Le Design Pattern DAO

Philippe Mathieu & Guillaume Dufrene

IUT-A Lille
http://www.iut-a.univ-lille.fr
prenom.nom@univ-lille.fr

Le problème

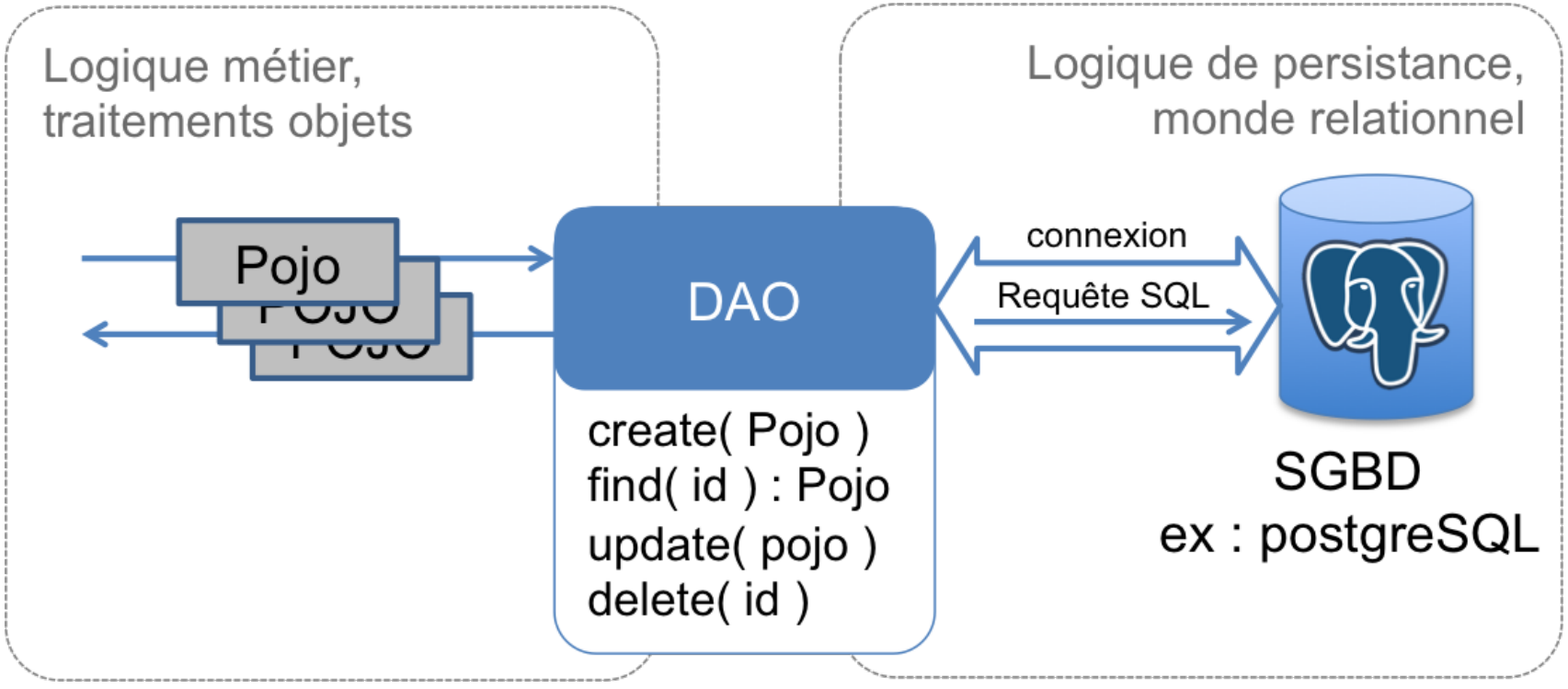
Université de Lille

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, JavaFX, . . .)

Le Design Pattern DAO

Université de Lille



Le Design Pattern DAO

Université de Lille

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 endoit
- 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)

Principe général

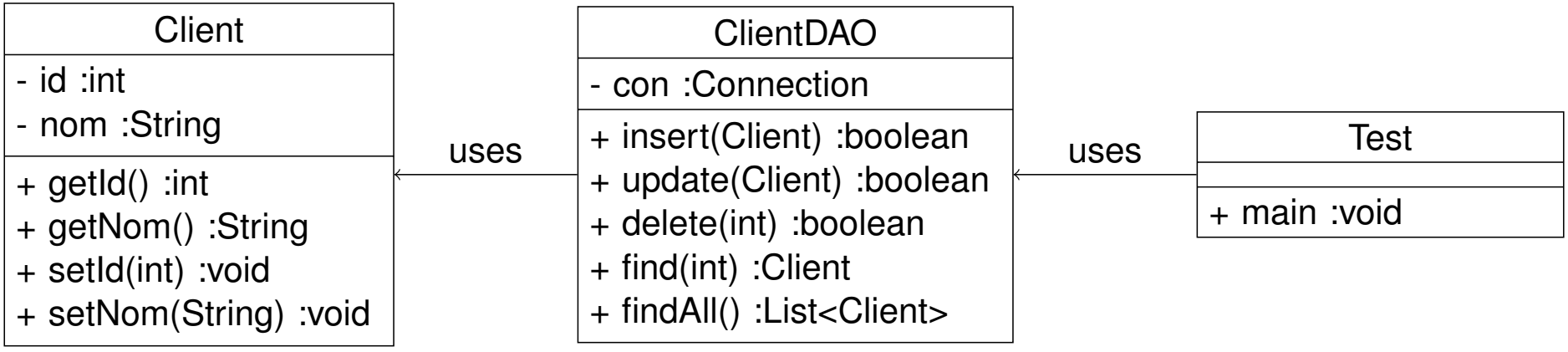
Université de Lille

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

Un exemple

Université de Lille

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



Le mapping objet - Relationnel

Université de Lille

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

la couche de persistance (DAO)

