Développement Web - Object Relational Mapping and Hibernate

Jean-Michel Richer
jean-michel.richer@univ-angers.fr
http://www.info.univ-angers.fr/pub/richer

FACULTÉ
DES SCIENCES
Unité de formation et de recherche

M1/M2 Informatique 2010-2011

Plan

ORM et Hibernate

Objectifs

- se familiariser avec l'ORM
- le mettre en oeuvre sans framework (jobyme)
- le mettre en oeuvre en utilisant Hibernate

Introduction

Introduction

Que'est-ce que l'ORM ?

ORM

L'ORM ou **Object Relational Mapping** a pour but d'établir la correspondance entre

- une table de la base de données
- et une classe du modèle objet

Pourquoi l'ORM ?

Nécessité de l'ORM

- le modèle logique des données est différent du modèle de classe
- réutilisabilité du code pour effectuer les opération de base :
  - DAO
  - CRUD

Rappel CRUD

CRUD

ensemble des fonctions à implanter dans un BD relationnelle :

Opération	SQL
Create	INSERT
Read (Retrieve)	SELECT
Update	UPDATE
Delete (Destroy)	DELETE

Pourquoi l'ORM ?

Comparaison Objet et Relationnel

La représentation sous forme Relationnelle n'entre pas en correspondance avec la représentation Objet

- Objet : notions d'héritage et de polymorphisme
- Objet : pas d'identifiant (pointeur)
- Objet : les relations n:m sont modélisées par des containers

ORM et Java

ORM et Java

Inconvénients liés à Java

- choix important de solutions et outils / API
- évolution des API suivant les versions :
  - difficulté d'apprentissage
  - difficulté de configuration

ORM et Java

ORM, Java et JDBC

la JDBC (Java DataBase Connectivity) a apporté une standardisation au niveau de l'accès des bases de données, mais :

- il faut écrire le code pour réaliser le CRUD
- et réaliser le mapping entre tables et objets

ORM et Java

Solutions existantes

- **API standard** : JDO (Java Data Objects + POJO)
- **API** : Hibernate + POJO
- **API complexes** : EJB Entity

Frameworks de persistance : `http://fr.wikipedia.org` (Hibernate, Cayenne, EJB3, ...

ORM et Java

Comparaison JDO et Hibernate

- JDO est un standard et Hibernate une solution Open Source
- JDO peut traiter la persistance des BD Objets ou XML (possible depuis Hibernate 3)
- JDO propose un langage de requête JDOQL / HSQL
- JDO modifie les POJO alors que ce n'est pas le cas d'Hibernate

ORM et Java

JDO

- www.oracle.com
- http://db.apache.org/jdo/index.html

Hibernate

- http://www.hibernate.org/

JOBYME

JOBYME

JOBYME

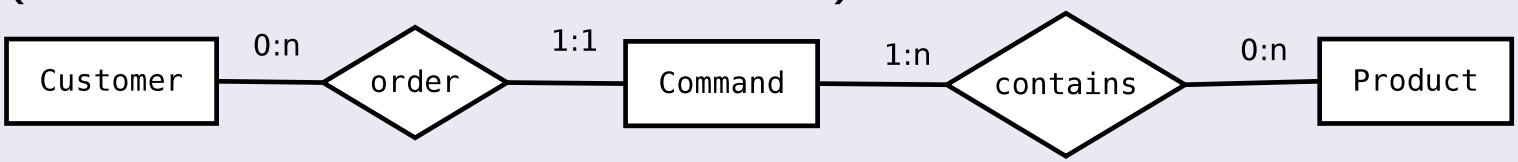
Définition JOBYME

- Java Orm BY ME
- tentative de génération automatique de l'ORM (Code Java + CRUD)
- lecture de la BD (MySQL) et génération du code correspondant

Etude de cas

Etude de cas

On désire modéliser la relation : Client, Commande, Produit (Customer, Command, Product).



