Thomas Duchatelle
31 janvier 2013

# TP – Initialisation aux Frameworks

– Hibernate –

## Table des matières

# 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

**Cygwin** : (facultatif) console type Linux (inutile sous la VM)

**Git** : (facultatif) gestionnaire de version (type svn)

## 2 Configuration du poste

### a. Système minimum

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 8)
3. Copier les sources. Il est possible d'utiliser la commande git : `git clone <emplacement des sources>`
4. Importer sous Eclipse (Fichier → Import, puis, Maven → Existing project)

### b. Facultatif

1. Installer *Cygwin* avec le module Git
2. Modifier le fichier `.bashrc` pour qu'il puisse accéder à internet et avoir les binaires de Maven dans le path.*

```
# Maven
export PATH=$PATH:/cygdrive/d/Utilitaires/springsource/apache-maven
    -3.0.4/bin
export JAVA_HOME="C:\Program Files\Java\jdk1.7.0"
# Donne acces a internet a GIT
export http_proxy=http://yr990200:<password>@yrproxy01:8080
export https_proxy=http://yr990200:<password>@yrproxy01:8080
```

# II - Présentation de l'environnement

## 1 Sujet du TP

Nous nous proposons de réaliser l'application de gestion des livres d'une librairie. Les fonctionnalités et les composants évolueront au fil des TP, et des frameworks utilisés.

Dans cette première étape, **Hibernate** est utilisé pour interagir avec la base de données. Le modèle simplifié est présenté figure 1 page suivante.

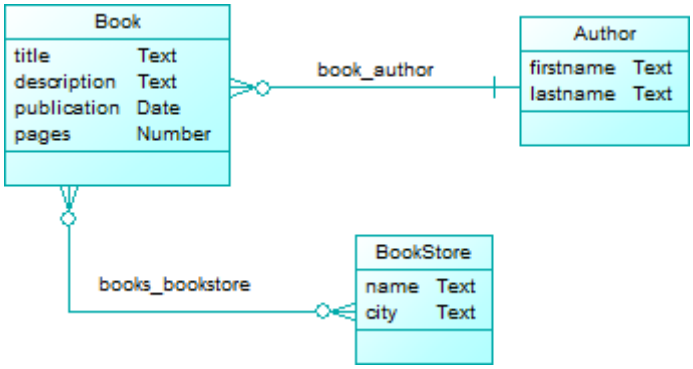


FIGURE 1 – Modèle simplifié

## 2 Structure du projet

### a. Maven

Le fichier *Maven* `pom.xml` est déjà créé.

Il décrit le projet comme 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)

### b. Arborescence

Le projet se présente sous forme de la convention *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

### c. Base de données

La base de données conseillée est une base de données *HSQL*. N'importe quelle autre base de données locale pourrait être utilisée.

Pour démarrer la BDD, double cliquez sur `runServer.bat`. Pour visualiser ce qu'il se trouve dedans, double cliquez sur *runManagerSwing.bat* et indiquez l'URL : `jdbc:hsqldb:hsql://localhost/` comme montré sur la figure c. page suivante.

Hibernate s'occupera de supprimer et créer le schéma à chaque exécution.

## 3 Configuration des frameworks

L'objectif étant d'apprendre l'utilisation des frameworks, ils sont pré-configurés.

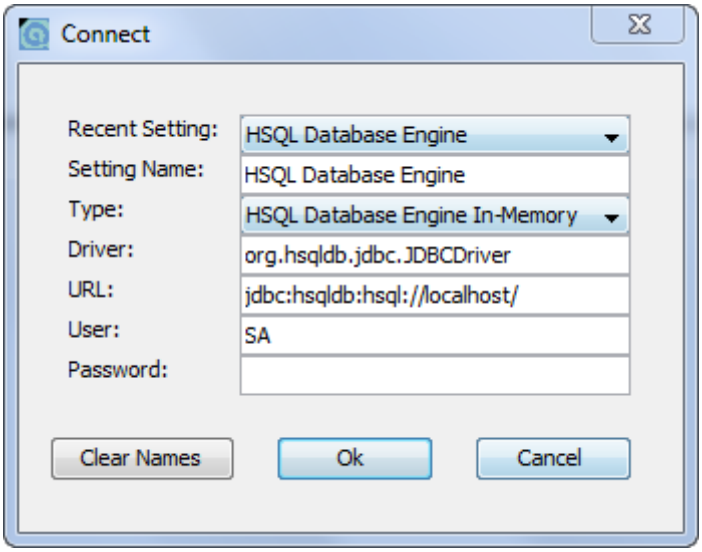


FIGURE 2 – Connexion à la base de données HSQL locale

### a. Hibernate

Hibernate est configuré dans le fichier `src/main/resources/hibernate.cfg.xml`. Y sont définis :
- 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`.

