

## P – JDBC et architecture d'application – eVente (1)

*Dans le TD précédent, nous avons pris en main JDBC, l'API Java permettant d'exploiter les bases de données relationnelles. Voyons maintenant, en plusieurs étapes, comment intégrer cette API dans un projet global.*

### 1 INTRODUCTION

---

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

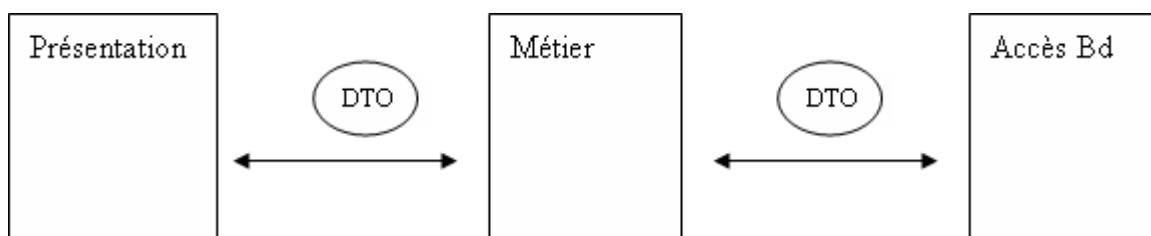
Pour pallier ce problème, nous veillerons à concentrer l'ensemble des accès aux données dans la couche '**Accès aux données**'. Ceci permettra aux autres couches de ne prendre en compte que l'OO. La couche intermédiaire '**Métier**' ou 'Logique métier' (**Business Logic**) prendra en compte les règles de gestion du monde réel et fera appel à des méthodes de la couche 'Accès aux données' pour gérer la persistance.

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' pour offrir les fonctionnalités de l'application.

La première partie de notre approche de eVente 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 objets échangés entre les différentes couches 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 de Persistence

Cette couche offrira les méthodes d'exploitation des données gérées par le gestionnaire de persistance (ici, SGBD relationnel) permettant

- d'extraire des données de la BD et de les fournir à la couche Métier sous forme de DTO,
- de mettre à jour la BD en utilisant comme paramètres des DTO ou des types primitifs.

La couche de persistance est la seule qui fera explicitement référence à JDBC et ses méthodes ne seront appelées que de la seule couche 'Métier'. 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 l'API Java JPA (*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 un objet façade par type d'utilisateur (visiteur, administrateur, comptable, ...) et/ou par cas d'utilisation (Use case). Chaque méthode des façades faisant appel à des fonctionnalités de la couche de Persistence 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 (EmpruntDto emprunt)
    throws 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 [cette notion sera abordée ultérieurement en BD]) 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.

Remarque: la couche 'Métier' reprendra les classes 'Façade' dont les méthodes sont exposées à la couche 'Présentation' mais aussi des classes reprenant des méthodes appelées uniquement par des méthodes définies dans des classes de la couche 'Métier' pour éviter la redondance de code.

Remarque:

Généralement les classes 'Façade' ne seront pas instanciées et n'offriront donc que des méthodes statiques. (cf. FrontOfficeFacade de l'exemple précédent). Dans certains rares cas, nous aurons des 'Façades' qui devront être instanciées car les fonctionnalités qu'elles offrent sont celles d'un 'cas d'utilisation' (Use Case) interactif (exemple: lorsqu'un client parcourra notre site, il remplira son 'panier' au travers de diverses actions et la couche métier aura donc des données à mémoriser avant que le client ne valide ou n'annule ses achats).

## 2.3. Couche Présentation

Elle a pour but de définir l'interface de l'application : les vues et la dynamique de l'application [enchaînement des différentes vues].

### Remarques

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

- complexité de mapping entre OO et modèle relationnel concentrée dans une couche (Accès aux données) permettant de dégager les autres couches de toute problématique de gestion de persistance.
- 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 de Persistance 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 le développement d'un type de couche

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, pour lesquelles la facilité de maintenance est un enjeu essentiel, il est illusoire de se passer de ce type d'architecture.

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 l'ossature<sup>1</sup> d'un projet (eVente).

### Les packages:

```
be.esi.adt.evente.db
```

repréente les méthodes implémentant la couche 'Accès aux données'

```
be.esi.adt.evente.business
```

repréente les méthodes implémentant la couche 'Métier'

---

<sup>1</sup> Ce projet ébauche la gestion d'un site de eCommerce mais simplifiée à outrance un grand nombre d'aspects.

be.esi.adt.evente.dto  
reprend quelques DTO

be.esi.adt.evente.seldto  
reprend les DTO qui reprennent des critères de sélection

be.esi.adt.evente.exception  
reprend différentes exceptions nécessaires à l'application

## **Premier temps**

Le be.esi.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 le 'Main' fonctionne complètement [ceci vous permettra de vous familiariser avec l'architecture présentée].

Pour commencer, vous aurez à réutiliser le JDialog créé dans le dernier TD pour permettre de vous connecter au Sgbd de votre choix.

## **Suite**

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.

Ensuite, nous développerons quelques fonctionnalités de l'application, ce qui nous imposera de réaliser des modifications dans diverses couches.

## **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 [ADI] mais les interfaces d'administration du site et de statistiques peuvent être de type 'Swing'.

## **4 ANNEXES**

---

Description de la db eVente:	DocEvente.pdf
DDL de création de la bd <sup>2</sup> :	eVenteDdl.sql
DML de remplissage des tables:	eVenteDml.sql
Photos des produits:	répertoire eVentesImages
Projet netBeans:	répertoire eVente-enonce <sup>3</sup>

---

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

3 Vous changerez ce nom (*eVente-gxxxxxx* par exemple)