

Le Design Pattern DAO



P.Mathieu

LP DA2I Lille
<http://www.iut-a.univ-lille.fr>
prenom.nom@univ-lille.fr

2 décembre 2018

1 Le Design Pattern DAO

2 Le Design Pattern Factory

Le Design Pattern DAO

Le constat

Une application complexe contient

- Du **savoir-faire** propre à l'entreprise
 - ▶ calculs d'arrêts de comptes pour une banque
 - ▶ calculs de ristournes pour les magasins
- Une gestion de la **persistance** de l'information
 - ▶ jusqu'à présent JDBC,
 - ▶ mais sinon, XML, CSV, ISAM, LDAP, ...
- Une partie **applicative** pour présenter les choses
 - ▶ ... qui peut-être polymorphe.

Le Design Pattern DAO

Le problème

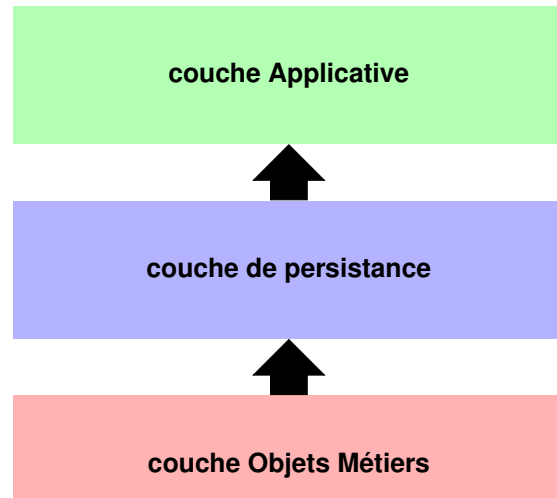


Plus une application grossit, plus elle a besoin d'être structurée

- Pb de lisibilité
 - ▶ Eviter que tout le code soit mélangé, voir dupliqué
- Pb d'évolution
 - ▶ si modification des règles "métier",
 - ▶ si il faut passer à une autre persistance genre "fichiers" ou XML
- Pb de réutilisabilité
 - ▶ si on souhaite utiliser les mêmes objets "métiers" dans différents contextes (web, batch, swing, JavFX, ...)

Le Design Pattern DAO

Le **pattern DAO** propose de séparer le développement en 3 couches : les objets "métier", la persistance, l'application



Le Design Pattern DAO

Principe

Le pattern DAO apporte une réponse pour l'accès aux données

- Permet de regrouper l'ensemble des accès à la base de données à un seul endroit
- Permet de manipuler les enregistrements comme des objets Java
- Implémente en partie ou en totalité les méthodes du **CRUD** (en SGBDR : Insert, Select, Update, Delete)

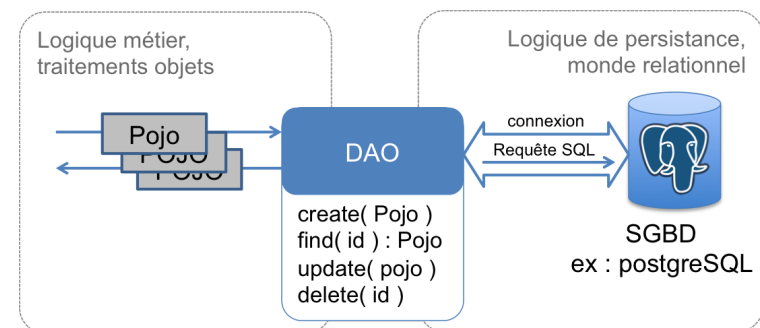
Le Design Pattern DAO

Fonctionnement

- Chaque entité du MCD donne naissance à un objet (**POJO**)
- Chaque propriété devient un attribut de l'objet
- Chaque POJO donne naissance à son DAO
- Le DAO contient une méthode par requête SQL souhaitée
- Les méthodes de lecture renvoient des POJO, des collections ou des itérateurs

Le Design Pattern DAO

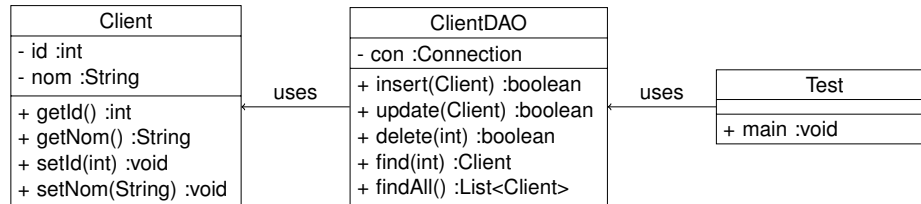
Concrètement



Le Design Pattern DAO

Un exemple

Le système d'information contient une table `client(id, nom)`



Le Design Pattern DAO

Le mapping objet - Relationnel

Les POJO

```
class Client implements Serializable
{
    private int id;
    private String nom;

    public void setId(int id) {this.id=id;}
    public void setNom(String nom) {this.nom=nom;}
    public int getId() {return id;}
    public String getNom() {return nom;}
}
```

