-à-dire d'insérer ou de mettre à jour des données dans la base de données depuis des fragments de document XML de façon très simple et similaire à la manipulation de POJO.

Mise en oeuvre du Framework Hibernate

Hibernate a besoin de plusieurs éléments pour fonctionner :

Une classe de type JavaBean qui encapsule les données d'une table donnée nommée « classe de persistance ».
Un fichier de correspondance qui configure la correspondance entre la classe et la table.
Des propriétés de configuration, notamment des informations concernant la connexion à la base de données.
Une fois ces éléments correctement définis, il est possible d'utiliser Hibernate dans le code des traitements à réaliser. L'architecture d'Hibernate est donc la suivante :



Installation de Hibernate

Décompresser l'archive **hibernate-3.2.4.ga.zip** fournis dans les ressources de l'atelier sous la racine **C:** de votre disque par exemple. Il n'est pas nécessaire de positionner les variables d'environement au cas où l'on utiliserait Hibernate dans le cadre d'un IDE tel NetBeans ou Eclipse.
Il est pratique d'ajouter également le Jar du pilote MySQL dans le répertoire lib de l'installation de Hibernate.
Avant de commencer la programmation d'applications DAO, nous devons créer la base de données MySQL qui supportera la persistance. Créer une base de données simple appelée par exemple "**HibernateDB**" et contenant une seule table appelée "**Clients**".

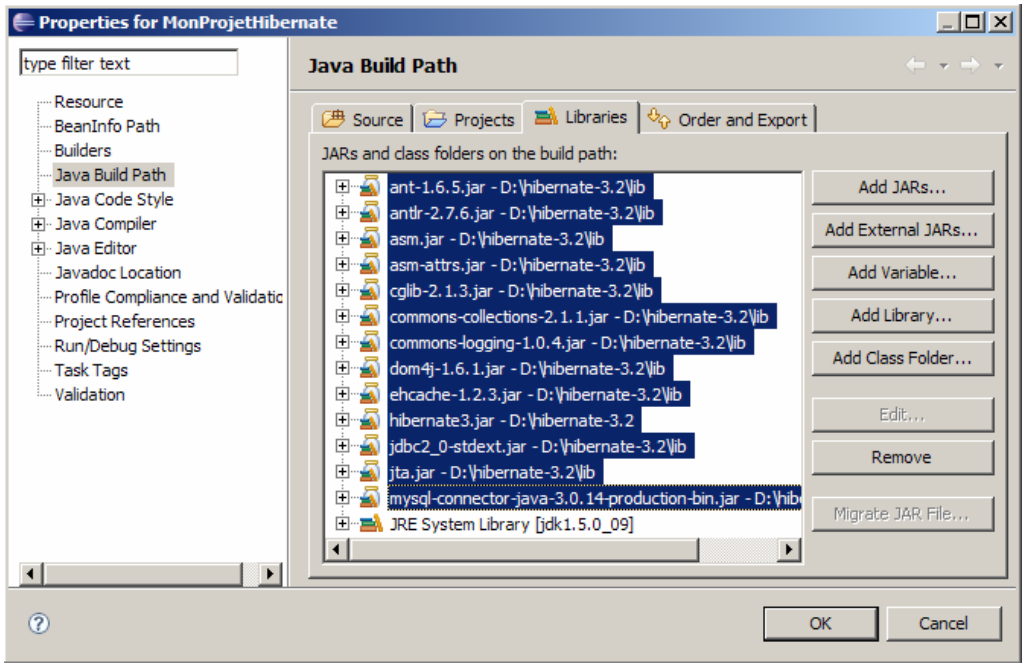
Voici le script de création de la table Clients :

```
CREATE TABLE `personnes` (
             `num_personne` int(11) NOT NULL default '0',
             `nom` varchar(30) default NULL,
             `prenom` varchar(30) default NULL,
             `date_naissance` date NOT NULL default '0000-00-00',
             `nb_enfants` int(11) default NULL,
             PRIMARY KEY  (`num_personne`)
           ) TYPE=MyISAM ;
```

Création du projet Java supportant Hibernate sous Eclipse WTP

Lancer l'IDE Eclipse WTP et créer un projet Java (appelé MonProjetHibernate par exemple). L'ajout du support de Hibernate revient à rajouter un ensemble de Jar du répertoire lib de Hibernate.
Ces Jar sont les suivants.

