Quelques patterns pour la persistance des objets avec DAO

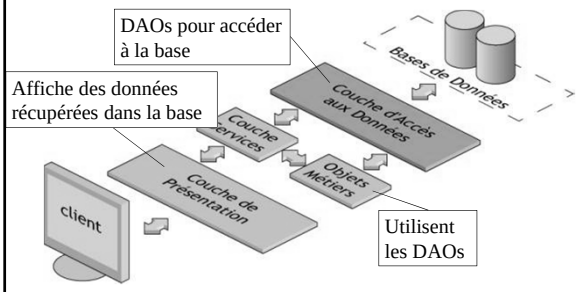
Université Française d'Égypte
Version O 0.9 – 12/10/12
Richard Grin

Principe de base

- Plus fréquent de changer la façon d'effectuer la persistance que de changer le modèle « métier »
- Il faut isoler le plus possible le code qui gère la persistance pour faciliter les changements
- La persistance est isolée dans des objets spécifiques, les DAO (*Data Access Objects*)
- Le modèle de conception le plus utilisé dans le monde de la persistance

Le modèle de conception DTO (*Data Transfer Object*)

Architecture n-tiers



- DTOs peuvent être utilisés pour transporter les données entre DAOs et autres couches

Un fait important

- Les appels de méthode distants sont beaucoup plus coûteux que les appels locaux
- Le coût dépend peu de la quantité de données transférée à chaque appel

Le problème à résoudre

- Des données provenant d'un ou de plusieurs objets distants doivent être récupérées
- Ces données ne peuvent être récupérées que par de nombreux appels de méthodes

Exemple des EJB entités, version 2

- Les DTO ont surtout été utiles avec les anciennes versions des EJB entités (objets qui représentent les données d'une BD)
- Les EJB entités n'étaient pas transportables sur le réseau (ils avaient besoin d'un serveur d'application)
- Depuis EJB 3, les entités JPA sont transportables (entités détachées) et les DTO sont beaucoup moins utiles

La solution DTO

- Le client demande un objet qui contient toutes les valeurs dont il a besoin
- Cet objet, un DTO, est construit sur le site distant et passé en une seule fois au client
- Il contient l'état d'un objet (ou de plusieurs), mais pas son comportement
- Il est donc facilement transportable sur le réseau (par sérialisation)

Le modèle de conception DAO (*Data Access Object*)

Usage des DAOs

- Encapsuler le code lié à la persistance dans des objets DAO dont l'interface est indépendante du type de persistance
- Le reste de l'application utilise les DAOs par cette interface
- Seuls les DAOs devront être modifiés si la façon d'effectuer la persistance est modifiée

DAO

- Chaque classe d'objet métier a son propre type de DAO (**`DAOEmploye`**, **`DAODepartement`**, …)
- Mais le même objet DAO peut être utilisé pour tous les objets d'une classe
- Exemple : une instance de **`DAOEmploye`** peut servir pour tous les employés

CRUD : opérations des DAOs

- **C**reate : créer une nouvelle « entité » dans la base
- **R**etrieve : retrouver des entités de la base
- **U**pdate : modifier une entité de la base
- **D**elete : supprimer une entité de la base
- Plusieurs variantes pour les signatures de ces méthodes

create – paramètres

- Prend en paramètre l'état de la nouvelle entité
- Cet état peut être donné
  - par une série de paramètres :
    `create(int id, String nom,…)`
  - par un DTO
    `create(DTOArticle dto)`
  - par un objet métier :
    `create(Article article)`

create – type retour

- Le type retour peut être
  - **`void`**
  - **`boolean`** pour indiquer si la création a pu avoir lieu (utiliser plutôt une exception)
  - identificateur de l'entité ajoutée (utile si la clé primaire est générée automatiquement)
  - un objet métier correspondant à l'entité ajoutée

Comparaison des variantes (pour create ou autres opérations CRUD)

- Les variantes qui passent les différentes valeurs individuellement sont les plus souples
- Lorsque les DTOs sont utilisées par ailleurs, les variantes avec DTO sont souvent rencontrées
- Les variantes avec objet métier conviennent bien lorsque l'API de persistance représente les entités persistantes par des objets (ORM et SGBDOO)

Exemple de code JDBC pour create

variable d'instance

```
private String insert =
"insert into STYLO (ref, nom, prix, couleur)
  values(?, ?, ?, ?)";
```

méthode **`create`**

```
...
PreparedStatement ps =
  connexion.prepareStatement(insert);
ps.setString(1, reference);
ps.setString(2, nom);
ps.setString(3, couleur);
ps.setBigDecimal(4, prix);
ps.executeUpdate();
```


Retrieve, Update, Delete

- Il existe aussi des variantes semblables aux variantes de Create, que nous ne détaillerons pas ici

Autres méthodes des DAOs

- Les DAO peuvent aussi implémenter des méthodes spécifiques au modèle métier de l'application
- Souvent des méthodes « *retrieve* », par exemple une méthode qui renvoie la liste des candidats qui ont une mention à un examen

2 stratégies d'utilisation des DAOs

1. Le programme qui manipule les objets métier ne connaît pas les DAOs.
   Chaque objet métier a une référence à un DAO et l'utilise pour sa propre persistance.
2. Le programme qui manipule les objets métier utilise directement les DAOs.
   Les objets métier n'ont pas de référence à un DAO.

Stratégie 1

- Les objets métier ont une référence vers le DAO qu'ils utilisent
- Cette référence peut être obtenue par une méthode **`static`** de la classe DAO

Exemple de code pour stratégie 1

```
class Stylo {
  private StyloDAO dao;
  ...
  public void sauvegardeToi() {
    dao = getDAO();
    dao.insertOrUpdate(this);
  }
  private StyloDAO getDAO() {
    if (dao == null)
      dao = StyloDAO.getDAO();
    return dao;
  }
```

Pour simplifier, on ne tient pas compte des exceptions

En dehors de la classe Stylo

```
// Travail avec un stylo (création,
// modification,…)
...
// Sauvegarde du stylo
stylo.sauvegardeToi();
```

Le DAO n'est pas visible en dehors de la classe **`Stylo`**

Exemple de code pour stratégie 2

En dehors de la classe **`Stylo`**

```
Stylo stylo =
  new Stylo(145, "Marker", "noir ",...);
StyloDao styloDao = new StyloDao();
styloDao.create(stylo);
. . .
Stylo stylo = styloDao.findById(1234);
stylo.setPrix(250);
styloDao.update(stylo);
List<Stylo> l = styloDao.findAll();
```

Le code de la classe **`Stylo`** n'utilise aucun DAO