


Mapper sa base de données avec le pattern DAO

Par Herby Cyrille 

Date de publication : 8 mai 2009

Utiliser un système de stockage de données, tel qu'une base de données, avec Java (ou tout autre langage orienté objet) peut s'avérer plus compliqué qu'il n'y paraît... Ceci parce qu'on se retrouve vite avec des objets encombrés de requêtes SQL (ou autres instructions) qui rendent leur utilisation assez lourde, leur modification problématique et leur maintenabilité dans le temps quasiment impossible ! Ce tutoriel vous présente un modèle de conception (design pattern) permettant d'éviter ou d'atténuer ce genre de contraintes afin de rendre votre programme plus souple.

Commentez

I - Posons le contexte.....	3
II - Le pattern DAO : Qu'est-ce que c'est ?.....	10
III - Les classes utilisées.....	12
IV - Allez plus loin avec le pattern Factory.....	23
V - Conclusion.....	30
VI - Remerciement.....	30

I - Posons le contexte

Il vous est certainement arrivé d'avoir à travailler avec un système permettant de stocker des données, comme une base de données par exemple. Bien sûr, dans le cas qui nous intéresse, il s'agira d'une base de données, mais le pattern DAO peut aussi fonctionner avec d'autres systèmes de stockage (fichier XML par exemple).

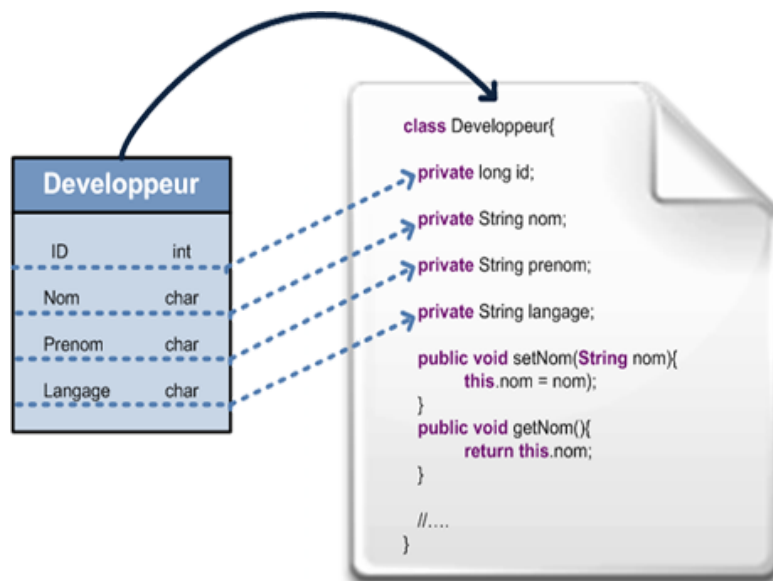
Bien entendu, les problèmes se profilent lorsque vous tentez de mapper ces données avec des objets Java afin de pouvoir :

- créer;
- modifier;
- afficher;
- supprimer;

des données présentent dans ledit système de stockage.

i *Le mappage des données est, en fait, le mécanisme visant à faire correspondre les attributs d'une fiche du système de stockage (BDD) avec les attributs d'un objet (objet Java en ce qui nous concerne).*

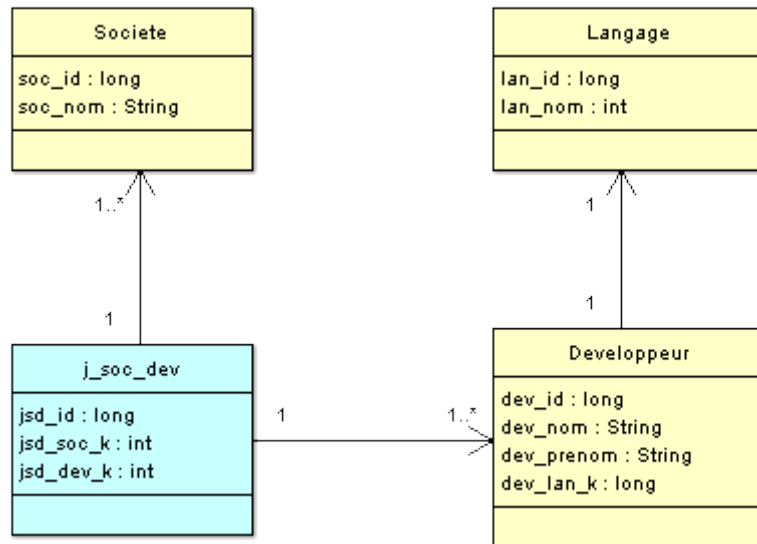
Pour l'exemple, nous allons voir en quoi consiste le mappage d'une table **Developpeur** avec un objet **Developpeur**



Vous pouvez voir qu'il est très simple de mapper une table avec un objet Java... Bon, il y a des cas plus compliqués, surtout lorsque les relations entre les tables sont de 1 à plusieurs ou de plusieurs à plusieurs. Mais pour le moment, la question n'est pas là...

Nous avons donc défini ce qu'est un mappage tables - objets. Je vous propose donc de voir quelle structure de base de données nous allons utiliser afin de mettre nos exemples en pratique.

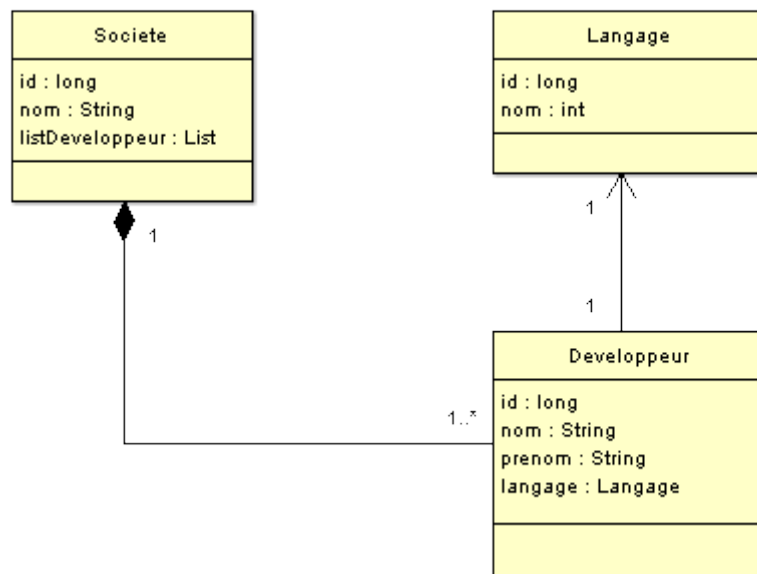
Afin de pouvoir illustrer les exemples de codes, nous allons avoir besoin d'une base de données contenant des tables et des données... Par souci de simplicité, je vous ai préparé une base, la voici :



Cette base de données est rudimentaire mais elle permettra de pouvoir voir les différents cas qui nous intéressent ! Vous pouvez voir :

- Qu'une société peut avoir 1 ou plusieurs développeurs.
- Qu'un développeur utilise un et un seul langage de programmation.

Avec cette structure de base, nous pouvons en déduire cette structure de classes Java :



La composition est remplacée par une collection dans la classe Societe.java et la contrainte de clé étrangère de la table Developpeur se transforme en un objet Langage dans la classe Developpeur.java. Ceci permet de remplir la signification des contraintes d'intégrités des tables de la base :

- Qu'une société peut avoir 1 ou plusieurs développeur(s).
- Qu'un développeur utilise (a) un et un seul langage de programmation.

Voici le script SQL de création de ces tables (j'espère qu'il conviendra à plusieurs SGBD...)