Conventions de nommage

Conventions de nommage des entités

- tout en minuscule, séparation par caractère souligné (_)
- **table** : même nom que l'entité (customer)
- **attributs** : préfixés par les 2 premiers caractères de la table et caractère souligné (cu_)
- toujours un identifiant (cu_id)
- minimiser la longueur des champs
- faire en sorte que leurs noms soient explicites

Cas de la table commande

Cas de la table commande

champ	rôle	type	index
co_id	identifiant	integer	PK
co_date	date création	date	NX
co_total_price	prix total	float	
co_cu_id	identifiant client	integer	NX

Conventions de nommage

Conventions de nommage des relations n/m

- par ordre alphabétique du nom des entités / tables
- nom : 4 premières lettres de la première table, 4 premières lettres
- préfixe : premières lettres des entités

Cas de la relation commande / produit

Relation commande / produit : commprod

champ	rôle	type	index
cp_id	identifiant	integer	PK
cp_co_id	identifiant commande	integer	NX
co_pr_id	identifiant produit	integer	NX
cp_qty	quantité	integer	

JOBYME

Installation de la base de données

Mettre en place la base de données `commands.sql` sous MySQL :

- nom de la base : commands
- identifiant de connexion : `commuser`
- mot de passe : `commpass`

Mapping Objet / Relationnel

Mise en correspondance objet / tuple

réalisée au travers d'un fichier de description des **tables** nommé `mapping.xml` placé à la racine du projet. On décrit le nom de la table et le nom de la classe associée :

- `table` : nom de la table
- `class` : nom de la classe
- `prefix` : préfixe du nom de table

Mapping Objet / Relationnel

Mise en correspondance relation / attribut

on décrit ensuite les **relations** :

- `type` : type de relation (one-to-one, one-to-many)
- `attribute` : nom de l'attribut
- `class` : nom de la classe associée
- `crud` : liste des opérations à effectuer (`create`,`retrieve*`,`update`,`delete`)

(*) en cascade

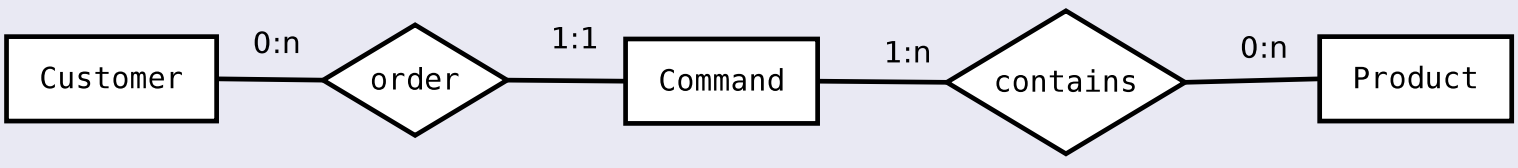
DTD du Mapping

DTD Mapping Objet / Relationnel

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <!ELEMENT mappings (mapping+) >
3  <!ELEMENT mapping (relation*) >
4  <!ATTLIST mapping table CDATA #REQUIRED >
5  <!ATTLIST mapping class CDATA #REQUIRED >
6  <!ATTLIST mapping prefix CDATA #REQUIRED >
7  <!ELEMENT relation EMPTY >
8  <!ATTLIST relation type CDATA #REQUIRED >
9  <!ATTLIST relation attribute CDATA #REQUIRED >
10 <!ATTLIST relation class CDATA #REQUIRED >
11 <!ATTLIST relation crud CDATA #REQUIRED >
12
```

Exemple

Customer, Command, Product



Mapping

Mapping Objet / Relationnel

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mappings SYSTEM "mappings.dtd">
3  <mappings>
4    <mapping table="command" class="Command" prefix="co">
5      <relation type="one-to-one" attribute="customer" class="Customer"
6                  crud="retrieve" />
7      <relation type="one-to-many" attribute="commprods" class="Commprod"
8                  crud="create,retrieve*,update,delete" />
9    </mapping>
10
11   <mapping table="customer" class="Customer" prefix="cu">
12     <relation type="one-to-many" attribute="commands" class="Command"
13                 crud="retrieve" />
14   </mapping>
15
16   <mapping table="commprod" class="Commprod" prefix="cp">
17     <relation type="one-to-one" attribute="product" class="Product"
18                 crud="retrieve" />
19   </mapping>
20
21   <mapping table="product" class="Product" prefix="pr" />
22 </mappings>
23
```