### b. 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`.

## 4 Code

### a. 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 représente les données qui seront persistantes[1]. Il ne contient **que** des données (et leurs accesseurs), pas de méthode fonctionnelle !

1. persistantes : conservées même en cas d'arrêt de l'application

#### b. 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. De plus, la classe contient des méthodes pour générer un petit jeu de test.

## III - Travaux pratiques

Après avoir pris connaissance des différents constituants du projet et de sa structure, passons au développement de l'*interaction avec la base de données*.

### 1 Mon premier mapping de classe

Dans un premier temps, nous ne nous intéresseront qu'à la classe `Book`.

#### a. Sauvegarde d'une entité simple (sans relation) : `Book`

1. mappez la classe `Book` afin de la rendre *persistante*.
2. testez la sauvegarde d'un nouveau livre
3. regardez via l'interface d'HSQL ce qu'il s'est produit :
   - création du schéma et de la structure de la table (nom et type des colonnes)
   - l'insertion d'un enregistrement (le livre)

#### b. Lister les livres présents en base

Sauvegardez plusieurs livres et cherchez à :
- les lister (tous)
- en charger qu'un à partir de son ID : l'afficher, puis 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 ?
- rechercher les livres parus avant 1980, ou entre 1950 et 2000
- en supprimer un

Attention : le schéma est dropé à chaque exécution de la méthode `main`. Il faut donc que les données soient insérés au début de la méthode main. Ce comportement est défini dans la configuration *Hibernate*.

#### c. Adapter la structure des tables

1. modifier le nom de certaines colonnes, regarder le résultat dans la base de données
2. modifier le nom de la table

### 2 Association avec les 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. Déclarer comme persistantes les classes `Author` et `BookStore`
2. Dé-commentez les attributs dans la classe `Book` et configurer 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.

### 3 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.

## IV - Pour aller plus loin ...

Pour aller plus loin, mapper le modèle présenté sur la figure 3 de la présente page.

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

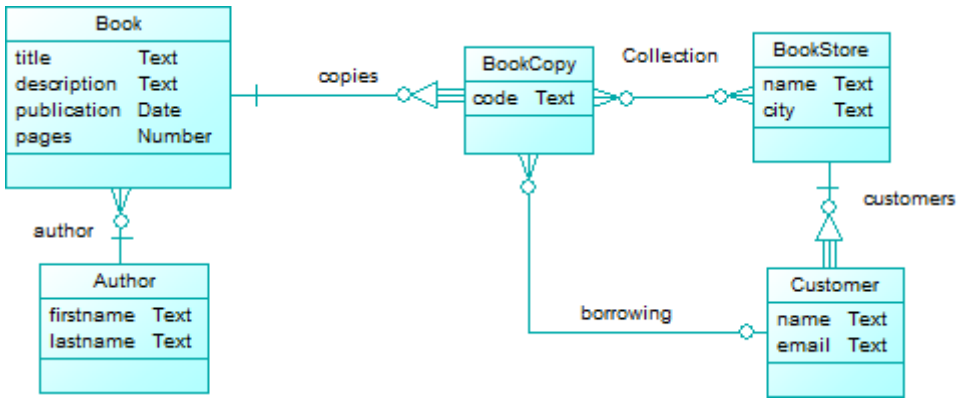


Figure 3 – Modèle complet

Exemple de recherche :
- lister les références d'un livre présent dans une ville.
- liste les références encore disponibles (au moins un exemplaire n'a pas été emprunté)

## V - Pour aller encore plus loin

Gestion de l'héritage : introduction de la classe `Person` dont héritent `Customer` et `Author`. Voir figure 4 de la présente page.

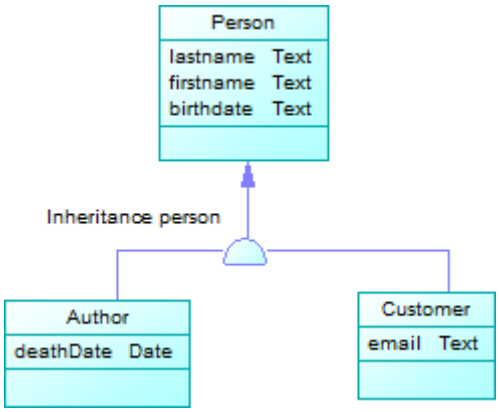


Figure 4 – Modèle avec héritage

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.