Université de Lille

.... 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;
    }

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

Un petit test de l'ensemble

Université de Lille

```
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();
    }
}
```

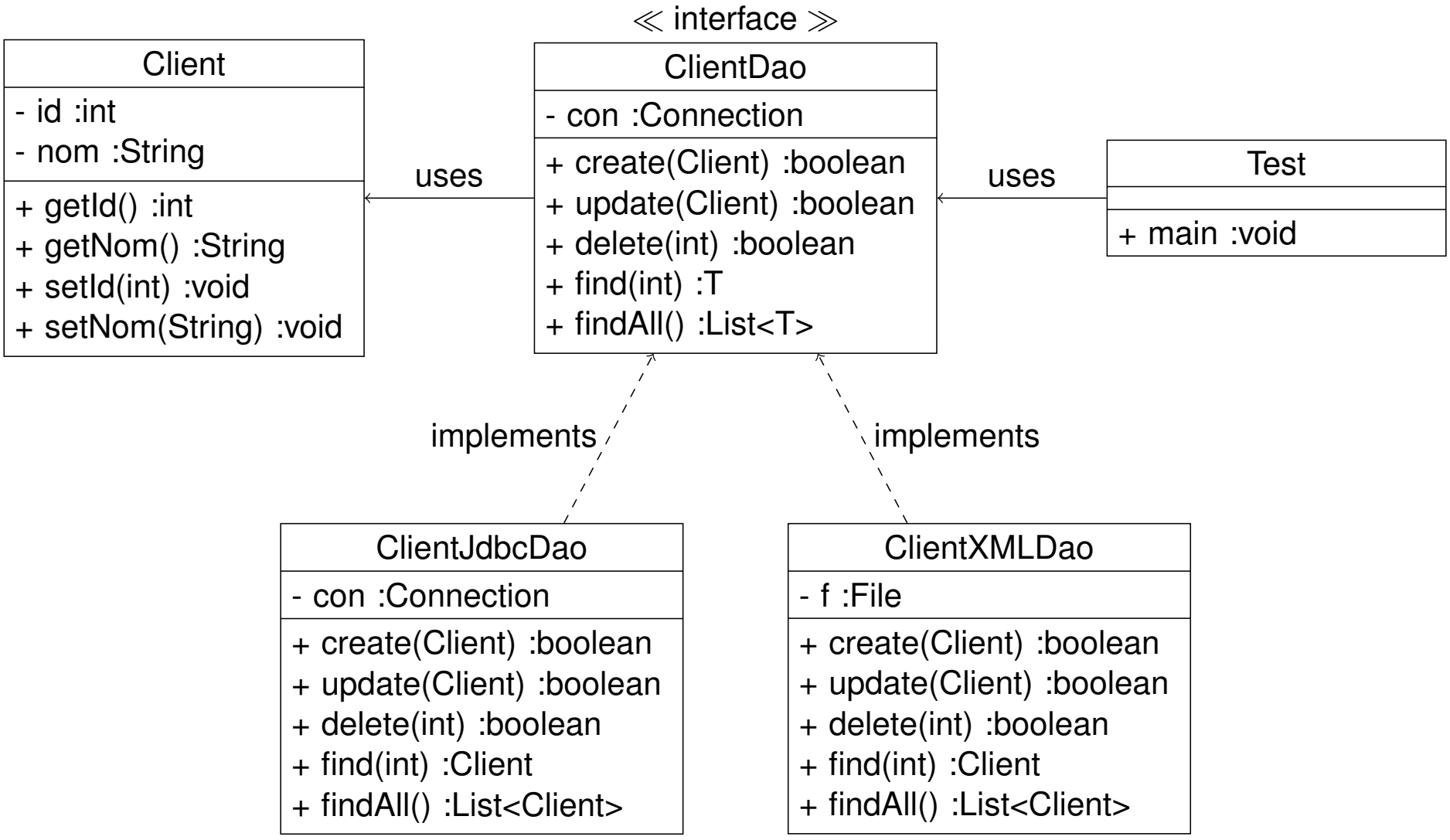
Deux améliorations impératives

Université de Lille

- 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

Université de Lille



Factoriser la connexion

Université de Lille

```
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 connection à partir du DS et ferment la connection

Ce qui donne

Université de Lille

```
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){...}
}
```

Pour aller plus loin

Université de Lille

JPA : Java Persistance API
paquetage `javax.persistence`

Quels que soient les objets DAO et POJO, la structure générale du code reste toujours la même. JPA fournit les bases des frameworks de génération automatique de code pour le pattern DAO.

Nombreuses implémentations :

- Hibernate
- iBatis
- EclipseLink (ex TopLink)
- JOOQ
- iciQL
- JDBi

En résumé

Université de Lille

- Le Design Pattern DAO permet de séparer les préoccupations de persistance de la logique métier
- Il permet d'utiliser les DAO dans des contextes différents (Servlets, programme java)
- Il se met en oeuvre à l'aide de Pojo et d'interfaces pour chaque entité manipulée