Un pour chaque entité du modèle conceptuel

Le Design Pattern DAO

la couche de persistance (DAO)

.... ici avec une Implémentation JDBC

```
class ClientDAO
{
    private Connection con;
    public ClientDAO(Connection con) {this.con=con;}
    ....
    public Client find(int id) {
        String query="SELECT * FROM client WHERE id = ?";
        PreparedStatement ps = con.prepareStatement(query);
        ps.setInt(1,id);
        ResultSet rs = st.executeQuery()
        Client client=null;
        if(rs.next())
        {
            client = new Client(id,rs.getString("nom"));
        }
        .... gestion des exceptions
        return client;
    }
}
```

Le Design Pattern DAO

Un petit test de l'ensemble

```
public class Test
{
    public static void main(String args[]) throws Exception
    {
        Class.forName("org.postgresql.Driver");
        Connection con = DriverManager.getConnection(...);
        ClientDAO clientDAO = new ClientDAO(con);

        Client x = clientDAO.find(3);
        System.out.println("Client " + x.getId() + " " + x.getNom());

        for (Client c : clientDAO.findAll())
            System.out.println(c);

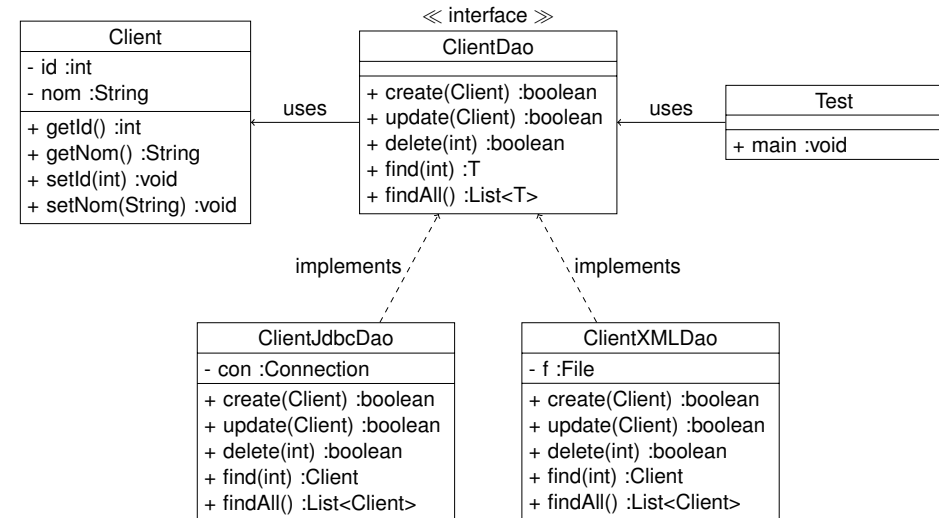
        con.close();
    }
}
```

Le Design Pattern DAO

Deux améliorations impératives

- Respecter le même contrat pour chaque DAO
Interface+classe abstraite
- Factoriser la connexion pour tous les DAO
Singleton sur la source de données

Factoriser via une interface



Le Design Pattern DAO

Factoriser la connexion

```
import java.sql.*;
class DS
{
    public static DS instance = new DS();
    private DS()
    { // gestion des exceptions
        Class.forName("org.postgresql.Driver");
    }
    public Connection getConnection()
    {
        con = DriverManager.getConnection(url,nom,mdp);
    }
}
```

- Chaque constructeur prend le DS en paramètre
- toutes les méthodes du DAO récupèrent la connexion à partir du DS et ferment la connexion

Le Design Pattern DAO

Ce qui donne

```
class ClientDAO
{
    private DS ds;
    public ClientDAO(DS ds) {this.ds=ds;}
    ....
    public Client find(int id) {
        Connection con = ds.getConnection();
        String query="SELECT * FROM client WHERE id = ?";
        PreparedStatement ps = con.prepareStatement(query);
        ....
        con.close();
        return client;
    }

    public boolean create(Client obj){...}
}
```

1 Le Design Pattern DAO

2 Le Design Pattern Factory

Le Design Pattern Factory

principe

- Rassemble les `new` des sous-classes au même endroit.
- Permet de créer toutes les sous-classes de la même manière
- Facilite l'ajout et la modification des objets concrets

Le Design Pattern Factory

Exemple

```
class DAOFactory
{
    //protected static final Connection con =

    public static DAO getClientDAO() {
        return new ClientDAOImpl(con);
    }
    // .....
}
```

Et à l'utilisation

```
DAO<Client> ClientDao = DAOFactory.getClientDAO();
```

Le Design Pattern Factory

En résumé ...

- Le Design Pattern DAO permet d'éviter le code "spaguetti" sépare la conception en 3 couches
- Il facilite la gestion de la persistance des différents objets métiers
- Ceci se fait au prix d'une certaine complexité en code et en exécution
- De nombreux frameworks implémentent JPA, l'interface décrivant le design pattern DAO en Java