

P2 – JDBC et architecture d'application – eVente (1)

Dans le TD précédent, nous avons pris en main –via le tutorial de Sun—JDBC, l'API Java permettant d'exploiter les bases de données relationnelles.

Voyons maintenant comment intégrer cette API dans un projet global. Comme pour le projet 1, ceci se fera en plusieurs parties.

1 INTRODUCTION

L' API JDBC de Sun se contente de nous fournir des outils d'accès aux données persistantes. Pour faciliter la conception et la maintenance, nous allons structurer l'application en plusieurs couches.

En ne structurant pas correctement l'application, le programmeur sera confronté à tout moment à deux modes de représentations complètement distincts des données : l'OO de Java et le modèle relationnel de la BD. Ce conflit de représentations (*impedence mismatch*) a tendance à complexifier la tâche du programmeur en le confrontant à tout moment à la correspondance (*mapping*) entre les deux modèles.

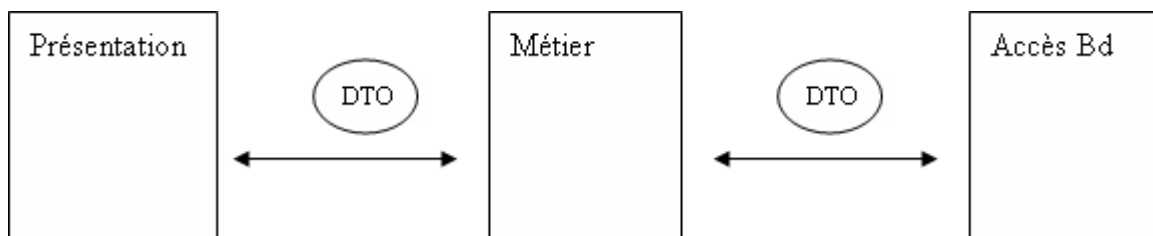
Pour pallier ce problème, nous veillerons à concentrer l'ensemble des accès aux données dans la couche '**Accès aux données**'. Ceci permettra aux autres couches de ne prendre en compte que l'OO.

La couche intermédiaire '**Métier**' ou 'Logique métier' (*Business Logic*) prendra en compte les règles de gestion du monde réel et fera appel à des méthodes de la couche 'Accès aux données'.

La couche supérieure, '**Interface utilisateur**' ou 'Présentation' (*User Interface*), prendra en charge la présentation de l'application et fera appel à des méthodes de la couche 'Métier'.

Cette première partie aura pour but de vous familiariser avec cette architecture. Attention, elle peut être mise en œuvre de multiples manières et l'exemple donné ne privilégie guère la performance.

2 ARCHITECTURE DE L'APPLICATION



Les différentes couches pourront échanger des objets et/ou des données de types primitifs. Les objets échangés seront des **DTO** (*Data Transfer Object* ou Value Object) qui serviront de conteneurs de valeurs extraites de la bd et/ou de valeurs à transmettre à la bd.

2.1 Couche Accès BD

Cette couche offrira des méthodes d'exploitation des données gérées par le gestionnaire de persistance (ici, SGBD relationnel) permettant d'extraire des données de la bd et fournies sous forme de DTO, de mettre à jour la bd en fournissant comme paramètres des DTO ou des types primitifs. Cette couche est la seule qui fera explicitement référence à JDBC. La couche 'Métier' appellera des méthodes de cette couche. Il est à remarquer qu'aucune des méthodes de cette couche ne débutera ou ne terminera une transaction (une transaction est une unité logique et sera initiée et clôturée au niveau de la couche Métier).

Remarque: nous découvrirons l'an prochain un concept Java (*Persistence Unit*) qui simplifie extrêmement la conception de cette couche et qui permettra de disposer d'une notion beaucoup plus riche que celle de DTO..

2.2 Couche Métier

Cette couche implémentera les fonctionnalités de l'application au travers de méthodes d'objets exposées à la couche Présentation. Les objets reprenant de telles méthodes seront qualifiés de '**Façade**'. Généralement, on concevra autant d'objets façades qu'il y a de types d'utilisateurs (visiteur, administrateur, comptable, ...). Chaque méthode des façades faisant appel à des fonctionnalités de la couche 'Data Access' devra initier et clôturer une transaction (ces méthodes seront les seules à mettre en œuvre les transactions).