Script SQL de création

```
--
```

Script SQL de création

```
CREATE DATABASE "Societe" WITH TEMPLATE = template0 ENCODING = 'UTF8';
ALTER DATABASE "Societe" OWNER TO postgres;
CREATE TABLE developpeur (
    dev_id integer NOT NULL,
    dev_nom character varying(64),
    dev_prenom character varying(64),
    dev_lan_k bigint NOT NULL
);
ALTER TABLE public.developpeur OWNER TO postgres;
CREATE SEQUENCE developpeur_dev_id_seq
    INCREMENT BY 1
    NO MAXVALUE
    NO MINVALUE
    CACHE 1;
ALTER TABLE public.developpeur_dev_id_seq OWNER TO postgres;
ALTER SEQUENCE developpeur_dev_id_seq OWNED BY developpeur.dev_id;
SELECT pg_catalog.setval('developpeur_dev_id_seq', 3, true);

CREATE TABLE j_soc_dev (
    jsd_id integer NOT NULL,
    jsd_soc_k bigint NOT NULL,
    jsd_dev_k bigint NOT NULL
);
ALTER TABLE public.j_soc_dev OWNER TO postgres;
CREATE SEQUENCE j_soc_dev_jsd_id_seq
    INCREMENT BY 1
    NO MAXVALUE
    NO MINVALUE
    CACHE 1;
ALTER TABLE public.j_soc_dev_jsd_id_seq OWNER TO postgres;
ALTER SEQUENCE j_soc_dev_jsd_id_seq OWNED BY j_soc_dev.jsd_id;
SELECT pg_catalog.setval('j_soc_dev_jsd_id_seq', 5, true);

CREATE TABLE langage (
    lan_id integer NOT NULL,
    lan_nom character varying(64) NOT NULL
);
ALTER TABLE public.langage OWNER TO postgres;
CREATE SEQUENCE langage_lan_id_seq
    INCREMENT BY 1
    NO MAXVALUE
    NO MINVALUE
    CACHE 1;
ALTER TABLE public.langage_lan_id_seq OWNER TO postgres;
ALTER SEQUENCE langage_lan_id_seq OWNED BY langage.lan_id;
SELECT pg_catalog.setval('langage_lan_id_seq', 3, true);

CREATE TABLE societe (
    soc_id integer NOT NULL,
    soc_nom character varying(64) NOT NULL
);
ALTER TABLE public.societe OWNER TO postgres;
CREATE SEQUENCE societe_soc_id_seq
    INCREMENT BY 1
    NO MAXVALUE
    NO MINVALUE
    CACHE 1;
ALTER TABLE public.societe_soc_id_seq OWNER TO postgres;
ALTER SEQUENCE societe_soc_id_seq OWNED BY societe.soc_id;
SELECT pg_catalog.setval('societe_soc_id_seq', 2, true);

ALTER TABLE developpeur ALTER COLUMN
dev_id SET DEFAULT nextval('developpeur_dev_id_seq'::regclass);
ALTER TABLE j_soc_dev ALTER COLUMN jsd_id SET DEFAULT nextval('j_soc_dev_jsd_id_seq'::regclass);
ALTER TABLE langage ALTER COLUMN lan_id SET DEFAULT nextval('langage_lan_id_seq'::regclass);
ALTER TABLE societe ALTER COLUMN soc_id SET DEFAULT nextval('societe_soc_id_seq'::regclass);
```

Script SQL de création

```
INSERT INTO developpeur (dev_id, dev_nom, dev_prenom,
dev_lan_k) VALUES (1, 'HERBY', 'Cyrille', 1);
INSERT INTO developpeur (dev_id, dev_nom, dev_prenom,
dev_lan_k) VALUES (2, 'PITON', 'Thomas', 3);
INSERT INTO developpeur (dev_id, dev_nom, dev_prenom,
dev_lan_k) VALUES (3, 'COURTEL', 'Angelo', 2);

INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k) VALUES (1, 1, 1);
INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k) VALUES (2, 1, 2);
INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k) VALUES (3, 1, 3);
INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k) VALUES (4, 2, 1);
INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k) VALUES (5, 2, 3);

INSERT INTO langage (lan_id, lan_nom) VALUES (1, 'Java');
INSERT INTO langage (lan_id, lan_nom) VALUES (2, 'PHP');
INSERT INTO langage (lan_id, lan_nom) VALUES (3, 'C++');

INSERT INTO societe (soc_id, soc_nom) VALUES (1, 'Societe 1');
INSERT INTO societe (soc_id, soc_nom) VALUES (2, 'Societe 2');

ALTER TABLE ONLY developpeur
    ADD CONSTRAINT developpeur_pkey PRIMARY KEY (dev_id);
ALTER TABLE ONLY j_soc_dev
    ADD CONSTRAINT j_soc_dev_pkey PRIMARY KEY (jsd_id);
ALTER TABLE ONLY langage
    ADD CONSTRAINT langage_pkey PRIMARY KEY (lan_id);
ALTER TABLE ONLY societe
    ADD CONSTRAINT societe_pkey PRIMARY KEY (soc_id);

CREATE INDEX fki_ON developpeur USING btree (dev_lan_k);
CREATE INDEX fki_fki_developpeur ON j_soc_dev USING btree (jsd_dev_k);
CREATE INDEX fki_fki_societe ON j_soc_dev USING btree (jsd_soc_k);

ALTER TABLE ONLY developpeur
    ADD CONSTRAINT developpeur_dev_lan_k_fkey FOREIGN KEY (dev_lan_k) REFERENCES
langage(lan_id);
ALTER TABLE ONLY j_soc_dev
    ADD CONSTRAINT fki_developpeur FOREIGN KEY (jsd_dev_k) REFERENCES developpeur(dev_id);
ALTER TABLE ONLY j_soc_dev
    ADD CONSTRAINT fki_societe FOREIGN KEY (jsd_soc_k) REFERENCES societe(soc_id);
```

Voilà, nous avons une base de données et des données, il nous manque encore des objets mappant cette structure.



Ce script SQL provient d'un pgdump de ma base PostgreSQL ! Pour ceux n'utilisant pas PostgreSQL, vous devrez sûrement retravailler ce script.

Comme je vous le disais tout à l'heure, nous allons mapper les données de notre base de données dans des objets Java, le tout en respectant le diagramme de classes vu plus haut.

Langage.java

```
package com.developpez.bean;

public class Langage {
    private long id = 0;
    private String nom = "";

    public Langage() {}

    public Langage(long id, String nom) {
        this.id = id;
        this.nom = nom;
    }
}
```

Langage.java

```
}

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String toString(){
        return "LANGAGE DE PROGRAMMATION : " + this.nom;
    }
}
```

Developpeur.java

```
package com.developpez.bean;

public class Developpeur {
    private long id = 0;
    private String nom = "", prenom = "";
    private Langage langage = new Langage();

    public Developpeur() {}

    public Developpeur(long id, String nom, String prenom, Langage langage) {
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.langage = langage;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }

    public Langage getLangage() {
        return langage;
    }
}
```

Developpeur.java

```
}

public void setLangage(Language langage) {
    this.langage = langage;
}

public String toString(){
    String str = "NOM : " + this.getNom() + "\n";
    str += "PRENOM : " + this.getPrenom() + "\n";
    str += this.langage.toString();
    str += "\n.....\n";

    return str;
}
}
```

Societe.java

```
package com.developpez.bean;

import java.util.ArrayList;

public class Societe {

    private long id = 0;
    private String nom = "";
    private ArrayList<Developpeur> listDeveloppeur = new ArrayList<Developpeur>();

    public Societe(){}

    public Societe(long id, String nom, ArrayList<Developpeur> listDeveloppeur) {
        this.id = id;
        this.nom = nom;
        this.listDeveloppeur = listDeveloppeur;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public ArrayList<Developpeur> getListDeveloppeur() {
        return listDeveloppeur;
    }

    public void setListDeveloppeur(ArrayList<Developpeur> listDeveloppeur) {
        this.listDeveloppeur = listDeveloppeur;
    }

    public void addDeveloppeur(Developpeur dev){
        this.listDeveloppeur.add(dev);
    }

    public Developpeur getDeveloppeur(int indice){
        return this.listDeveloppeur.get(indice);
    }

    public String toString(){
```


Societe.java

```
String str = "*****\n";
str += "NOM : " + this.getNom() + "\n";
str += "*****\n";
str += "LISTE DES DEVELOPPEURS : \n";

for (Developpeur dev : this.listDeveloppeur)
    str += dev.toString() + "\n";

return str;
}
```

Nous avons maintenant nos objets Java et notre base de données, avant de voir comment fonctionne ce pattern de conception, il ne nous reste juste à voir comment nous allons implémenter la connexion à la base de données.

