Thomas Duchatelle
10 décembre 2012

Initialisation aux Frameworks : TP

– Hibernate –

Table des matières

I Installation du poste de travail **2**
1 Les outils . . . 2
2 Configuration du poste . . . 2

II Travaux pratiques **2**
1 Contexte et composants pré-configurés . . . 2
a. Structure du projet . . . 3
b. Maven . . . 3
c. Hibernate . . . 3
d. SLF4J . . . 3
e. Le modèle . . . 3
f. Classe *main* . . . 3
2 Mon premier mapping de classe . . . 3
3 Mapping des classes `Author` et `BookStore` . . . 4
4 Quelques idées de requêtes de recherche . . . 4

III Pour aller plus loin ... **4**

IV Pour aller encore plus loin **4**

V Annexes **6**
1 Configuration du poste . . . 6
a. Configuration de *Maven* : `.m2/settings.xml` . . . 6

I - Installation du poste de travail

1 Les outils

Eclipse : utiliser la version *SpringSource Tools Suite* qui contient déjà tous les plugins nécessaires.

Maven : outils de gestion de dépendance, de compilation

HSQL DB : base de données ne nécessitant pas d'installation

Git : gestionnaire de version

Cygwin : console type Linux (inutile sous la VM)

2 Configuration du poste

1. Installer *SpringSource Tools Suite* (Eclipse) : il contient une installation de *Maven*
2. Configurer *Maven* pour qu'il puisse accéder à internet (cf annexes page 6)
3. Copier les sources. Il est possible d'utiliser la commande git : `git clone <emplacement des sources>`
4. Importer sous Eclipse (Maven − > Existing project)

II - Travaux pratiques

1 Contexte et composants pré-configurés

Dans ce TP, nous nous proposons de réaliser le système informatique de gestion d'une librairie présente dans plusieurs villes.

Le modèle qu'on se propose de réaliser est présenté figure 1 de la présente page.

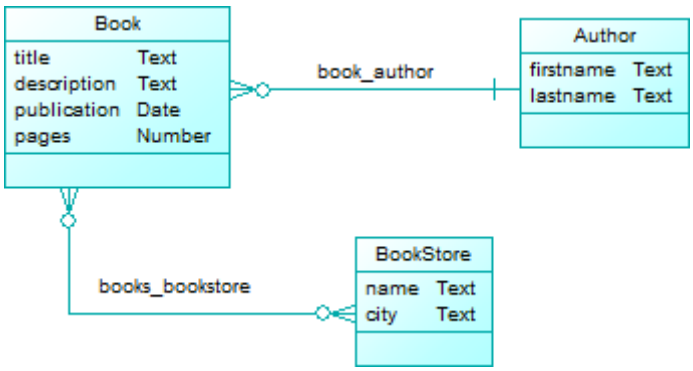

Figure 1 – Modèle simplifié

a. Structure du projet

Le projet se présente sous forme standard Maven :
- `src/main/java` : sources JAVA de l'application
- `src/main/resources` : ressources et fichiers de configuration
- `src/test/java` : sources des tests unitaires
- `src/test/resources` : ressources pour les tests unitaires

b. Maven

Le fichier maven `pom.xml` est déjà créé. Il défini une application Java se présentant sous forme de *jar* et ayant comme dépendances, entre autre :
- Hibernate et le drivers HSQL (version 2.x)
- Logging (SQL4J via LOG4J)
- Outils liés à JUNIT (junit, festassert, mockito)

c. Hibernate

Hibernate est configuré dans le fichier `src/main/resources/hibernate.cfg.xml`. Il y est défini :
- les paramètres de la base de données (url, driver, dialecte SQL)
- les entités (classes persistantes)
- le schéma de la base sera supprimé et recréé à chaque exécution du programme

Ce qui correspond aux paramètres de la `SessionFactory`. Pour accéder à cette dernière, il faut passer par la classe `HibernateUtils`.

d. SLF4J

Le loggeur est configuré pour afficher l'ensemble des requêtes SQL exécutées par Hibernate. Le fichier de configuration est `src/main/resources/log4j.properties`.

e. Le modèle

Les première classes sont déjà écrites pour gagner du temps. Elles sont présentes dans le package `net.yvesrocher.training.frameworks.dto.model`.

Pour rappel, DTO correspond à "Data Transfer Object", ou en français *objet de transfert de données*. Il rassemble différents paramètres de façon logique et objets. Il ne contient **que** des données (et leurs accesseurs), pas de méthode fonctionnelle !

f. Classe *main*

Afin d'aller droit au but, ce TP se fera directement dans la méthode `main` (point d'entrée de l'application) et sera lancé via Eclipse.

2 Mon premier mapping de classe

Dans un premier temps, mappez la classe `Book` afin qu'elle soit persistante.

