LE DESIGN PATTERN DAO

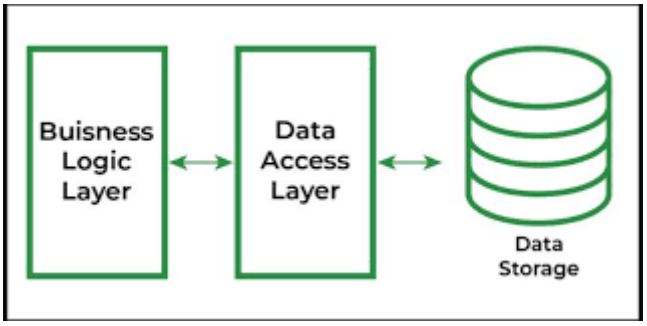
(Data Access Object)

Table des matières

Le modèle DAO propose de regrouper les accès aux données persistantes dans des classes à part, plutôt que de les disperser.

Il s'agit surtout de ne pas écrire ces accès dans les classes "métier", qui ne seront modifiées que si les règles de gestion métier changent.

L'utilisation de DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application.



https://fr.wikipedia.org/wiki/Objet_d%27acc%C3%A8s_aux_donn%C3%A9es

Vous allez développer une DAO dans votre projet clients/sprospects.

Cette couche DAO consistera à se connecter à une base de données MySql, d'aller chercher des données et de les mettre à jour.

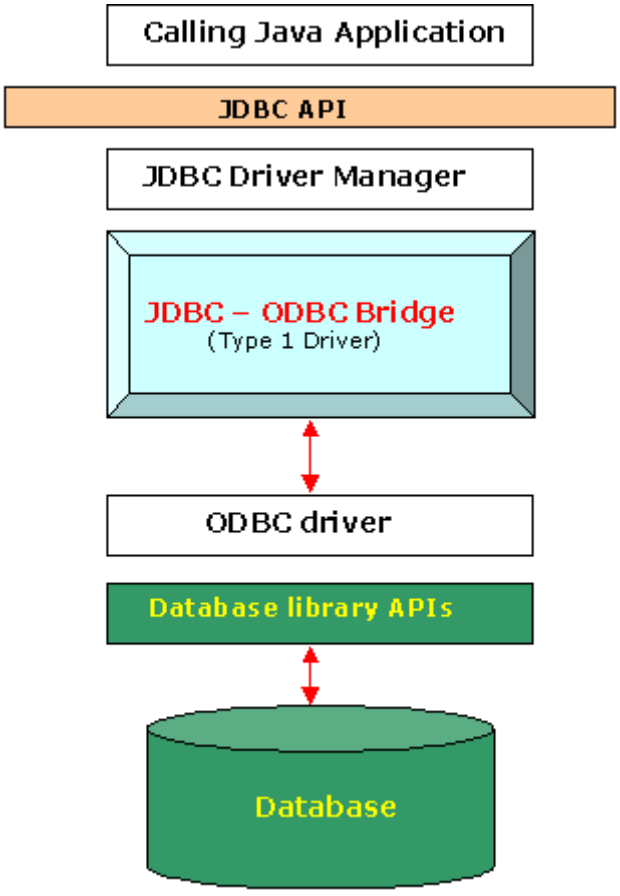
Java étant un langage de programmation orienté objet, la DAO fournira à l'application **des objets** et l'application fournira à la DAO **des objets**.

1. Java DataBase Connectivity (JDBC)

JDBC (***Java Database Connectivity***) est une interface de programmation pour les programmes utilisant la plateforme Java.

Elle permet aux applications Java d'accéder par le biais d'une interface commune à des sources de données pour lesquelles il existe des pilotes JDBC.

JDBC utilise ODBC (Open DataBase Connectivity) qui est un intergiciel (middleware) qui permet à une application informatique, de manipuler plusieurs bases de données qui sont mises à disposition par des systèmes de gestion de bases de données (SGBD), SQL ou NoSQL, ayant chacun un procédé propre.



Nous aurons donc à passer par le JDBC Driver Manager, inclus dans l'API Java JDBC pour faire le lien entre notre application Java et la ou les bases de données.

2. Installation de Wampserver

Wampserver est un environnement de développement qui inclus : Apache2 (serveur Web), PHP et MySQL

Taper Wampserver URL http://www.wampserver.com/

Cliquez sur EXPERIMENTER WAMPSERVER -> version 64 bits -> passer au téléchargement direct

Une fois installé, redémarrer votre ordinateur.

Démarrer l'application Wamp. Un icone apparait en bas à droite de votre écran dans l'onglet ^ (afficher les icônes cachées)

Cliquez sur le W qui doit être vert. Si il n'y est pas, cliquez dessus et démarrer les services.

Cliquez sur phpMyAdmin. Rentrez votre mot de passe.

Pour créer une base de données, cliquez sur « Nouvelle base de données » à gauche. Rentrez votre base de données et choisissez « utf8mb4_0900_ai_ci » si vous êtes en version 8 de MySQL, sinon chercher dans la liste utf8 et ci.