Afin de pouvoir gagner en souplesse et en allocation mémoire, nous allons utiliser le pattern singleton afin d'instancier et d'utiliser la connexion à la base de données utilisée. Il va de soit que vous savez vous connecter à une base de données via JDBC (Java DataBase Connectivity), si vous ne savez pas comment procéder, vous pouvez aller lire [ce tuto](#) ou [celui-ci](#).

Voici un bref rappel sur ce qu'est un singleton :

Il s'agit d'un objet dont le constructeur est déclaré **private**, ceci afin d'assurer que seule une instance de l'objet en question puisse être créée.

Voici le code source de mon singleton, celui-ci sert à créer une connexion vers la base PostgreSQL, modifiez le code source selon vos besoins :

Singleton de connexion à une base de donnée

```
package com.developpez.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConnectionPostgreSQL{

    /**
     * URL de connection
     */
    private static String url = "jdbc:postgresql://localhost:5432/Societe";
    /**
     * Nom du user
     */
    private static String user = "postgres";
    /**
     * Mot de passe du user
     */
    private static String passwd = "postgres";
    /**
     * Objet Connection
     */
    private static Connection connect;

    /**
     * Méthode qui va nous retourner notre instance
     * et la créer si elle n'existe pas...
     * @return
     */
    public static Connection getInstance(){
        if(connect == null){
            try {
                connect = DriverManager.getConnection(url, user, passwd);
            } catch (SQLException e) {
```

Singleton de connexion à une base de donnée

```
e.printStackTrace();
}
}
return connect;
}
}
```

Bon, nous avons maintenant tous les éléments nécessaires afin de travailler avec notre base de données via le pattern DAO, mais il nous reste à savoir ce que fait exactement ce pattern...

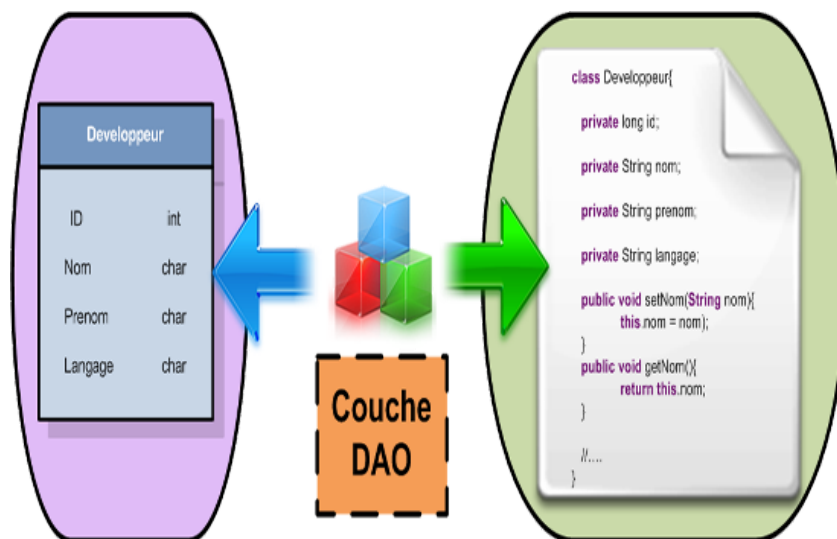
II - Le pattern DAO : Qu'est-ce que c'est ?

Le pattern DAO (Data Access Object) permet de faire le lien entre la couche métier et la couche persistante, ceci afin de centraliser les mécanismes de mapping entre notre système de stockage et nos objets Java. Il permet aussi de prévenir un changement éventuel de système de stockage de données (de PostgreSQL vers Oracle par exemple).

La couche persistante correspond, en fait, à notre système de stockage et la couche métier correspond à nos objets Java, mapper sur notre base. Le pattern DAO consiste à ajouter un ensemble d'objets dont le rôle sera d'aller :

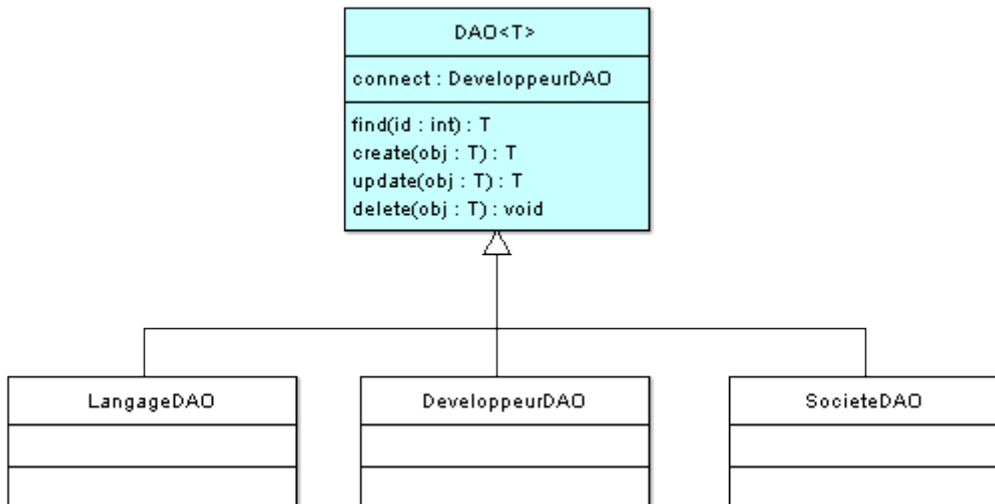
- Lire.
- Ecrire.
- Modifier.
- Supprimer.

dans notre système de stockage. Cet ensemble d'objet s'appelle la couche DAO. Voici, schématiquement, à quoi va ressembler notre structure :

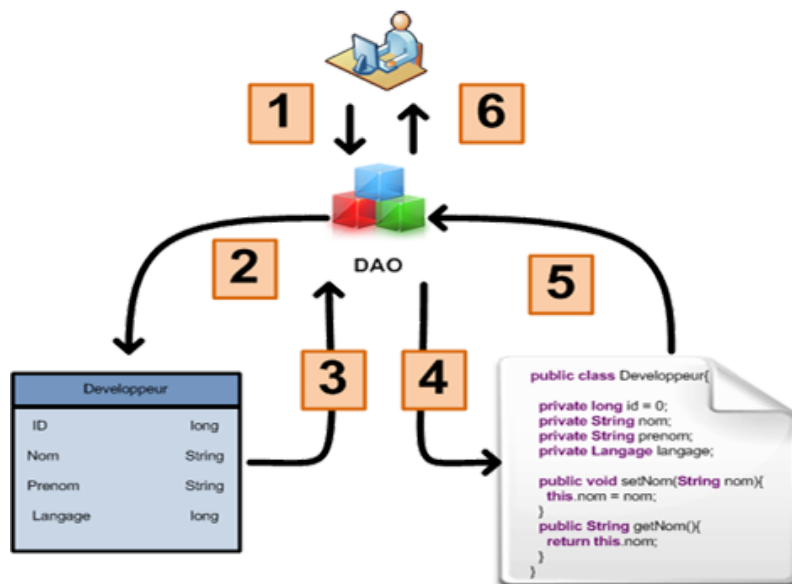


Ce sera donc par le biais de ces objets spécifiques que nous pourrions récupérer des instances de nos objets métiers correspondants à des entrées dans notre base de données... Nous pouvons donc déterminer la façon dont nos objets vont travailler, ceci parce que nous connaissons les actions que ces objets devront faire !

Un certain DAO devra s'occuper de créer, mettre à jour, lire, supprimer des données de la table "**Developpeur**", un autre DAO se chargera de la table "**Langage**" et un autre se chargera de la table "**Societe**". Evidemment, tout ceci devra être orchestré et c'est pour ceci que nous allons créer une classe abstraite permettant de créer des classes héritées. Voici le diagramme de classes que nous allons utiliser :



Afin de mieux comprendre comment le tout fonctionne, voici un schéma récapitulant les étapes d'utilisation d'un DAO.



