TD 2 Java EE

Mavenisation & Persistance des données

A rendre pour le 29 octobre 2010 à 18h.

L'ensemble du code, des noms de classes, des champs, des méthodes doivent être en anglais. Le code doit répondre aux normes SUN et être obligatoirement testé unitairement..

Le rendu du TD doit être fait par mail à l'adresse suivante: `loyaute@univ-mlv.fr`. *Il s'agit de rendre un fichier zip constitué d'un rapport au format pdf et du code source dans un jar non exécutable.*

1. **Mavenisation du projet**

   Nous allons "Maveniser" le projet que nous avons commencé lors du premier TD.

   Télécharger le Superpom fournit en lien. Le Superpom permet de définir quelles versions doivent être utilisées dans l'esemble d'une entreprise. Les projets font référence à ce parent et déclarent les dépendances dont ils ont besoin dans leurs fichiers pom.xml.

   Nous allons maintenant créer un projet de type Maven pour le TD. Ce projet sera le projet principal et sera constitué des différents modules du projet (persistance, service, web, etc.). Dans ce projet, nous allons modifier le fichier pom.xml en mettant le pom.xml que vous aviez télécharger précédemment. Traditionnellement, le Superpom se trouve sur la repository maven mais l'objectif du TD n'est pas de configurer une repository Maven.

   Dans un second temps vous allez rajouter un module à ce projet principal à partir des sources existantes. Dans le fichier pom.xml vous rajouterez les dépendances dont vous allez avoir besoin (Hibernate, JUnit, etc.).

   Nous allons maintenant lancer la compilation de votre module. Pour se faire cliquez sur votre module, puis faites "Maven..." tapez ensuite la règle "compile". Pour lancer les tests unitaires, tapez la règles "test". Pour l'installation, packaging sous forme de jar, tapez "install".

   Dans les dépendances, vous pouvez configurer le scope des dépendances. Par exemple, test pour dire que la dépendance n'est utile que pour les tests unitaires. Provided pour dire de ne pas ajouter la librairie lors de la constitution du jar ou du war, etc.

2. **Persister nos objets métiers Auteurs & Livres**

   Dans un premier temps, nous allons persister nos objets métiers `Author` et `Book`. Pour se faire, nous allons tout d'abord utiliser les deux annotations `Entity` et `Table` qui permettent de définir respectivement qu'une classe est une entité à persister et le nom de la table qui va la contenir. Nous ne modélisons pas encore les relations existantes entre les différentes classes (auteurs, livres, adresses, téléphones, etc.).

   Dans un second temps, nous allons définir un identifiant technique (clé primaire) à l'aide de l'annotation `@Id`. Nous laissons pour le moment Hibernate géré la séquence de génération de cet clé primaire. La colonne contenant la clé primaire va être définie à l'aide de l'annotation

`@Column`.

Nous allons maintenant définir l'ensemble des colonnes pour les caractéristiques d'un auteur et d'un livre.

Dans le cas ou vous avez utilisé une énumération pour les différentes civilités d'une personne, il faut utiliser l'annotation `@Enumerated` en plus de l'énumération `@Column`.

Concernant la date de naissance, il faut utiliser le package fournie avec joda-time pour la persistance. Il faut utiliser l'annotation `@Type` pour spécifier la "transformation" de DateTime vers une "Date base de donnée".

3. **Relation entre tables Auteurs-Adresses et Auteurs-Telephones**

   Dans votre modélisation UML, il est possible qu'un auteur puisse avoir plusieurs adresses ou plusieurs téléphones. Nous sommes donc dans une relation de type OneToMany. Nous allons donc rajouter l'annotation `@OneToMany` sur les objets de la collection modélisant l'ensemble d'adresses et l'ensemble de téléphones. Plusieurs paramètres sont à rajouter dans cette annotation:

   - mappedBy: quel champ est clé étrangère dans la table jointe;
   - fetch: stratégie de remonter des objets joints (tout ou paresseuse)
   - cascade: quelles opérations doivent être faites en cascande

4. **Relation entre tables Livres-Maison d'Edition**

   Nous allons maintenant nous intéresser au lien existant en un Livre et une Maison d'édition. Un livre est édité par une seule maison d'édition. Nous allons donc avoir une relation `@OneToOne` entre un livre et une maison d'édition. La encore, plusieurs parametres vont devoir être paramétrés:

   - fetch
   - cascade

   Ainsi que le nom de la colonne permettant de faire la jointure en utilisant l'annotation `@JoinColumn`.