Modèle objet généré

La classe Command

```
1  public class Command extends PersistentObject
2    implements PersistentInterface {
3
4    protected int id;
5    protected String date;
6    protected int cuId;
7    protected float totalPrice;
8    protected int nbCmdlines;
9    // relational fields
10   protected Customer customer;
11   protected List<Commprod> commprods;
12 }
```

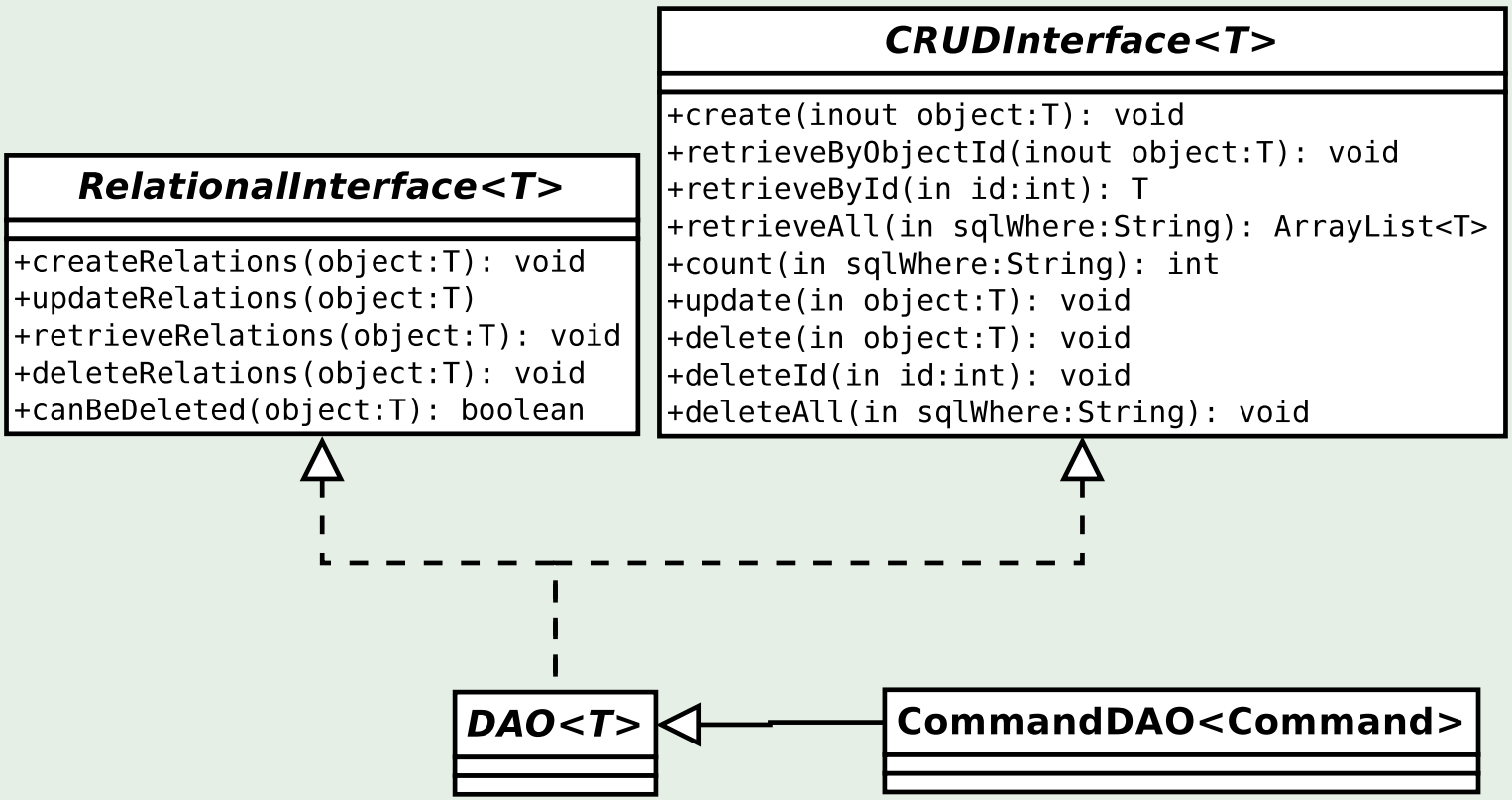
Modèle objet généré

La classe Customer

```
1  public class Customer extends PersistentObject
2    implements PersistentInterface {
3
4    protected int id;
5    protected String firstName;
6    protected String lastName;
7    protected String email;
8    protected String password;
9    protected int rights;
10   // relational fields
11   protected List<Command> commands;
12 }
```