- 1 L'application cliente (client riche ou client lourd) requière un objet **Developpeur** correspondant à l'entrée 1 dans la base de données : utilisation de l'objet **DeveloppeurDAO**.
- 2 L'objet **DeveloppeurDAO** récupère cette demande (méthode **find(1)**) : il s'occupe d'exécuter la requête SQL.
- 3 Le moteur SQL interprète la requête SQL et retourne le résultat attendu (ou pas...).
- 4 L'objet **DeveloppeurDAO** récupère ces informations.
- 5 L'objet **DeveloppeurDAO** instancie un objet **Developpeur** avec les informations nécessaires.
- 6 Enfin, l'objet **DeveloppeurDAO** retourne l'instance de l'objet **Developpeur**.

Voilà, vous venez de voir comment le tout va fonctionner. Vous vous demandez sûrement pourquoi faire tout ce ramdam pour récupérer des informations d'une base de données et instancier un objet. Le fait est que, en procédant de la sorte, vous encapsulez toute la partie mappage dans des objets particuliers et, si des changements sont à faire, ils ne se feront pas dans ce qui fonctionne (les objets métiers) mais dans les DAO. Ainsi, si vous devez passer d'un système de stockage en base de données vers un système de stockage en fichiers XML, rien de plus simple, on crée une nouvelle hiérarchie de DAO et c'est tout !

III - Les classes utilisées

Voici les classes que nous allons utiliser pour illustrer le tutoriel et nous allons commencer par la classe abstraite dont tous nos DAO vont hériter :

Classe DAO

```
package com.developpez.dao;

import java.sql.Connection;
import com.developpez.jdbc.ConnectionPostgreSQL;

public abstract class DAO<T> {

    public Connection connect = ConnectionPostgreSQL.getInstance();

    /**
     * Permet de récupérer un objet via son ID
     * @param id
     * @return
     */
    public abstract T find(long id);

    /**
     * Permet de créer une entrée dans la base de données
     * par rapport à un objet
     * @param obj
     */
    public abstract T create(T obj);

    /**
     * Permet de mettre à jour les données d'une entrée dans la base
     * @param obj
     */
    public abstract T update(T obj);

    /**
     * Permet la suppression d'une entrée de la base
     * @param obj
     */
    public abstract void delete(T obj);
}
```

Et voici le code source de nos DAO respectifs, mais je n'y ai pas encore mis le contenu des méthodes qui nous intéressent...

Classe LangageDAO

```
package com.developpez.dao.concret;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.developpez.bean.Langage;
import com.developpez.dao.DAO;

public class LangageDAO extends DAO<Langage> {

    public Langage create(Langage obj) {
        try {

            //Vu que nous sommes sous postgres, nous allons chercher manuellement
            //la prochaine valeur de la séquence correspondant à l'id de notre table
        }
    }
}
```

Classe LangageDAO

```

    ResultSet result = this .connect
                                .createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE
                                )
                                .executeQuery(
                                    "SELECT NEXTVAL('langage_lan_id_seq') as id"
                                );

    if(result.first()){
        long id = result.getLong("id");
        PreparedStatement prepare = this .connect
                                .prepareStatement(

"INSERT INTO langage (lan_id, lan_nom) VALUES(?, ?)"
                                );

        prepare.setLong(1, id);
        prepare.setString(2, obj.getNom());

        prepare.executeUpdate();
        obj = this.find(id);

    }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    }
    return obj;
}

public Langage find(long id) {
    Langage lang = new Langage();
    try {
        ResultSet result = this .connect
                                .createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE
                                )
                                .executeQuery(
                                    "SELECT * FROM langage WHERE lan_id = " + id
                                );

        if(result.first())
            lang = new Langage(
                                id,
                                result.getString("lan_nom")
                            );

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return lang;
}

public Langage update(Langage obj) {
    try {

        this .connect
            .createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE
            )
            .executeUpdate(
                "UPDATE langage SET lan_nom = '" + obj.getNom() + "'" +
                " WHERE lan_id = " + obj.getId()
            );

        obj = this.find(obj.getId());
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return obj;
}

```

Classe LangageDAO

```
}

public void delete(Langage obj) {
    try {

        this .connect
            .createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE
            ).executeUpdate(
                "DELETE FROM langage WHERE lan_id = " + obj.getId()
            );

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

}
```

Classe DeveloppeurDAO

```
package com.developpez.dao.concret;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.developpez.bean.Developpeur;
import com.developpez.bean.Langage;
import com.developpez.dao.AbstractDAOFactory;
import com.developpez.dao.DAO;
import com.developpez.dao.FactoryType;

public class DeveloppeurDAO extends DAO<Developpeur> {

    public Developpeur create(Developpeur obj) {

        try {

            //Si le langage n'existe pas en base, on le créé
            if(obj.getLangage().getId() == 0){
                DAO<Langage> langageDAO = AbstractDAOFactory .getFactory(FactoryType.DAO_FACTORY)
                                                                .getLangageDAO();

                obj.setLangage(langageDAO.create(obj.getLangage()));
            }
            //Vu que nous sommes sous postgres, nous allons chercher manuellement
            //la prochaine valeur de la séquence correspondant à l'id de notre table
            ResultSet result = this .connect
                .createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE
                ).executeQuery(
                    "SELECT NEXTVAL('developpeur_dev_id_seq') as id"
                );

            if(result.first()){
                long id = result.getLong("id");
                PreparedStatement prepare = this .connect
                    .prepareStatement(

                        "INSERT INTO developpeur (dev_id, dev_nom, dev_prenom, dev_lang_k)" +
                        "VALUES(?, ?, ?, ?)"
                    );

                prepare.setLong(1, id);
                prepare.setString(2, obj.getNom());
                prepare.setString(3, obj.getPrenom());
                prepare.setLong(4, obj.getLangage().getId());
            }
        }
    }
}
```

Classe DeveloppeurDAO

```
prepare.executeUpdate();
obj = this.find(id);

}
} catch (SQLException e) {
    e.printStackTrace();
}
return obj;
}

public Developpeur find(long id) {

    Developpeur dev = new Developpeur();
    try {
        ResultSet result = this .connect
                                .createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_READ_ONLY
                                ).executeQuery(
                                    "SELECT * FROM developpeur WHERE dev_id = " + id
                                );

        if(result.first())
            dev = new Developpeur(
                id,
                result.getString("dev_nom"),
                result.getString("dev_prenom"),
                new LangageDAO().find(result.getLong("dev_lang_k"))
            );

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return dev;
}

public Developpeur update(Developpeur obj) {

    try{
        DAO<Langage> langageDAO = AbstractDAOFactory .getFactory(FactoryType.DAO_FACTORY)
                                                    .getLangageDAO();

        //Si le langage n'existe pas en base, on le crée
        if(obj.getLangage().getId() == 0){
            obj.setLangage(langageDAO.create(obj.getLangage()));
        }
        //On met à jours l'objet Langage
        langageDAO.update(obj.getLangage());

        this.connect
            .createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE
            ).executeUpdate(
                "UPDATE developpeur SET dev_nom = '" + obj.getNom() + "'," +
                " dev_prenom = '" + obj.getPrenom() + "'," +
                " dev_lang_k = '" + obj.getLangage().getId() + "'" +
                " WHERE dev_id = " + obj.getId()
            );

        obj = this.find(obj.getId());
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return obj;
}

public void delete(Developpeur obj) {
    try {

        this.connect
```

Classe DeveloppeurDAO

```
.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE  
) .executeUpdate(  
    "DELETE FROM developpeur WHERE dev_id = " + obj.getId()  
);  
  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
}  
}
```

