

Leçon 1 – Objets persistants

Java EE offre, dans sa nouvelle version des EJB (la version 3), une toute nouvelle approche pour la persistance des données via l'API JPA (Java Persistence Api). Cette api peut-être utilisée dans des applications distribuées mais également dans des applications mono-poste; c'est d'ailleurs par là que nous allons commencer.

1 PRÉSENTATION

Vos programmes créent des objets. Une fois le programme terminé, ces objets sont détruits. La persistance des données est la propriété d'exister au delà de la durée de vie du programme. Cela demande évidemment de sauver l'objet pour une relecture ultérieure. Quelles solutions peut-on envisager ?

Sérialisation

Ce mécanisme permet de transformer un objet en une série d'octets. Il peut ainsi être sauvé dans un fichier. Plus tard, il pourra être relu et dé-sérialisé. Cela ne peut être envisagé que pour une petite quantité de données ou des données faiblement partagées

Jdbc

Cette API permet d'accéder à des BD via des ordres SQL C'est la solution que vous avez utilisée en deuxième année. Elle est assez lourde car il faut ajouter, pour chaque classe persistante, du code pour assurer la synchronisation. Ce code est long et répétitif.

Mapping objet-relationnel

Le code avec JDBC est assez répétitif; on peut automatiser sa production. Un système de ORM (Objet-relational mapping) automatise les tâches de synchronisation que vous devez coder manuellement avec la technique précédente. Dans sa nouvelle version (Java EE 5), Java introduit les EJB 3 qui permettent une persistance facile des données au travers de l'api JPA¹ (Java Persistence Api). Attention : ce mécanisme est fort différent de ce que proposait la version précédente (EJB 2) qu'on rencontre encore parfois.

Remarquons aussi que le mécanisme de persistance peut aussi être géré par fichiers classiques, bd-objets,... même si la BD relationnelle est la plus fréquemment utilisée.

¹Cette API s'inspire fortement de la technologie Hibernate

2 UNE ENTITÉ

Une **entité** (*entity*; on dit aussi parfois *entity bean* mais ce n'est plus recommandé) est un objet Java qui est automatiquement synchronisé avec une BD (ou tout autre support permanent) via l'api JPA

Exemple

```
@Entity
public class Client {
    @Id
    private long id;
    private String nom;
    // + autres attributs + accesseurs.
}
```

Avec ces annotations, on sait qu'il s'agit d'un objet persistant et on peut déduire la table associée. Par contre, on n'a aucune information sur la BD où se trouve la table.

3 UN CONTEXTE DE PERSISTANCE

Le **contexte de persistance** renseigne sur la BD où faire persister les objets. Il est défini via un fichier XML ce qui facilite les modifications en cas de changement dans l'environnement d'exécution du programme. Il est assez complexe, mais heureusement un IDE comme Netbeans offre un assistant pour la création automatique de ce fichier.

Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
  <persistence-unit name="eVentePU" transaction-type="RESOURCE_LOCAL">
    <provider>oracle.toplink.essentials.ejb.cmp3.EntityManagerFactoryProvider</provider>
    <class>mcd.evente.entity.Client</class>
    <class>mcd.evente.entity.Commande</class>
    <class>mcd.evente.entity.Produit</class>
    <properties>
      <property name="toplink.jdbc.user" value="mcd"/>
      <property name="toplink.jdbc.password" value="mcd"/>
      <property name="toplink.jdbc.url" value="jdbc:derby://localhost:1527/eVente"/>
      <property name="toplink.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver"/>
    </properties>
  </persistence-unit>
</persistence>
```

Vous pouvez y lire notamment : les classes gérées, le produit implémentant la persistance¹; le SGBD, le nom de la BD

¹Toplink est le produit livré avec le Sun Application Server qui s'occupe par défaut de la persistance.

4 LE GESTIONNAIRE D'ENTITÉ

Le gestionnaire d'entité est le service centralisant toutes les actions de persistance. C'est par lui qu'on va passer pour sauver un objet dans la table, y rechercher un objet et d'autres choses encore.

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("eVentePU");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();
Client cl = new Client(...);
em.persist(cl);           // sauver l'objet
cl.setNom(« Dupont »); // le client est synchronisé avec la BD
cl = em.find( Client.class, 2 ); // un client est recherché dans la BD
em.getTransaction().commit();
em.close();
emf.close();
```

5 UN PETIT TUTORIEL

Netbeans fournit un petit tutoriel pour se familiariser à l'utilisation des entités.

<http://www.netbeans.org/kb/55/persistence.html>

Dans ce tutoriel, les entités sont utilisés dans une application web auto-générée grâce à JSF. Ne vous préoccupez pas de tout comprendre pour cette partie, nous y reviendrons.

6 UN SITE DE VENTE

Tout au long du cours, nous prendrons comme exemple un site de vente sur internet. Nous allons commencer par écrire un petit programme mono-poste d'interrogation de la BD liée au site.

Exercice

1. Créez la BD qu'on va utiliser
2. Lancez Netbeans
3. Créez une nouvelle application simple
4. Demandez à Netbeans de créer les entités à partir de la BD (et créez une unité de persistance par la même occasion). Lisez calmement ce qu'il a produit et voyez si vous comprenez tout, surtout la façon dont sont traduites les relations entre les tables.
5. Créez un programme de test qui affiche les informations d'un client à partir de l'identifiant donné en argument du programme. N'oubliez pas de créer un gestionnaire d'entité (cf. exemple donné)
6. Créez un programme de test qui modifie les informations d'un client (données codées en dur dans le test). Relancez le programme précédent pour vérifier que les informations ont bien été sauvées.

7 LA BIBLIOTHÈQUE

Nous utiliserons également la bibliothèque comme exemple pour nous familiariser avec les technologies introduites.

Exercice

Repartez de la BD de la bibliothèque et écrivez un programme de test qui affiche les informations d'un lecteur donné en paramètre (nom, prénom, nb maximum d'emprunts et liste des livres empruntés).

Écrivez aussi un programme qui permet d'effectuer un emprunt (à partir d'un numéro de lecteur et d'un numéro de livre). Pour l'emprunt, il faut effectuer tous les tests de validité comme le fait la version de l'année passée.

8 DÉFINIR LA BASE OU LES ENTITÉS

Si on définit les entités, Java EE peut en déduire et créer la BD. A l'inverse, Netbeans peut générer les entités à partir du schéma de la BD. En pratique, la table est souvent préexistante. Et même dans le cas contraire, les gestionnaires de BD veulent garder le contrôle sur les tables créées.

La pratique la plus répandue est, dès lors, de créer les entités à partir de la BD, quitte à adapter le résultat produit. Par exemple, les noms des attributs ne respectent pas souvent les conventions Java vu que Netbeans ne peut les comprendre (il crée « lecnom » là où d'autres noms seraient plus appropriés : « lecNom » ou encore mieux « lecteurNom », « nomLecteur » ou « nom » tout simplement)

Exercice

Adaptez vos entités pour qu'elle respectent mieux les conventions Java en terme de noms d'attributs et de méthodes. Un conseil : ne modifiez pas tout à la main; Netbeans offre des fonctions de « refactoring » qui vont vous simplifier la tâche.