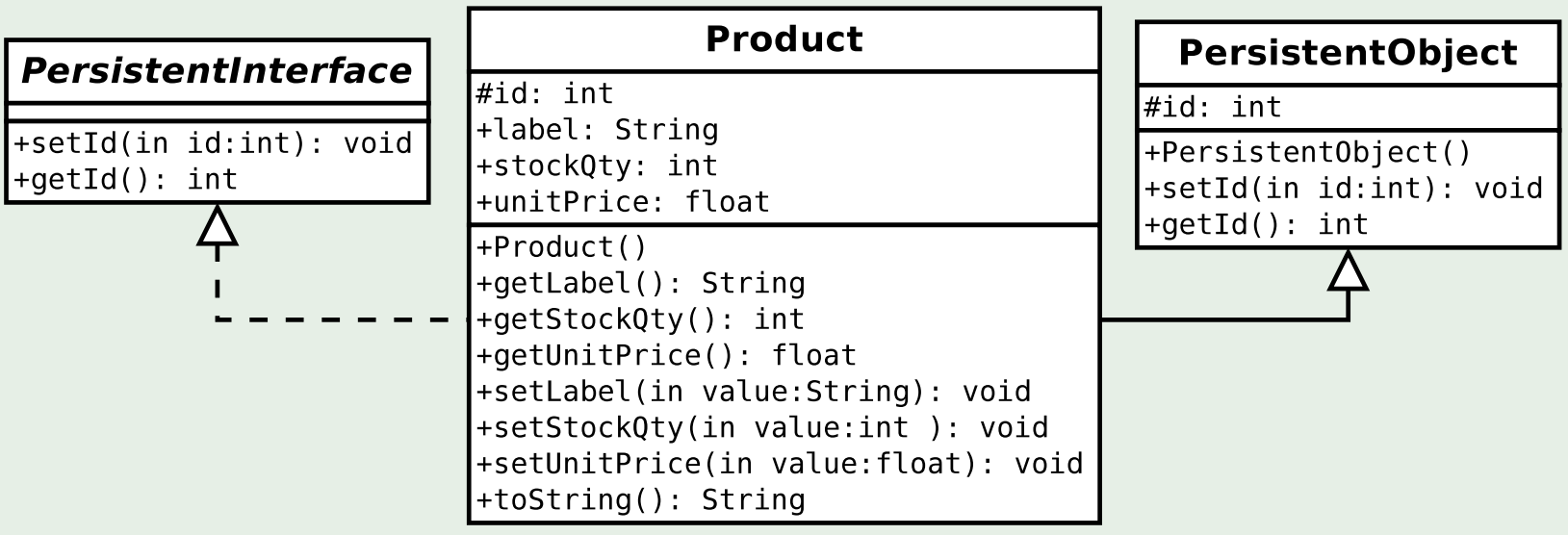
Persistance

Mise en place de la persistance



Persistance

Mise en place de la persistance



Utilisation de JOBYME

Utiliser JOBYME

dans le répertoire du projet lancer : ant

- compile le générateur
- génére les classes à partir du fichier de mapping
- lance un test sommaire

Hibernate

Hibernate

Hibernate

Hibernate

- framework ORM pour Java (BD Relationnelles et Objet)
- requêtes pour CRUD (HQL Hibernate Query Language)
- utilisation des POJOs (Plain Old Java Objects)
- configuration au travers de fichiers XML ou d'annotations
- très fortement configurable
- très difficile à maîtriser !