Pour créer une table, cliquez sur Nouvelle table et créez la structure de la table.

Choisissez des tables InnoDB, sinon les relations des clés étrangères ne fonctionneront pas

[MySQL] MyISAM vs InnoDB

Après avoir installé Wamp, vous pourrez choisir un autre client pour MySQL comme HeidiSQL, Workbench, DBeaver, …

3. Ajout du driver Mysql dans votre projet

a) Téléchargement

https://www.mysql.com/products/connector/

JDBC Driver for MySQL (Connector/J) Download

Select Operating System : Platform independent

Connector/J 9.2.0

Platform Independent (Architecture Independent), ZIP Archive

Dézipper

b) Ajouter le .jar dans votre projet

Mettre le .jar dans votre dossier projet

Allez dans

File -> Projet Structure -> Dependencies (à droite de Sources et Paths)

+ JARs or Directories

Sélectionner le .jar

Vérifier dans Librairies que vous avez bien la librairie

4. Les classes utilisées pour accéder aux données la base de données

Il y a 5 classes importantes : DriverManager, Connection, Statement, PreparedStatement, et ResultSet, chacune correspondant à une étape de l'accès aux données :

Classe	Rôle
DriverManager	Charger de configurer le driver de la base de données.
Connection	Réaliser la connexion et l'authentification à la base de données.
Statement (et PreparedStatement)	Contenir la requête SQL et la transmettre à la base de données.
ResultSet	Parcourir les informations retournées par la base de données dans le cas d'une sélection de données

Rq : Depuis une version récente de JDBC, les drivers, quand ils sont dans le classPath, sont automatiquement « découverts » par JDBC.

Rq : **Classpath** est un paramètre passé à une machine virtuelle Java qui définit le chemin d'accès au répertoire où se trouvent les classes et les packages Java afin qu'elle les exécute.

Au sein de la machine virtuelle, c'est le chargeur de classe qui parcourt les répertoires spécifiés par le classpath pour trouver où sont installées les classes requises par un programme Java

5. La connexion à la base de données

https://docs.oracle.com/javase/tutorial/jdbc/basics/connecting.html

Créer un package pour la DAO

La connexion à la base de données sera dans une classe particulière que l'on peut appeler Connexion ou ConnexionManager, ou tout autre nom significatif.

Cette classe contiendra un **attribut de classe**, la connexion, et **une méthode de classe retournant un objet de la classe Connection**.

En effet, la connexion est un objet de la classe Connection qui demande comme valeurs :

- L'URL du server de la base de données :
  par exemple pour MySQL
  jdbc:mysql://localhost/nomBaseDeDonnees?serverTimezone=UTC
  Ou
  jdbc:mysql://127.0.0.1/nomBaseDeDonnees?serverTimezone=UTC

- Le login et le password de la base de données (si connexion avec user et mot de passe)

Pour des raisons de sécurité et de mise en production, toutes ces informations ne doivent pas se trouver dans le code source de l'application.

On va donc créer un fichier database.properties que on crée en allant directement dans le dossier de votre projet.

Celui-ci est de la forme clé= valeur et pourrait ressembler à :

url=jdbc:mysql:// localhost /ecfpoo?serverTimezone=UTC

(ou url=jdbc:mysql://127.0.0.1/ecfpoo?serverTimezone=UTC)

login=root

password=root

Forme de l'url : jdbc :(SGBD)://adresse IP du serveur/nom base de données ?serverTimezone=UTC (par défaut pour localhost)

Récupération dans la classe Connexion des valeurs des propriétés de la database :
On lit le fichier database.Properties avec FileInputStream qui retourne un flux de

données provenant d'un d'un fichier via la méthode getClass().getClassLoader() de la classe Java.lang.Object qui renvoie un objet de type Class.

final Properties dataProperties = new Properties();

File fichier = new File("database.properties");
FileInputStream input = new FileInputStream(fichier);
dataProperties.load(input);

On créera la connexion à l'aide de la classe DriverManager et de sa méthode getConnection() qui peut lever une SQLException

Connection connection = DriverManager.getConnection(url, "myLogin", "myPassword");

Dans notre cas :

private static Connection connection = DriverManager.getConnection(

dataProperties.getProperty("url"),

dataProperties.getProperty("login"),

dataProperties.getProperty("password")

);

Ne pas oublier de gérer les exceptions

Fermeture de la connexion à la base de données à l'arrêt de l'application :

On utilisera un hook (crochet) : opportunité laissée au programmeur ou à l'utilisateur de modifier le fonctionnement d'un code préexistant.