- hibernate3.jar
- ant-1.6.5.jar
- antlr-2.7.6.jar
- asm-attrs.jar
- asm.jar
- cglib-2.1.3.jar
- commons-logging-1.0.4.jar
- commons-collections-2.1.1.jar
- dom4j-1.6.1.jar
- ehcache-1.2.3.jar
- jdbc2_0-stdext.jar
- jta.jar (utile pour le déploiement avec le conteneur Tomcat)
- mysql-connector-java-3.0.14-production-bin.jar (Pilote JDBC de MySQL)



Création du fichier de configuration : hibernate.cfg.xml

Hibernate propose des classes qui héritent de la classe Dialect pour chaque base de données supportée. C'est le nom de la classe correspondant à la base de données utilisée qui doit être obligatoirement fourni à la propriété hibernate.dialect.

Les propriétés sont alors définies par un tag <property>. Le nom de la propriété est fourni grâce à l'attribut « name » et sa valeur est fournie dans le corps du tag.

Il est possible de fournir les propriétés de configuration « Hibernate » dans un fichier hibernate.properties.

Le fichier **hibernate.cfg.xml** se présente comme suit :

```
<?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:3306/HibernateDB</property>
      <property name="hibernate.connection.driver_class">
                     com.mysql.jdbc.Driver</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password"></property>

            <!-- dialect for MySQL -->
        <property name="dialect">
                org.hibernate.dialect.MySQLDialect</property>

        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.transaction.factory_class">
           org.hibernate.transaction.JDBCTransactionFactory</property>

        <mapping resource="./com/dao/Client.hbm.xml" />

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

Création des fichiers xml de mapping

Ces fichiers sont des éléments majeurs puisqu'ils vont permettre à Hibernate de faire le pont entre les classes de persistance et les sources de données.

Dans cette partie, on va présenter la du fichier de mapping relatif à la table «Clients » nommé Client.hbm.xml :

```
 <?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.dao.Client" table="Clients">
        <id name="clientId" type="int" column="id" unsaved-
value="0">
            <generator class="assigned"/>
        </id>
        <property name="nom"/>
        <property name="prénom"/>
    </class>
</hibernate-mapping>
```

Il est absolument indispensable d'ajouter la référence du fichier « Client.hbm.xml » dans le fichier de configuration « hibernate.cfg.xml » au niveau de la balise <mapping /> comme suit :

```
        <mapping resource="./com/dao/Client.hbm.xml" />
```

Création des Classes de données

Une classe de données est un Javabean qui va encapsuler les propriétés de la table dans ses champs private avec des getters et setters et qui a un constructeur par défaut.

```
package com.dao;

public class Client implements java.io.Serializable {
      private static final long serialVersionUID = 1L;

      private int clientId;
      private String nom;
      private String prénom;

      public Client() { }  //Constructeur par défaut
      public Client(int clientId, String nom, String prénom) {
            this.clientId = clientId;
            this.nom = nom;
            this.prénom = prénom;
      }
      public int getClientId() {return clientId;}
      public void setClientId(int clientId) {
            this.clientId = clientId;}
      public String getNom() {return nom; }
      public void setNom(String nom) {this.nom = nom; }
      public String getPrénom() {return prénom; }
      public void setPrénom(String prenom) {this.prénom = prenom; }
      public String toString() {
        return "Je suis le client : "+clientId+ " nommé : "+ nom+
                  "  "+prénom;
      }
}
```

La classe HibernateUtil

La classe Hibernate nommée SessionFactory permet d'établir la connexion avec la source de données à partir du fichier de configuration « hibernate.cfg.xml ».On remarque que la classe SessionFactory serait instanciée autant de fois qu'il y a de threads, il est donc plus adapté de rendre une même instance de SessionFactory accessible par les threads. Cette classe possède une méthode appelée currentSession() qui retourne la session hibernate en cours si elle existe sinon elle se charge d'ouvrir une nouvelle session .

HibernateUtil.java