5. **Relation entre tables Livres-Auteurs**

   Nous allons ici nous intéresser à la relation entre les livres et les auteurs. Un auteur peut écrire plusieurs livres et un livre peut être écrit par plusieurs auteurs. Nous sommes donc ici en présence d'une relation de type `@ManyToMany`. En utilisant cette annotation il est possible de spécifier le nom de la table permettant de faire cette jointure via l'annotation `@JoinTable`. Sans spécifier cette annotation, la table de jointure sera nommée en fonction du nom de la table source et le nom de la table destination.

6. **Ecriture des DAO**

   Téléchargez maintenant la DAO Générique fournie sur l'URL des TDs. L'ensemble des DAO que vous allez maintenant écrire (interface et implantation doivent étendre de l'interface de la DAO générique et de la classe d'implantation utilisant Hibernate.

   Ecrivez les DAO pour l'ensemble des objets métiers dont vous pensez qu'il est nécessaire d'avoir une DAO pour pouvoir y accèder. Tous ne sont pas forcément pertinent. Justifiez votre choix.

   Un certain nombre de méthode sont déjà disponibles / fournies par la DAO. Nous allons dans l'exercice suivant écrire les méthodes qui pourraient manquer.

7. **Hibernate Query Language**

   Ajoutez dans vos classes DAO de type Hibernate, les requêtes permettant:

   - Trouver un auteur en fonction de son numéro de sécurité social;
   - Trouver un livre en fonction de son numéro isbn;
   - Trouver l'ensemble des livres d'une maison d'édition;
   - Trouver l'ensemble des livres d'une maison d'éditon pour une année donnée;
   - Trouver tout les livres d'un auteur;
   - Trouver tout les auteurs d'un livre;
   - Trouver tout les auteurs d'une maison d'édition;
   - etc.

8. **Criteria Hibernate**

   Nous allons utiliser ici les Criteria Hibernate. La méthode permettant de faire une requête avec les Criteria est disponible dans la DAO Générique. Quel est l'intêret des Criteria Hibernate? Quels sont les différents types de Criteria? Doit on toujours utiliser les Criteria? Justifiez votre réponse.

9. **Héritage de classes**

   Il existe trois grandes stratégies pour représenter l'héritage et faire le mapping Objet-Relationnel.

   - TABLE_PER_CLASS
   - SINGLE_TABLE
   - JOINED

   Explicitez les différences entre ces trois stratégies. Donnez des exemples d'utilisation et identifier des bonnes pratiques d'utilisation de chacune de ces stratégies.

   Comment peut-on partager des propriétés entre deux classes via une class mère sans inclure cette classe mère comme une entité (pas de table sous-jacente)? Donnez un exemple, quel est le comportement sous-jacent?

10. **Test de performance avec Japex: LAZY vs EAGER**

    Dans cet exercice, vous allez tester la différence de chargement de 100000 auteurs dont les livres sont en chargement de type `EAGER` faites la comparaison avec un chargement de type `LAZY`. Explicitez ce comportement. Existe t-il un autre test pertinent pour montrer la différence entre `LAZY` et `EAGER`? Si oui, mettez en place ce "benchmark" et explicitez les tests fait dans votre rapport.

11. **Différence entre les NamedQuery et les Query**

    Quelle différence existe t-il entre une `NamedQuery` et une `Query`?

    Faites des tests de performance avec Japex pour illustrer votre propos.

12. **Cache de second niveau hibernate**

    Configurer un cache de second niveau hibernate. Il existe deux méthodes pour le faire. Donnez les avantages et les inconvénients de chacune des méthodes.

    Quel est l'intérêt d'avoir un cache de second niveau? Illustrez votre propos à l'aide de tests de performance.

    Explicitez les différents paramètres permettant de configurer le cache de second niveau d'Hibernate.