Exemple:

Dans une bibliothèque, il faut pouvoir saisir un emprunt. Cette fonctionnalité sera offerte par une méthode de la façade dédiée au FrontOffice (cette façade pourra s'appeler FrontOfficeFacade). La méthode pourra présenter la signature suivante:

```
public static void saisieEmprunt (EmpruntJDBC emprunt)
    throw BiblioBusinessException;
```

EmpruntDto pourra avoir comme propriétés, entre autres, `int lecteurNo`, `int livreNo`. La couche présentation permettra à l'utilisateur de saisir le lecteur et le livre et se contentera d'appeler la méthode appropriée de la couche métier. La méthode 'saisieEmprunt' devra initier une transaction (avec un d° d'isolation adéquat) et vérifier que l'emprunt est licite (le lecteur existe-t-il?, le lecteur a-t-il le droit d'emprunter (en ordre de cotisation, nombre maximum d'emprunts atteint, aucun retard, ..., le livre existe-t-il, est-il disponible, ...?). Si une des conditions n'est pas remplie, la transaction sera annulée et une exception sera lancée; si tout se passe bien, la méthode appellera une méthode de la couche 'Accès aux données' qui fera mémoriser l'emprunt à la bd.

2.3 Couche Présentation

Elle a pour but de fournir à l'utilisateur son interface.

2.4 Remarques

Cette présentation est très sommaire mais permet de déjà percevoir un certain nombre d'avantages:

- x complexité de mapping entre OO et modèle relationnel concentrée dans une couche (Accès aux données)

- x indépendance des couches:
 - ✓ si nous décidons d'utiliser un autre type de gestionnaire de persistance de données qu'une BD relationnelle (BD Objet, fichiers classiques, données Xml, ...), seule la couche basse est à modifier.
 - ✓ si nous désirons disposer de plusieurs interfaces utilisateurs distinctes (Web, pda, ...), nous nous contenterons d'écrire une autre couche de présentation en appelant la même couche 'Métier'
 - ✓ les développeurs peuvent se spécialiser dans un type de couche

Attention, la mise en œuvre d'une telle architecture peut paraître lourde dans la mesure où, dans le cadre scolaire, nous n'abordons que des applications relativement restreintes. En pratique, pour des applications de plus grande envergure, il est illusoire de se passer de ce type d'architecture.

Attention, nous sommes loin d'aborder l'ensemble de la problématique du développement d'application, de nombreux aspects ne seront abordés que l'année prochaine (la sécurité d'accès par exemple).

3 LE TRAVAIL À RÉALISER

Pour permettre de ne pas se limiter à une application de trop faible envergure, nous vous fournissons un projet eVente incomplet.

Les packages:

`adt.evente.db`, reprend les méthodes implémentant la couche 'Accès aux données'

`adt.evente.business`, reprend les méthodes implémentant la couche 'Métier'

`adt.evente.dto`, reprend les différents DTO

`adt.evente.seldto`, reprend les dto qui définissent des critères de sélection

`adt.evente.exception`, reprend différentes exceptions nécessaires à l'application

3.1 Premier temps

La classe `adt.evente.Main` présente quelques étapes dont certaines font appel à des fonctionnalités non encore entièrement implémentées. Il vous est demandé de compléter le projet pour que la classe 'Main' fonctionne complètement.

3.2 Deuxième temps (un autre énoncé sera fourni)

Dans la continuation de ce que nous avons fait précédemment, nous allons veiller à réaliser des composants graphiques réutilisables pour faciliter le développement de certaines fonctionnalités de l'interface utilisateur de eVente.

3.3 Remarque

Le choix de cette application pour ALG2 peut paraître curieux dans la mesure où les clients visiteront le site via une interface Web. En pratique, nous aborderons l'interface Web dans le cours d'application distribuée mais les interfaces d'administration du site et de statistiques peuvent être de type 'Swing'.

4 ANNEXES

Vous trouverez en annexe:

- | | | |
|---|---|---------------------------|
| x | Description de la db eVente: | eVenteDataDescription.pdf |
| x | DDL de création de la bd ¹ : | eVenteDdl.sql |
| x | DML de remplissage des tables: | eVenteDml.sql |
| x | Photos des produits: | répertoire img |
| x | Projet netBeans: | répertoire eVente |

¹ Malgré que SQL soit un standard, de légères adaptations seront sans doute nécessaires pour créer la base biblio sous d'autres SGBD qu'Oracle ou JavaDb.