```
package com.hibernate;
import org.hibernate.*;
import org.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("Problè¨me 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();
	}
}
```

Quelques exemples d'utilisation des classes Hibernate

Insertion

Pour créer une nouvelle occurrence dans la source de données, il suffit de créer une nouvelle instance de classe encapsulant les données, de valoriser ces propriétés et d'appeler la méthode save() de la session en lui passant en paramètre l'objet encapsulant les données.

La méthode save() n'a aucune action directe sur la base de données. Pour enregistrer les données dans la base, il faut réaliser un commit sur la connexion ou la transaction ou faire appel à la méthode flush() de la classe Session.

Exemple d'insertion d'une nouvelle occurrence dans la table « reservation » :

```
package com.tests;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.dao.Client;
import com.utilities.HibernateUtil;

public class MaClasseDeTestDAO {
  public static void main(String[] args) {
   // On obtient la session hibernate courante
   Session session = HibernateUtil.currentSession();
   Transaction tx = null;
   try {
      tx = session.beginTransaction(); // débuter la transaction
      // on crée une instance de la classe de données Reservations
      Client c1 = new Client(300, "Romdhani", "Mohamed");
      session.save(c1);
      session.flush();
      tx.commit();// " commit" de la transaction
   } catch (Exception e) {
   if (tx != null) {
      tx.rollback();// on effectue un roll back en cas d'exception
                  // afin de garder la cohérence des données
       HibernateUtil.closeSession();
   }
   HibernateUtil.closeSession();// fermeture de la session hibernate
  }
 }
}
```

Lecture & mise à jour

La méthode load() de la classe Session permet d'obtenir une instance de la classe des données encapsulant les données de l'occurrence de la base dont l'identifiant est fourni en paramètre.

```
session = HibernateUtil.currentSession();
Transaction tx = null;
try {  tx = ss.beginTransaction(); // débuter la transaction
        // On charge la classe de données Client dont l'identifiant est égale à 300
        Client cli= (Client)session.load(Client.class,newInteger(300));
        cli.setPrénom("Youssef");//modification de la propriété //Prénom
        session.save(cli);// sauvegarde des modifications en mémoire
        session.flush() ;
        tx.commit(); //" commit" de  la transaction  et mise à jour de la base de données
} catch (Exception e) {
if (tx != null) {
         tx.rollback();// on effectue un roll back en cas d'exception afin de garder la cohérence des
                       //données
         HibernateUtil.closeSession();
}
```

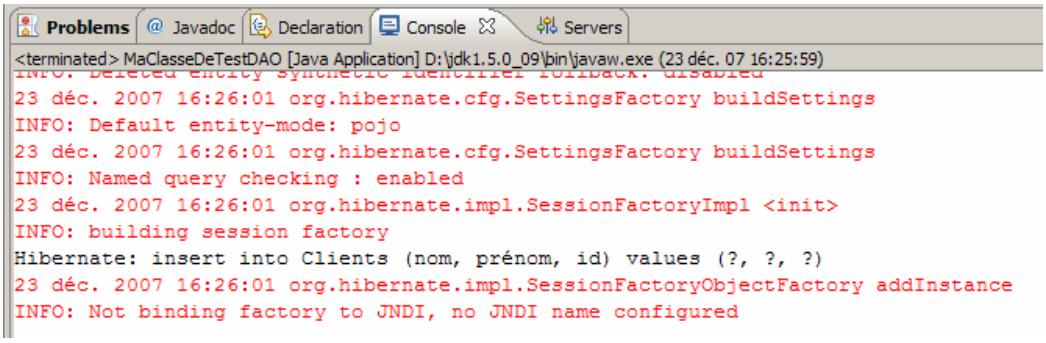
Suppression

La méthode delete() de la classe Session permet de supprimer une instance de la classe des données fournie en paramètre.