Classe SocieteDAO

```
package com.developpez.dao.concret;  
  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
  
import com.developpez.bean.Developpeur;  
import com.developpez.bean.Langage;  
import com.developpez.bean.Societe;  
import com.developpez.dao.AbstractDAOFactory;  
import com.developpez.dao.DAO;  
import com.developpez.dao.FactoryType;  
  
public class SocieteDAO extends DAO<Societe> {  
  
    public Societe create(Societe obj) {  
        try{  
  
            //Vu que nous sommes sous postgres, nous allons chercher manuellement  
            //la prochaine valeur de la séquence correspondant à l'id de notre table  
            ResultSet result = this .connect  
                .createStatement(  
                    ResultSet.TYPE_SCROLL_INSENSITIVE,  
                    ResultSet.CONCUR_UPDATABLE  
                ) .executeQuery(  
                    "SELECT NEXTVAL('societe_soc_id_seq') as id"  
                );  
  
            if(result.first()){  
                long id = result.getLong("id");  
                PreparedStatement prepare = this .connect  
                    .prepareStatement(  
                        "INSERT INTO societe (soc_id, soc_nom)" +  
                        "VALUES(?, ?)"  
                    );  
  
                prepare.setLong(1, id);  
                prepare.setString(2, obj.getNom());  
                prepare.executeUpdate();  
  
                //Maintenant, nous devons créer les liens vers les développeurs  
                //Si le développeur n'existe pas en base, on le crée  
                for(Developpeur dev : obj.getListDeveloppeur()){  
                    if(dev.getId() == 0){  
                        DAO<Developpeur> developpeurDAO = AbstractDAOFactory  
                            .getFactory(FactoryType.DAO_FACTORY)  
                                .getDeveloppeurDAO();  
                        dev = developpeurDAO.create(dev);  
                    }  
                }  
  
                //On récupère la prochaine valeur de la séquence  
                ResultSet result2 = this .connect
```


Classe SocieteDAO

```

        .createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE
        ).executeQuery(
            "SELECT NEXTVAL('j_soc_dev_jsd_id_seq') as id"
        );

    if(result2.first()){

        long id2 = result2.getLong("id");
        PreparedStatement prepare2 = this .connect
                                .prepareStatement(
                "INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k)" +
                " VALUES(?, ?, ?)"
            );

        prepare2.setLong(1, id2);
        prepare2.setLong(2, id);
        prepare2.setLong(3, dev.getId());
        prepare2.executeUpdate();
    }
}

obj = this.find(id);
}
} catch (SQLException e) {
    e.printStackTrace();
}
return obj;
}

public Societe find(long id) {
    Societe societe = new Societe();

    try {
        ResultSet result = this .connect
                                .createStatement(
                                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                                    ResultSet.CONCUR_UPDATABLE
                                ).executeQuery(
                                    "select * from societe " +
                                    " left join j_soc_dev on jsd_soc_k = soc_id AND soc_id = "+ id +
                                    " inner join developpeur on jsd_dev_k = dev_id"
                                );

        if(result.first()){
            DeveloppeurDAO devDao = new DeveloppeurDAO();
            ArrayList<Developpeur> listDeveloppeur = new ArrayList<Developpeur>();

            result.beforeFirst();
            while(result.next() && result.getLong("jsd_dev_k") != 0)
                listDeveloppeur.add(devDao.find(result.getLong("jsd_dev_k")));

            result.first();
            societe = new Societe(id, result.getString("soc_nom"), listDeveloppeur);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return societe;
}

public Societe update(Societe obj) {

    try{

        //On met à jours la liste des développeurs au cas ou
        PreparedStatement prepare = this .connect
                                .prepareStatement(

```

Classe SocieteDAO

```

        "UPDATE societe SET soc_nom = '" + obj.getNom() + "'" +
        " WHERE soc_id = " + obj.getId()
    );

    prepare.executeUpdate();

    //Maintenant, nous devons créer les liens vers les développeurs
    //Si le développeur n'existe pas en base, on le crée
    for(Dveloppeur dev : obj.getListDeveloppeur()){

        DAO<Developpeur> developpeurDAO = AbstractDAOFactory
            .getFactory(FactoryType.DAO_FACTORY)
            .getDeveloppeurDAO();

        //Si l'objet n'existe pas, on le crée avec sa jointure
        if(dev.getId() == 0){

            dev = developpeurDAO.create(dev);

            //On récupère la prochaine valeur de la séquence
            ResultSet result2 = this .connect
                .createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE
                ).executeQuery(
                    "SELECT NEXTVAL('j_soc_dev_jsd_id_seq') as id"
                );

            if(result2.first()){

                long id2 = result2.getLong("id");
                PreparedStatement prepare2 = this .connect
                    .prepareStatement(
                        "INSERT INTO j_soc_dev (jsd_id, jsd_soc_k, jsd_dev_k)" +
                        "VALUES(?, ?, ?)"
                    );

                prepare2.setLong(1, id2);
                prepare2.setLong(2, obj.getId());
                prepare2.setLong(3, dev.getId());
                prepare2.executeUpdate();
            }

        }
        else{
            developpeurDAO.update(dev);
        }

    }

    obj = this.find(obj.getId());

} catch (SQLException e) {
    e.printStackTrace();
}

return obj;
}

public void delete(Societe obj) {

    try {

        this.connect
            .createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE
            ).executeUpdate(
                "DELETE FROM j_soc_dev WHERE jsd_soc_k = " + obj.getId()
            );

    }

```

Classe SocieteDAO

```
this.connect()
    .createStatement(
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE
    ).executeUpdate(
        "DELETE FROM societe WHERE soc_id = " + obj.getId()
    );

} catch (SQLException e) {
    e.printStackTrace();
}

}
```

Maintenant que nous avons nos DAO, nous allons pouvoir faire nos premiers tests !



Les plus avisés d'entre vous auront remarqué que les contrôles d'usages ne sont pas présents dans nos DAO. Par là j'entends le bon déroulement de la requête ainsi que la vérification de l'existence d'un enregistrement lors d'une recherche... Enfin, ce genre de choses.

Voici un exemple de code utilisant un DAO :

Code de test

```
import com.developpez.bean.Societe;
import com.developpez.dao.DAO;
import com.developpez.dao.concret.SocieteDAO;

public class Main {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DAO<Societe> societeDao = new SocieteDAO();
        for(long i = 1; i <= 2; i++)
            System.out.println(societeDao.find(i));
    }

}
```

Voici le résultat que me donne ce code :

```
*****
ID : 1
NOM : Societe 1
*****
LISTE DES DEVELOPPEURS :
ID : 1
NOM : HERBY
PRENOM : Cyrille
ID : 1 - LANGAGE DE PROGRAMMATION : Java
.....

ID : 2
NOM : PITON
PRENOM : Thomas
ID : 3 - LANGAGE DE PROGRAMMATION : C++
.....

ID : 3
NOM : COURTEL
PRENOM : Angelo
ID : 2 - LANGAGE DE PROGRAMMATION : PHP
.....

*****
ID : 2
NOM : Societe 2
*****
LISTE DES DEVELOPPEURS :
ID : 1
NOM : HERBY
PRENOM : Cyrille
ID : 1 - LANGAGE DE PROGRAMMATION : Java
.....

ID : 3
NOM : COURTEL
PRENOM : Angelo
ID : 2 - LANGAGE DE PROGRAMMATION : PHP
.....
```

Notre mapping fonctionne à merveille. Bon, bien sûr, si votre base de données est plus conséquente, il sera peut-être onéreux de charger toutes les dépendances d'objets... Après, il n'en tient qu'à vous de faire fonctionner vos DAO comme vous le souhaitez.

