UNIVERSITÉ DE BOURGOGNE
UFR Sciences et Techniques

Cours: BD et Environnement Distribuées

TP 4 - Mapping Objet Relationnel

Préparé par:

MATTA Elie et al.

Table des matières

Introduction

L'objectif de ce TP est de se familiariser avec 'outil de développement Hibernate à l'aide du tutorial fourni avec la documentation de référence.

Dans ce TP, nous avons réalisé une application qui réalise la gestion des produits (objets java). Certains objets de notre applications doivent être stocké dans une base de données Oracle afin qu'ils soient persistants. Pour ce faire, nous avons tout d'abord créé les objets. Puis, nous avons configuré Hibernate pour qu'il fonctionne avec une base de données Oracle. En fin, nous avons effectué la gestion (gestion de concurrence aux objets) des produits.

Nous avons utilisé aussi ant, un outil qui permet la construction d'applications et l'automatisation des opérations répétitives d'un cycle de développement (nettoyage du projet, compilation, génération d'une documentation, test, déploiement…)

Question 1: Application Hibernate

A) Architecture Hibernate

Hibernate est une architecture de haut niveau qui s'intercale entre une application et une base de données pour proposer des services de persistances. C'est un outil de *mapping* objet relationnel pour les applications *JAVA*. Nous utiliserons donc dans ce TP la version *hibernate3.2* d'*hibernate*.

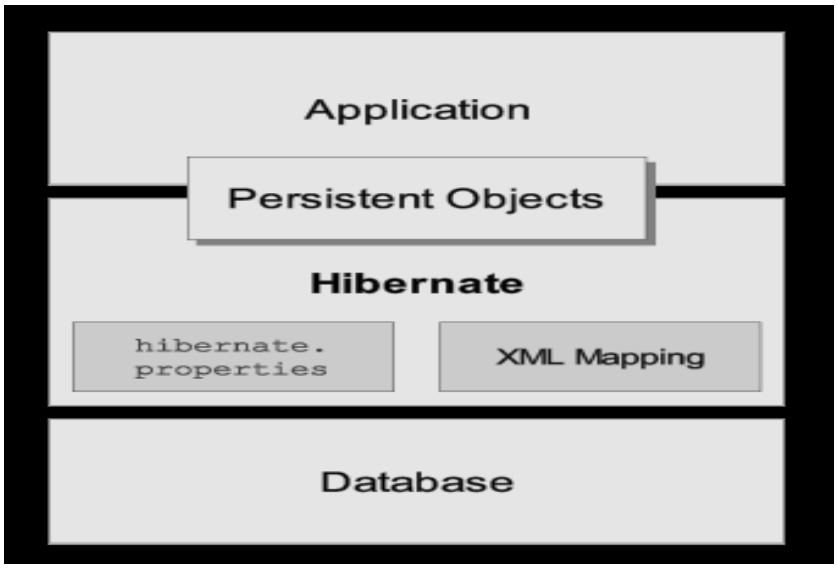

Figure 1 - Architecture simplifié d'Hibernate

B) Structure du répertoire de travail

Tout d'abord, il faut s'assurer que les variables d'environnement nécessaires sont fixés. Dans notre cas, on avait besoin du jdk1.5 et d'un fichier *ant* que nous décrivons dans les paragraphes qui suivent. Nous nous sommes donc placés dans notre répertoire de travail et nous avons saisi les instructions suivantes :

```
export PATH=/usr/local/jdk-1.5/bin:$PATH
export JAVA_HOME=/usr/local/jdk-1.5
export PATH=/opt/csw/bin:$PATH
```

Figure 2 - Instructions

Nous avons ensuite ajouté dans ce même répertoire les sous-répertoires suivant:

- ✓ **Le répertoire** *src/:* Ce répertoire contient les fichiers sources (`.java`) de l'application que nous allons vous faire découvrir dans les sections suivantes. Il contient également le fichier de configuration (*`hibernate.cfg.xml`*) mais aussi les fichiers *`hibernate-mapping`*`(.hbm.xml`) qui sert à décrire les objets persistants.

- ✓ **Le répertoire** *Lib/:* On stock dans ce répertoire tous les librairies (.jar) nécessaires à l'application à savoir *hybernate3.jar*, *OJDBC14.jar*, et tous les bibliothèques se trouvant dans le répertoire lib/ de la distribution hibernte3 téléchargeable dans le site d'hibernate.

- ✓ **Le répertoire** *bin/:* Celui ci contiendra les fichiers générées après compilation.

C) Description des objets

i. Implémentation de l'objet (classe produit)

Dans cette étape, on crée les classes *java* du modèle de données. Ces classes doivent respecter les directives *javaBean*. En effet, elles doivent contenir obligatoirement un constructeur sans argument mais aussi un accesseur *(get)* et un modificateur *(set)* pour chaque attribut.
A ce stade, nous avons implémenté juste la classe *Produit.java*. Cette classe se trouve dans un package *produits* lequel se place dans le répertoire *src*. Ci-dessous l'état de notre répertoire de travail.