```
session = HibernateUtil.currentSession();
Transaction tx = null;
try {      tx = ss.beginTransaction();// débuter la transaction
    // On charge la classe de données Client dont l'identifiant est égale à 300
    Client cli= (Client)session.load(Client.class,newInteger(300));
      session.delete(cli) ; //suppression de l'objet «  cli »
      session.save(cli);// sauvegarde des modifications en mémoire
    session.flush() ;
    tx.commit();//" commit" de  la transaction  et mise  à jour de  la base de données
 } catch (Exception e) {
if (tx != null) {
tx.rollback();// on effectue un roll back en cas d'exception afin de garder la cohérence des données
HibernateUtil.closeSession();
}
```

Exemple de scénario d'exécution

La capture suivante illustre le déroulement avec succès de l'insertion d'une nouvelle entité dans la base de données MySQL. L'option show_sql de la configuration de Hibernate (hibernate.cfg.xml) permet de visualiser sur la console la trace des ordres SQL engagés par Hibernate pour synchroniser assurer le service de persistance.



Une Application plus complète avec Struts et Hibernate (StrutsGrandHotel)

Intégration de Hibernate

Commençons par intégrer Hibernate 3.2 à l'application et traiter le cas d'utilisation « l'administrateur ajoute une chambre à l'hôtel ».

Installation et configuration de Hibernate pour notre application :

1. Création du fichier hibernate.cfg.xml dans le package par défaut et ajout du code suivant :

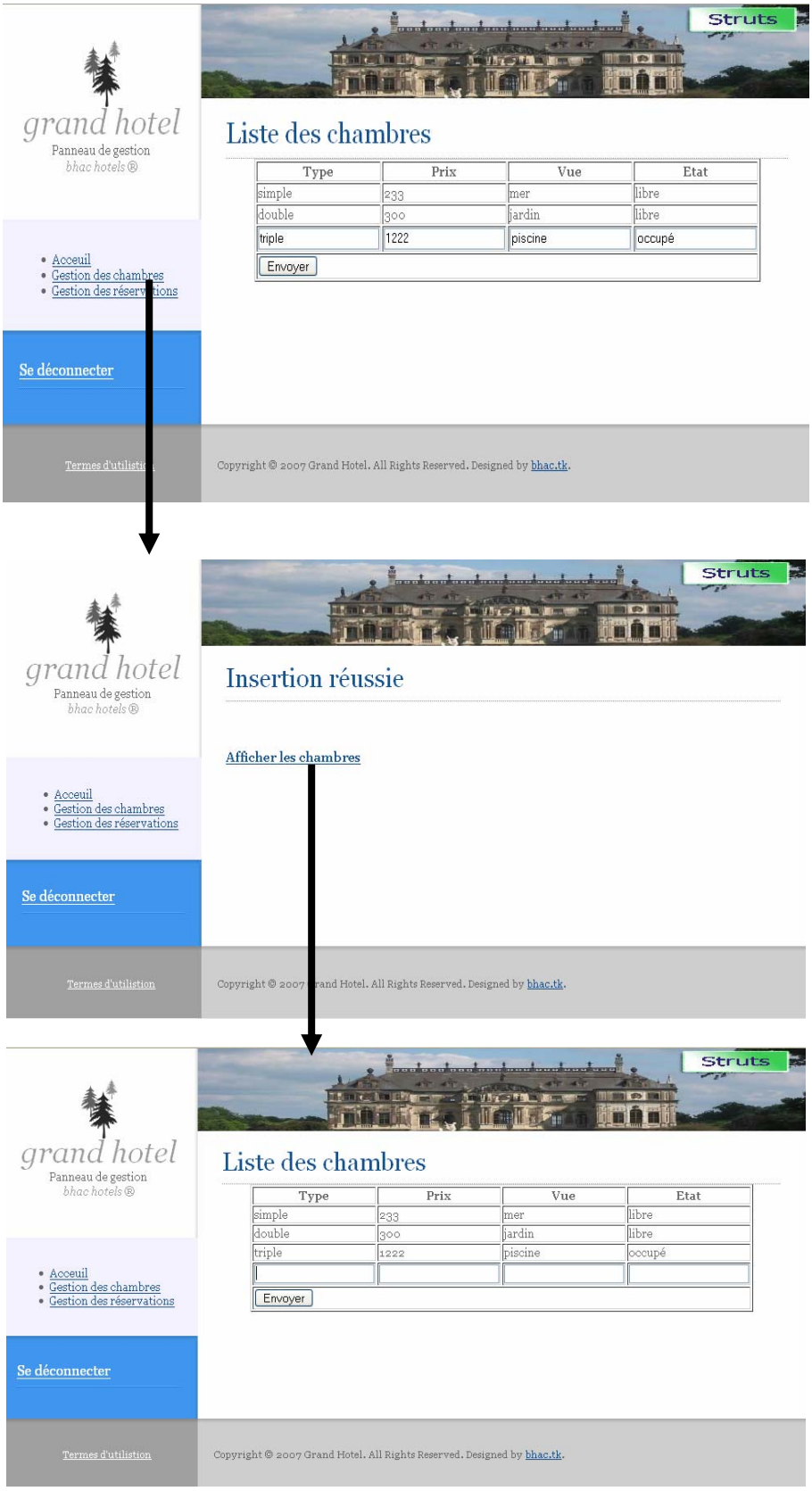
```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/grandhotel</prope
rty>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="show_sql">true</property>
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property
name="hibernate.transaction.factory_class">org.hibernate.transaction.JDBCTran
sactionFactory</property>