Historique

Hibernate

- 2001 par Gavin King, alternative à EJB2
- 2003 version 2 qui devient un standard incontournable
- intégration JBOSS (RedHat)
- 2010 version 3 : annotations

Mapping Hibernate

Utilisation des mappings

- par convention l'extension est `.hbm.xml`
- on définit un fichier par classe / table
- placer le fichier de mapping dans le même répertoire que le fichier de classe

Mapping Hibernate

fichier Mapping **NomClasse.hbm.xml**

- permet de faire le lien entre les champs de la table et ceux de la classe
- doit être déclaré dans le répertoire de la classe

example Product

Product.hbm.xml déclaré dans `com.mysite.model`

Mapping Hibernate

Modèle de déclaration

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <!DOCTYPE hibernate-mapping
3    PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5  <hibernate-mapping package="com.commands.model">
6
7  <class name="Product" table="product">
8    <id name="id" type="integer" column="pr_id">
9      <generator class="native"/>
10     </id>
11   <property name="label" column="pr_label"/>
12   <property name="stock" type="int" column="pr_stock"/>
13   <property name="price" type="float" column="pr_price"/>
14 </class>
15
16 </hibernate-mapping>
17
```

Mapping Hibernate

Structure du mapping

```
<hibernate-mapping package="PackageName">
<class
  name="ClassName"
  table="tableName"
    lazy="true|false"
  polymorphism="implicit|explicit"
  where="arbitrary sql where condition"
  rowid="rowid"
    subselect="SQL expression"
  ...
>
  ...
</hibernate-mapping>
```

Mapping Hibernate

Structure du Mapping

- **PackageName** : nom du package ou se trouve la classe
- **ClassName** : nom de la classe Java
- **tableName** : nom de la table de la base de données
- autres paramètres à définir selon la base de données et l'environnement

Mapping Hibernate

Définition des champs de la table

- **id** : permet de définir l'identifiant de la table
- **property** : champ simple
- **composite-id** : clé composée
- **timestamp** : champ de type date/heure

Mapping Hibernate

Structure d'un identifiant

```
<id
        name="propertyName"
        type="typename"
        column="column_name"
        unsaved-value="null|any|none|undefined|id_value"
        access="field|property|ClassName">
        node="element-name|@attribute-name|element/@attribute|."

        <generator class="generatorClass"/>
</id>

```

Mapping Hibernate

Définition d'un identifiant

- **name** : nom du champ de la classe
- **type** : integer, long, float, string, character, timestamp, binary, ...
- **column** : nom du champ dans la table
- **generator** : méthode de génération de l'identifiant

Mapping Hibernate

Méthode de génération d'un identifiant

- **assigned** : l'utilisateur est responsable de la génération
- **native** : la base de données est responsable de la génération

Mapping Hibernate

Structure d'une propriété

```
<property
        name="propertyName"
        column="column_name"
        type="typename"
        update="true|false"
        insert="true|false"
        formula="arbitrary SQL expression"
        access="field|property|ClassName"
        lazy="true|false"
        unique="true|false"
        not-null="true|false"
        index="index_name"
        unique_key="unique_key_id"
        length="L"
        precision="P"
        scale="S"
/>
```

Mapping Hibernate

Définition d'une propriété

- **name**, **type**, **column**
- **update**, **insert** : indique que le champ doit être ajouté lors d'une modification ou insertion (défaut à vrai)
- **formula** : expression SQL qui permet de définir le champ
- **lazy** : accès aux objets associés

Stratégies de chargement

Fetching

- concerne le chargement des objets liés par des relations
- problème complexe pour assurer l'efficacité

Exemple : chargement d'un client

Faut-il charger toutes les commandes lors du chargement du client ?

Stratégies de chargement

Fetching strategies

Hibernate définit 4 stratégies de chargement

- Immediate
- Lazy : ne charge pas tout
- Eager : on spécifie quels objets doivent être chargés
- Batch

Utiliser Hibernate

Utilisation de Hibernate

- définir un fichier de configuration
- démarrer Hibernate requiert la création d'une `SessionFactory`

Fichier de configuration

Fichier de configuration `hibernate.cfg.xml`

- définit les paramètres d'accès à la base de données
- définit où trouver les fichiers de mapping (ressource)
- doit être placé dans le répertoire `src` du projet

Configuration d'Hibernate

Fichier de configuration `hibernate.cfg.xml`