Voici trois codes permettant de tester les autres méthodes de nos DAO :

Test des autres méthodes de LangageDAO

```
import java.sql.SQLException;

import com.developpez.bean.Langage;
import com.developpez.dao.AbstractDAOFactory;
import com.developpez.dao.DAO;
import com.developpez.dao.FactoryType;
import com.developpez.jdbc.ConnectionPostgreSQL;
```

Test des autres méthodes de LangageDAO

```
public class Main2 {

    public static void main(String[] args){

        try {
            ConnectionPostgreSQL.getInstance().setAutoCommit(false);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        DAO<Langage> langageDAO = AbstractDAOFactory .getFactory(FactoryType.DAO_FACTORY)
                                                    .getLangageDAO();

        System.out.println("\n Avant création d'un nouveau langage :");
        for(int i = 1; i < 5; i++)
            System.out.println(langageDAO.find(i).toString());
        /*
        Langage lan = new Langage();
        lan.setNom("ActionScript");
        lan = langageDAO.create(lan);
        */

        System.out.println("\n Après création d'un nouveau langage :");
        for(int i = 1; i < 5; i++)
            System.out.println(langageDAO.find(i).toString());

        lan.setNom("ActionScript 2");
        lan = langageDAO.update(lan);

        System.out.println("\n Après mise à jour de l'objet langage :");
        for(int i = 1; i < 5; i++)
            System.out.println(langageDAO.find(i).toString());

        langageDAO.delete(lan);
        System.out.println("\n Après suppression l'objet langage :");
        for(int i = 1; i < 5; i++)
            System.out.println(langageDAO.find(i).toString());

    }

}
```

Test des autres méthodes de DeveloppeurDAO

```
import java.sql.SQLException;

import com.developpez.bean.Developpeur;
import com.developpez.bean.Langage;
import com.developpez.dao.AbstractDAOFactory;
import com.developpez.dao.DAO;
import com.developpez.dao.FactoryType;
import com.developpez.jdbc.ConnectionPostgreSQL;

public class Main3 {

    public static void main(String[] args){

        try {
            ConnectionPostgreSQL.getInstance().setAutoCommit(false);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        DAO<Developpeur> developpeurDAO = AbstractDAOFactory .getFactory(FactoryType.DAO_FACTORY)
                                                            .getDeveloppeurDAO();
```

Test des autres méthodes de DeveloppeurDAO

```
System.out.println("\n Avant création d'un nouveau développeur :");
for(int i = 3; i < 5; i++)
    System.out.println(developpeurDAO.find(i).toString());
/*
Developpeur dev = new Developpeur();
dev.setNom("Coquille");
dev.setPrenom("Olivier");
Langage lan = new Langage();
lan.setNom("COBOL");
dev.setLangage(lan);

dev = developpeurDAO.create(dev);
//*/

System.out.println("\n Après création d'un nouveau développeur :");
for(int i = 3; i < 5; i++)
    System.out.println(developpeurDAO.find(i).toString());

dev.setNom("MERLET");
dev.setPrenom("Benoit");
lan.setNom("4gl");
dev.setLangage(lan);
dev = developpeurDAO.update(dev);

System.out.println("\n Après mise à jour de l'objet développeur :");
for(int i = 3; i < 5; i++)
    System.out.println(developpeurDAO.find(i).toString());

developpeurDAO.delete(dev);
System.out.println("\n Après suppression l'objet développeur :");
for(int i = 3; i < 5; i++)
    System.out.println(developpeurDAO.find(i).toString());

}

}
```

Test des autres méthodes de SocieteDAO

```
import java.sql.SQLException;

import com.developpez.bean.Developpeur;
import com.developpez.bean.Langage;
import com.developpez.bean.Societe;
import com.developpez.dao.AbstractDAOFactory;
import com.developpez.dao.DAO;
import com.developpez.dao.FactoryType;
import com.developpez.jdbc.ConnectionPostgreSQL;

public class Main4 {

    public static void main(String[] args) {

        /*
        try {
            ConnectionPostgreSQL.getInstance().setAutoCommit(false);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        //*/

        DAO<Societe> societeDAO = AbstractDAOFactory .getFactory(FactoryType.DAO_FACTORY)
                                                    .getSocieteDAO();

        System.out.println("\n Avant création d'une nouvelle société :");
        for(int i = 2; i < 4; i++)
            System.out.println(societeDAO.find(i));
```

Test des autres méthodes de SocieteDAO

```

Societe societe = new Societe();
societe.setNom("SUN");

Developpeur dev1 = new Developpeur();
dev1.setNom("Bugault");
dev1.setPrenom("Noël");
Langage lan = new Langage();
lan.setNom("Windev");
dev1.setLangage(lan);

societe.addDeveloppeur(dev1);
societe = societeDAO.create(societe);

System.out.println("\n Après création d'une nouvelle société :");
for(int i = 2; i < 4; i++)
    System.out.println(societeDAO.find(i));

societe.setNom("SUN Microsystem");

Developpeur dev2 = new Developpeur();
dev2.setNom("MAHE");
dev2.setPrenom("Marie-pierre");
Langage lan2 = new Langage();
lan2.setNom("Fortran");
dev2.setLangage(lan2);

societe.addDeveloppeur(dev2);

societe = societeDAO.update(societe);

System.out.println("\n Après modification de la société :");
for(int i = 2; i < 4; i++)
    System.out.println(societeDAO.find(i));

societeDAO.delete(societe);

System.out.println("\n Après suppression de la société :");
for(int i = 2; i < 4; i++)
    System.out.println(societeDAO.find(i));

}

}

```

*Je vous laisse voir ce que font ces codes... Sinon, pour informations, j'ai désactivé l'auto-commit afin de ne pas avoir supprimer les lignes dans la base de données à la main. Par contre, vu que je suis sous PostgreSQL, vous devrez soit augmenter vos incréments de boucle, soit réinitialiser vos séquences de tables... Vous constaterez aussi que j'ai dû coder la suppression de certaines données à la main dans certain DAO, ceci est dû au fait que je n'ai pas mis de **DELETE CASCADE** sur les clés étrangères.*

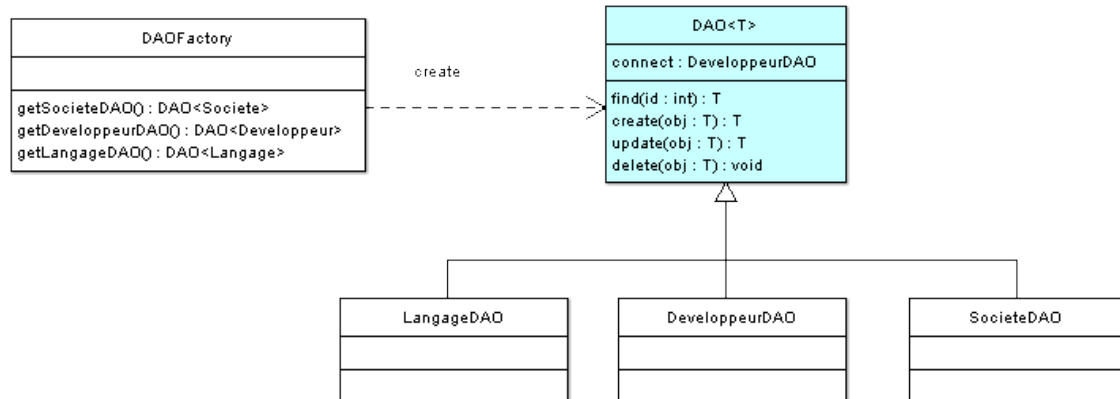
Par contre, je vous disais plus haut que ce pattern permettait de favoriser la migration vers un autre système de stockage, pour ce faire, un autre pattern est utilisé : le pattern factory. Je vous propose de vous faire faire un tour de cette implémentation.

IV - Allez plus loin avec le pattern Factory

Pour plus de souplesse, le pattern DAO sera couplé avec le pattern Factory. Pour rappel, le pattern Factory permet d'encapsuler l'instanciation de nos objets dans une classe.

Ceci permet de prévenir des modifications sur la façon de créer les objets concrets car l'instanciation de nos DAO sera centralisée dans un seul objet.

Voici un diagramme de classe schématisant le pattern Factory combiné avec le pattern DAO :



Voici le code source de cette classe dont le rôle est uniquement la création de nos DAO.