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Runtime.html

// La méthode run() est appelée quand le programme se termine normalement, suite à une interruption de l'utilisateur ou un shutdown

```
Static {
Runtime.getRuntime().addShutdownHook(new Thread()
      {
         public void run()
         {
            if (connection != null) {
               try {
                  LOGGER.info("Database fermée");
                  connection.close();
               } catch (SQLException ex) {
                  LOGGER.fatal(ex.getMessage());
```

```
                }
            }
        }
    });
}
```

6. Les accès à la base de données : Statement (et PreparedStatement)

Pour des raisons de sécurité et notamment pour empêcher les injections SQL, nous utiliserons les requêtes préparées plutôt que les caractères spéciaux (https://fr.wikipedia.org/wiki/Injection_SQL)

a) Sélection

Doc oracle :
https://docs.oracle.com/javase/tutorial/jdbc/basics/processingsqlstatements.html

```
public static void viewTable(Connection con, String dbName)
    throws SQLException {

    Statement stmt = null;
    String query = "select COF_NAME, SUP_ID, PRICE, " +
                   "SALES, TOTAL " +
                   "from " + dbName + ".COFFEES";
    try {
        stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + "\t" + supplierID +
                               "\t" + price + "\t" + sales +
                               "\t" + total);
        }
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
    } finally {
        if (stmt != null) { stmt.close(); }
    }
}
```

Statement : L'interface Statement représente une instruction SQL. L'obtention d'une instance de cette interface se fait à partir de la Connection

```
Statement stmt = null;

stmt = con.createStatement();
```

Le résultat de la requête est récupéré dans un ResultSet : l'accès se fait enregistrement par enregistrement

b) Mise à jour

Doc oracle https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

```
public void updateCoffeeSales(HashMap<String, Integer> salesForWeek)
    throws SQLException {

    PreparedStatement updateSales = null;
    PreparedStatement updateTotal = null;

    String updateString =
        "update " + dbName + ".COFFEES " +
        "set SALES = ? where COF_NAME = ?";

    String updateStatement =
        "update " + dbName + ".COFFEES " +
        "set TOTAL = TOTAL + ? " +
        "where COF_NAME = ?";

    try {
        con.setAutoCommit(false);
        updateSales = con.prepareStatement(updateString);
        updateTotal = con.prepareStatement(updateStatement);

        for (Map.Entry<String, Integer> e : salesForWeek.entrySet()) {
            updateSales.setInt(1, e.getValue().intValue());
            updateSales.setString(2, e.getKey());
            updateSales.executeUpdate();
            updateTotal.setInt(1, e.getValue().intValue());
            updateTotal.setString(2, e.getKey());
            updateTotal.executeUpdate();
            con.commit();
        }
    } catch (SQLException e ) {
        JDBCTutorialUtilities.printSQLException(e);
        if (con != null) {
            try {
                System.err.print("Transaction is being rolled back");
                con.rollback();
            } catch(SQLException excep) {
                JDBCTutorialUtilities.printSQLException(excep);
            }
        }
```

```
    } finally {
        if (updateSales != null) {
            updateSales.close();
        }
        if (updateTotal != null) {
            updateTotal.close();
        }
        con.setAutoCommit(true);
    }
}
```

PreparedStatement étend de Statement et représente une instruction paramétrée. Les instances de PreparedStatement contiennent une **instruction SQL déjà compilée**

executeUpdate() : exécute l'instruction SQL du PreparedStatement
Il retourne le nombre de lignes mises à jour (int)

ResultSet : résultat d'une requête. L'accès se fait enregistrement par enregistrement

setAutoCommit : **true** afin d'activer le mode de validation automatique pour la connexion, **false** pour le désactiver. Il faut bien penser à remettre la valeur à true après la gestion de la transaction car c'est sa valeur par défaut.

Sinon, les méthodes qui n'utilisent pas de transaction (comme le DELETE par exemple) ne seraient pas commitée.

c) Clés auto-générées

https://docs.oracle.com/javadb/10.8.3.0/ref/crefjavstateautogen.html

```
Statement stmt = conn.createStatement();
stmt.execute(
    "INSERT INTO TABLE1 (C11) VALUES (1)",
    Statement.RETURN_GENERATED_KEYS);
ResultSet rs = stmt.getGeneratedKeys();
```

RETURN_GENERATED_KEYS : en cas d'insertion, cela retourne la ou les clés générées dans un Resultset
Pour savoir si une clé a été générée :

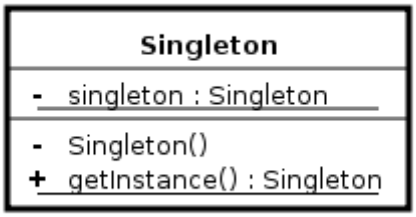
```
if (! rs.next())
        while (resultat.next()) {
                idGenere = resultat.getInt(1);
        }
```

1. Le Design Pattern Singleton

En génie logiciel, le singleton est un patron de conception dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet.

Dans notre cas, il servira à n'instancier qu'une seule connexion pour toute la durée de l'application.

Instancier une connexion à chaque appel à la base données saturerait celle-ci.



- Comme montré sur ce diagramme de classe, mettre le constructeur en « private » et créer une méthode « public » qui teste si l'objet est null avant d'appeler le constructeur et retourne l'objet connection