Cette classe subira par la suite des modifications. Ainsi, ci-dessous la version initiale de la classe *Produit.java.*

```
package produits;
import java.util.*;
public class Produit{
      private Long id;
      private String referenceICatalog;
      private String description;
      private int qteStock;

      public Produit() {}

      public Long getId(){ return id; }
      private void setId(Long id){ this.id = id;  }

      public String getReferenceICatalog(){ return referenceICatalog; }
      public void setReferenceICatalog(String referenceICatalog){
this.referenceICatalog = referenceICatalog; }

public String getDescription(){ return description; }
public void setDescription(String description){ this.description =
description; }

      public int getQteStock(){ return qteStock; }
      public void setQteStock(int qteStock){ this.qteStock = qteStock; }}
```

Figure 3 - Class produit

ii. Fichier de mapping

Le fichier de *mapping* lie une classe java à une table de la base de données. Il indique à *Hibernate* comment charger et stocker des objets d'une classe persistante. Il décrit aussi les correspondances des attributs d'un objet qui doivent persister avec des colonnes d'une table de la base de données.

La balise *<id>* définit l'identifiant de l'objet. Les balises *<property>* permettent de définir les champs simples. On peut y trouver d'autres balises comme la balise *<set>* qui définisse des associations entre classes. On peut mettre des paramètres dans les balises. Par exemple le paramètre *lazy* permet de ne charger les objets que lorsque l'application en fait explicitement la demande.

On définit ainsi un fichier *hbm.xml* pour chaque classe à persister. Pour l'instant, nous considérons la classe produit à l'état initial. Voici le fichier de mapping *Produit.hbm.xml* correspondant à la classe *Produit.java*.

```
<?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 = "produits.Produit" table = "PRODUIT">
     <id name = "id" column = "PRODUIT_ID">
          <generator class = "native"/>
     </id>
     <property name = "referenceICatalog"/>
     <property name = "description"/>
<property name = " qteStock "/>
</class>
</hibernate-mapping>
```

Figure 4 - Fichier de mapping

D) Configuration

La configuration *d'Hibernate* doit être spécifiée dans un fichier *hibernate.cfg.xml* ou un fichier *hibernate.properties*. Nous avons choisie d'utiliser *hibernate.cfg.xml*. Ce fichier doit être placé dans le répertoire racine. Au démarrage, *Hibernate* se chargera de le chercher automatiquement. Dans ce fichier, on spécifie entre autres :

- ✓ La base de données qui sera utilisée par *hibernate*. En effet, *Hibernate* est une couche de l'application qui se connecte à une base de données. Il a donc besoin des informations de connexion. Les connexions sont établies à travers un pool de connexions *JDBC*, que nous devons aussi configurer. Ainsi on rajoute les lignes suivantes dans notre fichier de configuration (*hibernate.cfg.xml*):

```
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property
name="connection.url">jdbc:oracle:thin:@butor:1521:m12010</property>
<property name="connection.username">login</property>
<property name="connection.password">password</property>
<property name="connection.pool_size">1</property>
```

Figure 5 - Fichier de configuration

- ✓ La variante du *SQL* qui va être générer par *Hibernate* en utilisant l'instruction suivante :

```
<property name="dialect">org.hibernate.dialect.OracleDialect</property>
```

- ✓ Des options comme « hbm2ddl.auto » qui active la génération automatique des schémas de base de données directement dans la base de données : supprime les objets dans la base de données et en créée des nouveaux. On peut utiliser aussi l'option « *show_sql* » qui permet d'afficher ou non sur la sortie standard le *sql* généré.

```
<property name="hbm2ddl.auto">create</property>
```

- ✓ Les fichiers du modèle de donnée (Class Java du modèle) qui vont être utilisé pour le mappage. Dans notre cas il s'agit du fichier *produit.hbm.xml* se trouvants dans le repertoite *Produit*.

L'implémentation du fichier de configuration « *hibernate.cfg.xml*» est consultable en annexe de ce document.

E) Chargement et stockage des objets

La classe *HibernateManager*.java est un singleton qui démarre *Hibernate* et fournit l'accès à un objet SessionFactory. Le service de persistance transparente, est réalisé en ouvrant une session par la création d'un objet Session à partir d'un objet *SessionFactory*. Voici l'implantation de cette classe :

```
package util;

import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateManager {
    private static final SessionFactory session;
    static {
        try {
            session = new
Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." +
ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
    public static SessionFactory getSessionFactory() { return session; }
}
```

Figure 6 - Classe HibernateManager

Question 2 : Manipulation des Objets

A) Initialisation

Pour pouvoir manipuler les objets java tout en respectant l'intégralité référentielle au niveau des tables, nous avons implémenté la classe *ProduitMain*. Cette classe permet de charger et sauvegarder des objets java (« Produit »).

La classe « ProduitMain» contient principalement la méthode *Main ()* qui entre autres récupère un objet de type session à trvers l'objet *sessionFactory*. La classe contient également la méthode *creerEtRestaurerProduit()* qui permet de créer les objets de la classe *Produit*.Voici la description de la classe *ProduitMain.java* :

```
package produits;

import org.hibernate.*;
import org.hibernate.criterion.Expression;
import org.hibernate.Session;
import util.HibernateUtil;
import java.util.*;
import util.HibernateManager;

public class ProduitMain {
    public static void main(String[] args) {
    ProduitMain pm = new ProduitMain();
        if (args[0].equals("store")) {
            pm.creerEtRestaurerProduit("KZEHjk123", "BIDU");
        }
         HibernateManager.getSessionFactory().close();
    }

    private Long creerEtRestaurerProduit(String referenceICatalog, String
description) {
       Session session =
HibernateManager.getSessionFactory().getCurrentSession();
       session.beginTransaction();

       Produit p = new Produit();
       p.setReferenceICatalog(referenceICatalog);
       p.setDescription(description);
       session.save(p);
       session.getTransaction().commit();
        return p.getId(); } }
```

Figure 7 - Classe ProduitMain


{EM}
Contact me for the full version
em@eliematta.com