Factory de DAO

```

package com.developpez.dao;

import com.developpez.bean.Developpeur;
import com.developpez.bean.Langage;
import com.developpez.bean.Societe;
import com.developpez.dao.concret.DeveloppeurDAO;
import com.developpez.dao.concret.LangageDAO;
import com.developpez.dao.concret.SocieteDAO;

public class DAOFactory {

    public static DAO<Societe> getSocieteDAO() {
        return new SocieteDAO();
    }

    public static DAO<Developpeur> getDeveloppeurDAO() {
        return new DeveloppeurDAO();
    }

    public static DAO<Langage> getLangageDAO() {
        return new LangageDAO();
    }
}

```

Une fois ceci fait, il n'y a pas grand chose à changer afin de se servir de notre factory. Voici le code d'exemple utilisé tout à l'heure, mais cette fois, au lieu d'instancier directement un DAO, nous passons par la factory :

Code de test

```

import com.developpez.bean.Societe;
import com.developpez.dao.DAO;
import com.developpez.dao.DAOFactory;
import com.developpez.dao.concret.SocieteDAO;

public class Main {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        DAO<Societe> societeDao = DAOFactory.getSocieteDAO();
        for(long i = 1; i <= 2; i++)
            System.out.println(societeDao.find(i));
    }
}

```


Code de test

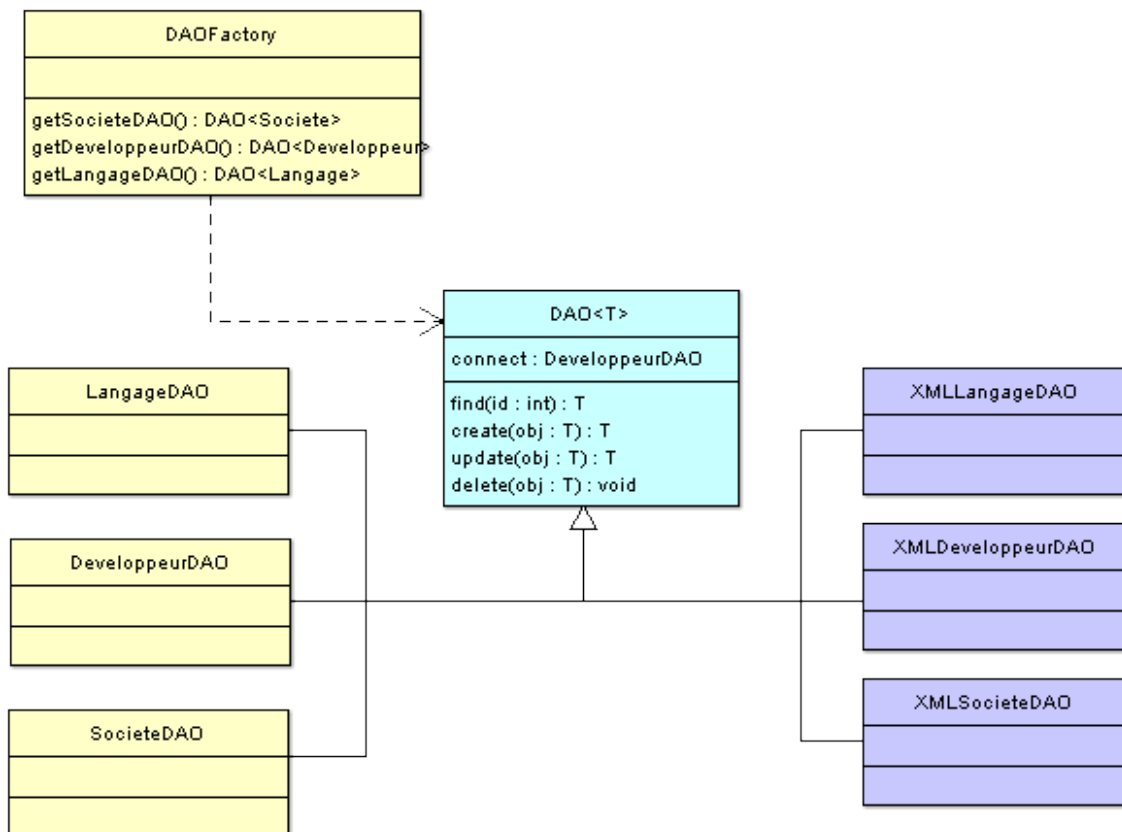
```
}  
}
```

Le jeu d'essai reste inchangé par rapport à tout à l'heure, donc nous avons réussi à encapsuler l'instanciation de nos DAO dans un objet.

Vous me direz que ceci ne permet pas ce que je vous avais promis plus haut : la possibilité de changer la façon de stocker nos données en réduisant l'impact sur notre programme.

Afin de pouvoir faire en sorte de faire cohabiter un ou plusieurs systèmes de stockage de données sur une même application, ce qui permet, au final, d'assouplir la façon dont nos objets métiers sont mappés avec des données et donc de permettre un changement de système en douceur, nous allons déléguer le choix de nos DAO à un super objet, d'un niveau hiérarchique supérieur à notre factory.

En effet, notre factory actuel ne sert qu'à instancier des DAO permettant d'agir avec la base de données postgreSQL et le but n'est pas là, bien au contraire... Vous l'aurez sûrement deviné, si nous voulons pouvoir interagir avec un autre système de données, nous allons devoir créer de nouveaux DAO, spécialisés dans ce fameux système. Bien entendu, ces objets héritent aussi de la classe **DAO<T>**. Voici le diagramme de classe de tout ceci :



Voici les classes (vides) de nos nouvelles implémentations

XMLLangageDAO

```
package com.developpez.dao.concret;

import com.developpez.bean.Langage;
import com.developpez.dao.DAO;

public class XMLLangageDAO extends DAO<Langage> {
    public Langage create(Langage obj) {
        // TODO Auto-generated method stub
        return obj;
    }
}
```

XMLLangageDAO

```
}  
  
public void delete(Langage obj) {  
    // TODO Auto-generated method stub  
}  
  
public Langage find(long id) {  
    // TODO Auto-generated method stub  
    return null;  
}  
  
public Langage update(Langage obj) {  
    // TODO Auto-generated method stub  
    return obj;  
}  
}
```

XMLDeveloppeurDAO

```
package com.developpez.dao.concret;  
  
import com.developpez.bean.Developpeur;  
import com.developpez.dao.DAO;  
  
public class XMLDeveloppeurDAO extends DAO<Developpeur> {  
    public Developpeur create(Developpeur obj) {  
        // TODO Auto-generated method stub  
        return obj;  
    }  
    public void delete(Developpeur obj) {  
        // TODO Auto-generated method stub  
    }  
    public Developpeur find(long id) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
    public Developpeur update(Developpeur obj) {  
        // TODO Auto-generated method stub  
        return obj;  
    }  
}
```