1. tester la création d'un nouveau livre

2. modifier le nom de certaines colonnes et de la table, regarder le résultat dans la base de données
3. Insérez plusieurs livres dans la base de données et réaliser quelques recherche dessus : tous, puis par titre par exemple
4. Charger un livre par son ID et le modifier.
   Est-ce nécessaire d'appeler la méthode `saveOrUpdate` après l'avoir modifier ? Est-ce possible de créer un autre objet `Book`, de lui donner la même ID, et d'appeler la méthode de sauvegarde ?

3 Mapping des classes `Author` et `BookStore`

Dans cette partie, il est conseillée de d'abord travailler sur le mapping d'`Author` et seulement après passer à `BookStore`.

1. Mapper les classes `Author` et `BookStore`
2. Décommenter les attributs dans la classe `Book` et mapper les associations
3. Que se passe-t-il quand on sauvegarde un `Book` qui est lié à un auteur ? Inversement ?
4. Modifier un livre ou un auteur, en le chargeant via son ID, puis en créant une nouvelle instance avec la même ID.

4 Quelques idées de requêtes de recherche

- Recherche de livres à partir de l'ID de l'auteur
- Recherche de livres à partir d'un nom d'auteur
- Recherche des auteurs dont des ouvrages sont présents dans une ville (on connait les villes des librairies).
- Recherche des auteurs qui ont écrit au moins 2 livres
- Recherche des livres présents dans au moins 2 libraires
- Recherche des auteurs dont au moins 2 livres sont présents dans un librairie données.

III - Pour aller plus loin ...

Pour aller plus loin, mapper le modèle présenté sur la figure 2 page suivante.

On intègre l'idée de BookCopy : un exemplaire d'un livre.

IV - Pour aller encore plus loin

Gestion de l'héritage : introduction de la classe `Person` dont héritent `Customer` et `Author`. Voir figure 3 page suivante.

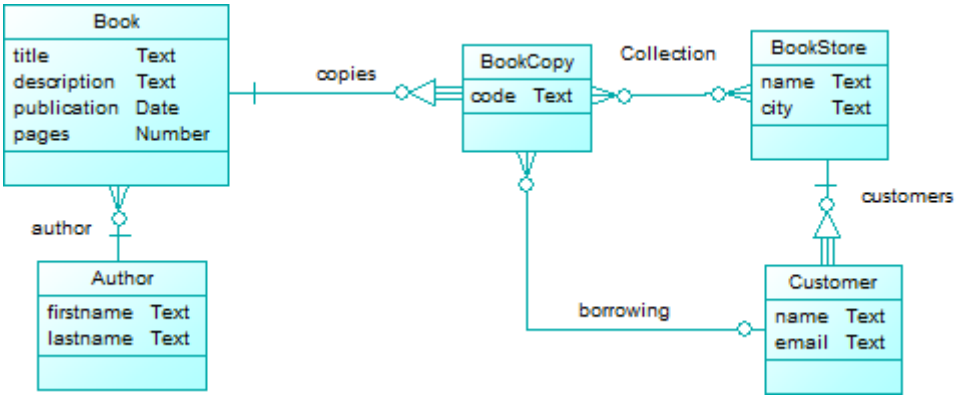

FIGURE 2 – Modèle complet

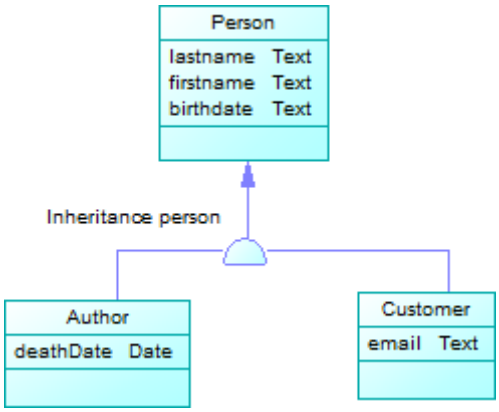

FIGURE 3 – Modèle avec héritage

1. Tester les différentes stratégie d'héritage (une seule table, 2 tables (une par classe), 3 tables (mutualisation des paramètres communs dans une table).
2. Pour chaque stratégie, essayer quelques requêtes : recherche d'auteurs ou de clients, puis recherche de personne en générale.
3. Un héritage au sens base de données est-il une si bonne idée dans ce cas ? Trouver une solution sans héritage en base à proprement parlé.

V - Annexes

1 Configuration du poste

a. Configuration de *Maven* : `.m2/settings.xml`

```
<settings>
  <proxies>
    <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>yrproxy01</host>
      <port>8080</port>
      <username>user</username>
      <password>password</password>
    </proxy>
    <proxy>
      <active>true</active>
      <protocol>https</protocol>
      <host>yrproxy01</host>
      <port>8080</port>
      <username>user</username>
      <password>password</password>
    </proxy>
  </proxies>
</settings>
```