```
1  <hibernate-configuration>
2    <session-factory>
3      <!-- Database connection settings -->
4      <property name="connection.driver_class">com.mysql.jdbc.Driver<
5                    /property>
6      <property name="connection.url">jdbc:mysql://localhost:3306/commands<
7                    /property>
8      <property name="connection.username">commuser</property>
9      <property name="connection.password">commpass</property>
10     <!-- JDBC connection pool (use the built-in) -->
11     <property name="connection.pool_size">1</property>
12     <!-- SQL dialect -->
13     <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
14     <!-- Enable Hibernate's automatic session context management -->
15     <property name="current_session_context_class">thread</property>
16     <!-- Disable the second-level cache    -->
17     <property name="cache.provider_class">org.hibernate.cache.
18                   NoCacheProvider</property>
19     <!-- Echo all executed SQL to stdout -->
20     <property name="show_sql">true</property>
21     <mapping resource="com/mysite/model/command.hbm.xml"/>
22     </session-factory>
23 </hibernate-configuration>
24
25
```

SessionFactory

classe `HibernateUtil`

- obtenir une instance de `SessionFactory`
- réalise l'initialisation de la connexion

HibernateUtil

HibernateUtil.java

```
1  import org.hibernate.SessionFactory;
2  import org.hibernate.cfg.Configuration;
3  import static java.lang.System.err;
4
5  public class HibernateUtil {
6     private static final SessionFactory sessionFactory;
7     static {
8        try {
9           // Create the SessionFactory from hibernate.cfg.xml
10          sessionFactory = new Configuration().configure().buildSessionFactory()
11                                                ;
12       } catch (Exception ex) {
13          // Make sure you log the exception, as it might be swallowed
14          err.println("Initial SessionFactory creation failed." + ex.getMessage()
15                                                                  ));
16          throw new ExceptionInInitializerError(ex.getMessage());
17       }
18    }
19
20    public static SessionFactory getSessionFactory() {
21       return sessionFactory;
22    }
23 }
```

Librairies `.jar`

Librairies Hibernate

Mettre dans le répertoire `WEB-INF/lib` ou `lib` :

- l'ensemble des fichiers de Hibernate `lib/required`
- ajouter également slf4j-simple `http://www.slf4j.org/`

HibernateUtil

Application

Mettre en place la configuration JAVA pour réaliser un test avec Hibernate

Hibernate et annotations

Annotations

Hibernate intègre un mécanisme d'annotation qui permet de remplacer les fichiers de mapping par des commentaires dans le code. L'objectif est de

- ne pas séparer le code Java du fichier de mapping
- de manière à configurer Hibernate automatiquement

Exemple d'annotations

Annotations

```
1  import org.hibernate.annotations.Index;
2  @Entity
3  @Table(name="ARTIST")
4  @NamedQueries({
5    @NamesQuery(name="com.oreilly.hh.artistByName",
6    query="from Artist as artist where upper(artist.name)=upper(:name)")
7  })
8  public class Artist {
9    @Id
10   @Column(name="ARTIST_ID")
11   @GeneratedValue(strategy=GenerationType.AUTO)
12   private Integer id;
13
14   @Column(name="NAME",nullable=false,unique=true)
15   @Index(name="ARTIST_NAME",columnNames={"NAME"})
16   private String name;
17
18   @ManyToMany
19   @JoinTable(name="TRACK_ARTISTS",
20     joinColumns={@JoinColumn(name="TRACK_ID")},
21     inverseJoinColumns={@JoinColumn(name="ARTIST_ID")})
22   private Set<Track> tracks;
23
24 }
```

Bibliographie

Bibliographie

Bibliographie, sitographie

- Hibernate Quickly, Patrick Peak, Nick Heudecker, **Manning**, 2006
- Hibernate in Action, Christian Bauer, Gavin King, **Manning**, 2005
- Harnessing Hibernate, James Elliot, Tim O'Brien, Ryan Fowler, **O'Reilly**, 2008
- `www.hibernate.org`