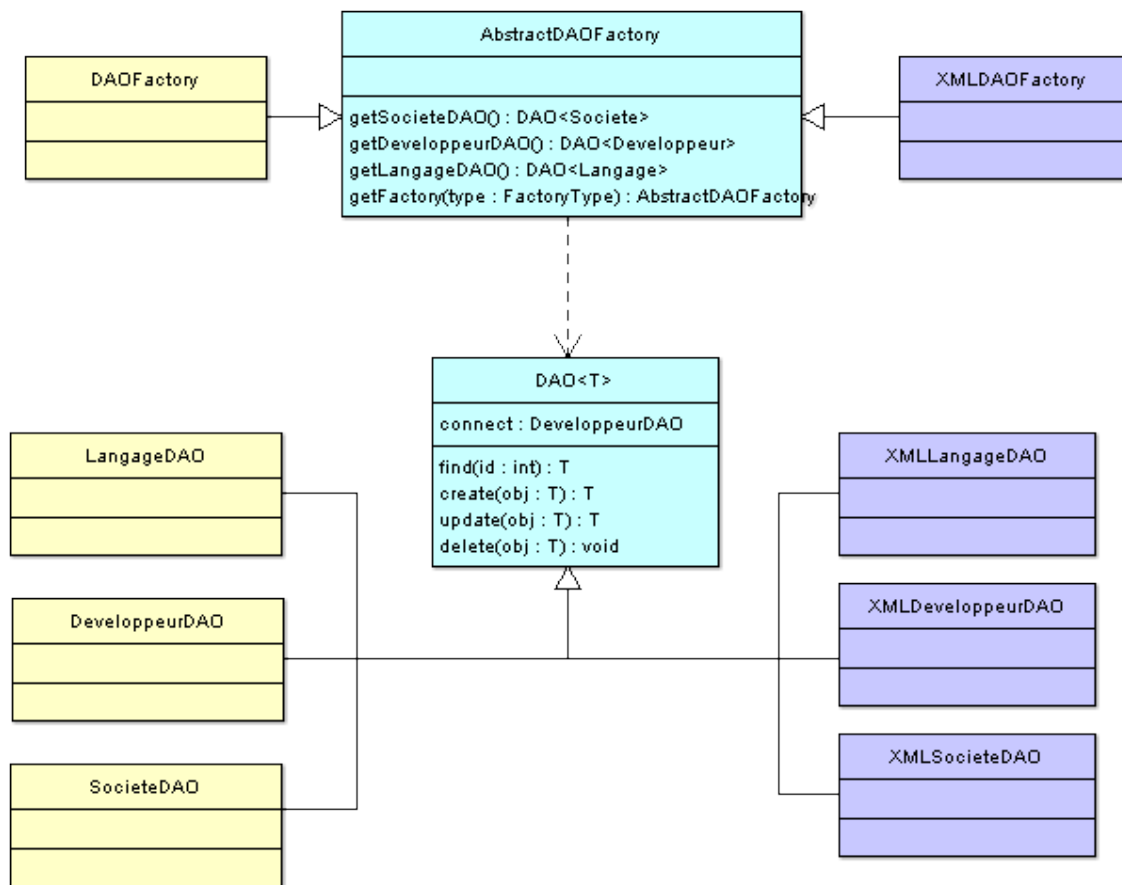
XMLSocieteDAO

```
package com.developpez.dao.concret;  
  
import com.developpez.bean.Societe;  
import com.developpez.dao.DAO;  
  
public class XMLSocieteDAO extends DAO<Societe> {  
    public Societe create(Societe obj) {  
        // TODO Auto-generated method stub  
        return obj;  
    }  
    public void delete(Societe obj) {  
        // TODO Auto-generated method stub  
    }  
    public Societe find(long id) {  
        // TODO Auto-generated method stub  
        return new Societe();  
    }  
    public Societe update(Societe obj) {  
        // TODO Auto-generated method stub  
        return obj;  
    }  
}
```



Il ne tient qu'à vous de compléter ces classes. Ici, elles ne servent qu'à titre d'exemple.

Avec ces nouvelles classes, spécialisées dans la communication avec des fichiers XML, nous allons avoir besoin d'une nouvelle factory dont le rôle sera de retourner les instances de ces fameux objets. Voici un diagramme de classes représentant tout ceci :



Et voici le code source de nos factory (oui, la première factory a un peu changée du coup)... Vu que nous allons avoir plusieurs objets fonctionnant de la même manière, nous allons créer une super classe dont nos factory vont héritées. Nous allons aussi rajouter une méthode dans cette super classe, cette dernière aura pour rôle de nous retourner la factory avec laquelle nous souhaitons travailler. Afin de bien contrôler ce que doit retourner notre factory de factory, nous allons utiliser une énumération Java.

Enumeration Java pour controler les type de factory

```
package com.developpez.dao;

public enum FactoryType {
    DAO_FACTORY, XML_DAO_Factory;
}
```

Super Factory

```
package com.developpez.dao;

public abstract class AbstractDAOFactory {

    public abstract DAO getSocieteDAO();
    public abstract DAO getDeveloppeurDAO();
    public abstract DAO getLangageDAO();

    /**
     * Méthode nous permettant de récupérer une factory de DAO
     * @param type
     * @return AbstractDAOFactory
     */
}
```

Super Factory

```
*/
public static AbstractDAOFactory getFactory(FactoryType type) {

    if(type.equals(type.DAO_FACTORY))
        return new DAOFactory();

    if(type.equals(type.XML_DAO_Factory))
        return new XMLDAOFactory();

    return null;
}
}
```

DAOFactory

```
package com.developpez.dao;

import com.developpez.bean.Developpeur;
import com.developpez.bean.Langage;
import com.developpez.bean.Societe;
import com.developpez.dao.concret.DeveloppeurDAO;
import com.developpez.dao.concret.LangageDAO;
import com.developpez.dao.concret.SocieteDAO;

public class DAOFactory extends AbstractDAOFactory{

    public DAO<Societe> getSocieteDAO() {
        return new SocieteDAO();
    }
    public DAO<Developpeur> getDeveloppeurDAO() {
        return new DeveloppeurDAO();
    }
    public DAO<Langage> getLangageDAO() {
        return new LangageDAO();
    }
}
```

XMLDAOFactory

```
package com.developpez.dao;

import com.developpez.dao.concret.XMLDeveloppeurDAO;
import com.developpez.dao.concret.XMLLangageDAO;
import com.developpez.dao.concret.XMLSocieteDAO;

public class XMLDAOFactory extends AbstractDAOFactory {
    public DAO getDeveloppeurDAO() {
        return new XMLDeveloppeurDAO();
    }
    public DAO getLangageDAO() {
        return new XMLLangageDAO();
    }
    public DAO getSocieteDAO() {
        return new XMLSocieteDAO();
    }
}
```

Le fonctionnement de cette nouvelle hiérarchie est très simple en fin de compte, la classe **AbstractDAOFactory** se chargera de retourner la factory dont nous avons besoin (factory de DAO standard, ou factory de DAO XML) et chaque factory est spécialisée dans ses propres DAO.

Du coup, l'invocation de nos DAO est quelque peu changée, mais leur fonctionnement reste identique.

Voici le code de test modifié pour cette nouvelle hiérarchie :

Code de test

Code de test

```
import com.developpez.bean.Societe;
import com.developpez.dao.AbstractDAOFactory;
import com.developpez.dao.DAO;
import com.developpez.dao.DAOFactory;
import com.developpez.dao.FactoryType;
import com.developpez.dao.concret.SocieteDAO;

public class Main {

    public static void main(String[] args) {

        //Récupération de la factory fabricant des DAO travaillant avec PostgreSQL
        DAO<Societe> societeDao =
        AbstractDAOFactory.getFactory(FactoryType.DAO_FACTORY).getSocieteDAO();
        //Récupération de la factory fabricant des DAO travaillant avec des fichiers XML
        DAO<Societe> xmlSocieteDao =
        AbstractDAOFactory.getFactory(FactoryType.XML_DAO_Factory).getSocieteDAO();

        for(long i = 1; i <= 2; i++)
            System.out.println(societeDao.find(i));

        for(long i = 1; i <= 2; i++)
            System.out.println(xmlSocieteDao.find(i));
    }
}
```

Et voilà le résultat du code de test ci-dessus :

```
Utilisation de la factory de DAO PostgreSQL :
*****
NOM : Societe 1
*****
LISTE DES DEVELOPPEUR :
NOM : PITON
PRENOM : Thomas
LANGAGE DE PROGRAMMATION : C++
.....

NOM : COURTEL
PRENOM : Angelo
LANGAGE DE PROGRAMMATION : PHP
.....

*****
NOM : Societe 2
*****
LISTE DES DEVELOPPEUR :
NOM : COURTEL
PRENOM : Angelo
LANGAGE DE PROGRAMMATION : PHP
.....

Utilisation de la factory de DAO XML :
*****
NOM :
*****
LISTE DES DEVELOPPEUR :

*****
NOM :
*****
LISTE DES DEVELOPPEUR :
```

V - Conclusion

Ce tutoriel touche à sa fin... Vous avez vu comment encapsuler la liaison avec un système de données et des objets métiers, ceci grâce au pattern DAO. Vous avez aussi vu comment améliorer celui-ci en le combinant avec le pattern factory. Avec cette nouvelle version du pattern DAO, vous pouvez utiliser plusieurs systèmes de données sans empiéter sur un autre. De plus, si vous avez à changer de système de stockage, vous pouvez tester la nouvelle implémentation sans toucher à l'ancienne et, pour mettre la nouvelle en place, un simple changement dans la factory générale et le tour est joué !

VI - Remerciement

Un grand merci à **Baptiste Wicht** pour sa relecture et ses remarques.