        <mapping resource="./com/myapp/beans/Chambre.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Il s'agit du fichier de configuration de Hibernate. Le framework pourra y trouver les informations nécessaires sur la base de données et les différents fichiers de mapping objet/relationnel.

2. Ajout des librairies Hibernate suivantes au répertoire WEB-INF/lib du projet :

- antlr-2.7.6.jar
- asm.jar
- asm-attrs.jar
- cglib-2.1.3.jar
- commons-collections-2.1.1.jar
- commons-logging-1.0.4.jar
- dom4j-1.6.1.jar
- hibernate3.jar
- jta.jar
- log4j-1.2.11.jar

A quoi doit-on arriver :



Construction de la couche de présentation

Le contrôleur ChambreInsertAction se charge de faire appel à la méthode insert de la classe d'accès aux données ChambreDAO en lui fournissant en paramétre la chambre à insérer :

```
public ActionForward execute(ActionMapping mapping, ActionForm  form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        Chambre chambre = (Chambre) form;
        ChambreDAO.insert(chambre);
        return mapping.findForward(SUCCESS);
```

Construction de la couche logique

Le use case que nous avons choisi de traiter concerne l'insertion d'une chambre.
Nous devons alors préparer un bean *Chambre* et un fichier de mapping objet/relationnel pour ce bean qu'on appelle « *Chambre.hbm.xml* » (ce type de nommage doit être respecté) :

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.myapp.beans.Chambre" table="chambre">
        <id name="id_chambre" type="int" column="id_chambre" unsaved-
value="0">
            <generator class="assigned"/>
        </id>
        <property name="type">
            <column name="type" length="55" not-null="true"/>
        </property>
        <property name="prix">
            <column name="prix" length="55" not-null="true"/>
        </property>
        <property name="vue">
            <column name="vue" length="55" not-null="true"/>
        </property>
        <property name="etat">
            <column name="etat" length="55" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>
```

Ce fichier comporte des méta données concernant les colonnes de la table chambre.

Construction de la couche d'accès aux données

Hibernate étant un outil de persistance, il doit s'assurer que la session de travail est unique. Voici l'utilitaire de création et d'obtention de la session en cours :

```
package com.dao;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {
```

```
    private static Log log = LogFactory.getLog(HibernateUtil.class);
    private static final SessionFactory sessionFactory;

    static {
        try {  // Create the SessionFactory
            sessionFactory =
                  new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
             ex.printStackTrace();
            log.error("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static final ThreadLocal session = new ThreadLocal();

    public static Session currentSession() {
        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() {
        Session s = (Session) session.get();
        if (s != null)
            s.close();
        session.set(null);
    }
}
```

Nous ajoutons ensuite la méthode *insert* à la classe ChambreDAO :

```
import org.hibernate.Transaction;
import javax.naming.InitialContext;

public static void  insert(Chambre chambre) {
        Session session;
        Transaction tx;
        //Creation de notre objet Session grâce à notre HibernateUtil
        try{
            session = HibernateUtil.currentSession();
            //Ouverture de notre transaction avec Hibernate grace
            // a la session
            tx = session.beginTransaction();
            // On sauve, on renvoi notre bean à la session Hibernate
            session.save(chambre);
            // Nous commitons la transaction vers la base
            tx.commit();
            //Enfin on ferme la session
            HibernateUtil.closeSession();
        } catch (Exception ex){ex.printStackTrace();}
    }
```