Introduction & Install

Hibernate

Hibernate est un **ORM** (Object Relational Mapping) open source, qui facilite le développement de la couche persistance d'une application. Hibernate permet de représenter une base de données en objets Java et vice versa.

Hibernate est une solution très populaire grâce notamment à ses bonnes performances et au nombre important de bases de données supportées : DB2, Oracle, MySQL, PostgreSQL, Sybase, SQL Server, Sap DB, Interbase, ...

Projet Java Exemple

On va créer une application Java avec intégration de Hibernate (version 5.2.x) pour se connecter à une base de données **catalogue** sous MySQL.

script création de la base de données 'Catalogue' :

```
CREATE DATABASE IF NOT EXISTS `catalogue`
USE `catalogue`;

CREATE TABLE IF NOT EXISTS `categories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(45) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `produits` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(25) NOT NULL,
  `description` varchar(255) DEFAULT NULL,
  `photo` varchar(25) DEFAULT NULL,
  `prix` float DEFAULT '0',
  `id_categorie` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `index_fk` (`id_categorie`),
  CONSTRAINT `fk_categorie` FOREIGN KEY (`id_categorie`)
      REFERENCES `categories` (`id`) ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;

CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `prenom` varchar(255) NOT NULL,
  `nom` varchar(25) NOT NULL,
  `login` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `photo` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=latin1;
```

Création Projet Java

Création Projet Java 'CatalogueHibernate5':

Maven : gérer les dépendances

Ajouter gestion Maven au projet (menu contextuel) :

Dans fichier **pom.xml** généré, ajouter les dépendances pour **MySql** et **Hibernate** :

Maven Dependencies après sauvegarde du fichier **pom.xml** :

Hibernate : Configuration

Utiliser **Hibernate Tools** pour créer le fichier de configuration d'Hibernate :

se positionner sur rép. **src** du projet : file>new>other..

Hibernate Tools : générer les classes 'Entity'

Utiliser **Hibernate Tools** pour générer les classes 'Entity' mappées sur les tables de la base de données 'Catalogue' :

apply + run :

Hibernate Config. : ajouter classes mappées

dans **hibernate.cfg.xml** ajouter le mapping des classes entity :

HibernateUtil : Création SessionFactory et Session

Ajouter la classe **HibernateUtil**qui permettra de créer une `SessionFactory` et un objet `Session` Hibernate.

> L'objet Session est utilisé pour établir une connexion physique avec une base de données. Il est léger et conçu pour être instancié chaque fois qu'une interaction est requise avec la base de données. Les objets ersistants sont enregistrés et récupérés via un objet Session.

```java
package mc.tuto.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class HibernateUtil {
    private static StandardServiceRegistry registry;
    private static SessionFactory sessionFactory;
    public static SessionFactory getSessionFactory() {
        if (sessionFactory == null) {
            try {
                // Create registry
                registry = new StandardServiceRegistryBuilder().
                    configure().build();
                // Create MetadataSources
                MetadataSources sources = new MetadataSources(registry);
                // Create Metadata
                Metadata metadata = sources.getMetadataBuilder().build();
                // Create SessionFactory
                sessionFactory = metadata.getSessionFactoryBuilder().build();
            } catch (Exception e) {
                e.printStackTrace();
                if (registry != null) {
                    StandardServiceRegistryBuilder.destroy(registry);
                }
            }
        }
        return sessionFactory;
    }
    public static void shutdown() {
        if (registry != null) {
            StandardServiceRegistryBuilder.destroy(registry);
        }
    }
}
```

Création SessionFactory : Test Unitaire (JUnit)

Créer un test unitaire avec JUnit :

```java
package mc.tuto.junit;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.junit.Assert;
import org.junit.jupiter.api.Test;
import mc.tuto.hibernate.HibernateUtil;

class SessionHibernateTests {

    @Test
    void test1() {
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        Assert.assertNotNull(sessionFactory);

        System.out.println("SessionFactory : "+sessionFactory);

        Session session = sessionFactory.openSession();
        Assert.assertNotNull(session);

        System.out.println("Session : "+session);

        session.close();
    }

}
```

Résultat test :

Couche Dao

Mise en place du pattern Dao

Création interface générique IDao

```java
package mc.tuto.dao;
import java.util.List;

public interface IDao<T> {
    List<T> getAll();
    T getById(int id);

    T add(T item);
    T update(T item);
    boolean delete(T item);
}
```

Implementer interface : calsses Dao (ProduitsDao, CategoriesDao,..)

classe **ProduitsDao** :

```java
package mc.tuto.dao;

import java.util.List;
```

```java
import javax.persistence.criteria.CriteriaQuery;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import mc.tuto.entities.Produits;
import mc.tuto.hibernate.HibernateUtil;

public class ProduitsDao implements IDao<Produits> {
    Session session;
    public ProduitsDao() {
        SessionFactory sessionfactory = HibernateUtil.getSessionFactory();
        session = sessionfactory.openSession();
    }

    @Override
    public List<Produits> getAll() {
        CriteriaQuery<Produits> criteria =
                session.getCriteriaBuilder().createQuery(Produits.class);
        criteria.from(Produits.class);

        List<Produits> produits = session.createQuery(criteria).getResultList();
        return produits;
    }

    @Override
    public Produits getById(int id) {
        return session.get(Produits.class, id);
    }

    @Override
    public Produits add(Produits item) {
        Transaction transaction = session.beginTransaction();
        session.save(item);
        transaction.commit();
        return item;
    }

    @Override
    public Produits update(Produits item) {
        Transaction transaction = session.beginTransaction();
        session.saveOrUpdate(item);
        transaction.commit();
        return item;
    }

    @Override
    public boolean delete(Produits item) {
        try {
            Transaction transaction = session.beginTransaction();
            session.remove(item);
            transaction.commit();
        } catch (Exception e) {
            return false;
        }
        return true;
```

```
    }
}
```

Tests

```
package mc.tuto.main;
import java.util.List;
import mc.tuto.dao.ProduitsDao;
import mc.tuto.entities.Produits;

public class Main {
    public static void main(String[] args) {

        ProduitsDao pdao = new ProduitsDao();

        List<Produits> produits = pdao.getAll();
        for (Produits p : produits) {
            System.out.printf("[%d] %s\n", p.getId(), p.getNom());
        }
    }
}
```

Copyright 2019, MC.