# VI - Annexes

## 1 Configuration du poste

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

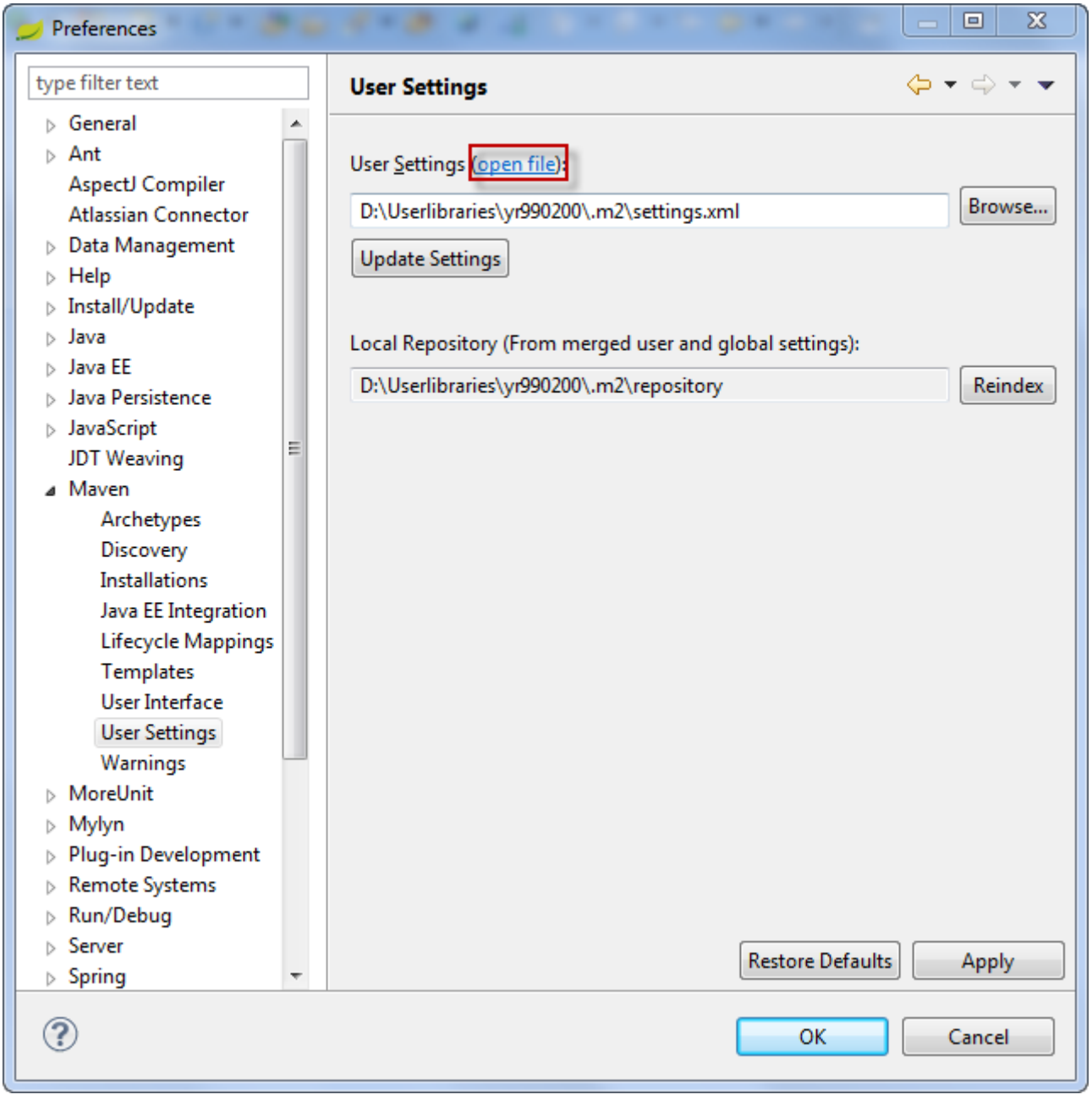


Figure 5 – Éditer le fichier de configuration via éclipse

```
<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>
```