Secteur Tertiaire Informatique

Filière « Etude et développement »

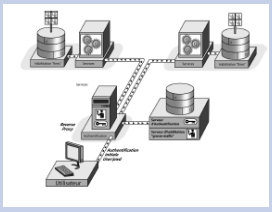
Accès BDD avec JDBC

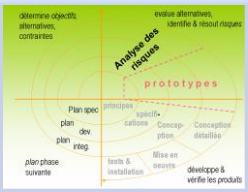
Persistance de données

Apprentissage

Mise en situation

Evaluation





1. PREAMBULE

Ce document va vous permettre de mettre en place une base de données afin d'assurer la persistance pour votre projet de **gestion de contacts**.

Liste des compétences abordées :

- Requêtage de base de données en utilisant JDBD
- Mise en place d'un « design pattern » **DAO** (Data Access Object)
- Mise en place d'un « design pattern » **singleton**

2. PRISE EN MAIN DE JDBC

Attention

Avant d'intégrer l'accès à la base de données dans votre projet gestion de contacts il vous est **fortement conseillé** de créer un **projet minimaliste** contenu dans un **nouveau dépôt.**

Ce projet vous permettra de prendre en main le requêtage en Java.

Il vous est donc proposé de créer un projet en ligne de commande afin de tester toutes les requêtes utiles pour votre application.

L'intégration des requêtes dans votre projet se fera dans un second temps.

2.1 QU'EST-CE-QUE JDBC ?

L'**API** JDBC (**Java Database Connectivity**) permet aux applications Java de se connecter à des bases de données relationnelles telles que MySQL, PostgreSQL, MS SQL Server ou encore Oracle.

Elle permet d'interroger et de mettre à jour des bases de données relationnelles, ainsi que d'appeler des fonctions et procédures stockée.

JDBC fait parti du JDK (Java Development Kit), elle est ainsi utilisable par toutes les applications Java.

L'API JDBC Java fait partie du SDK Java SE de base, ce qui rend JDBC disponible pour toutes les applications Java qui souhaitent l'utiliser. Voici un diagramme illustrant une application Java utilisant JDBC pour se connecter à une base de données relationnelle.

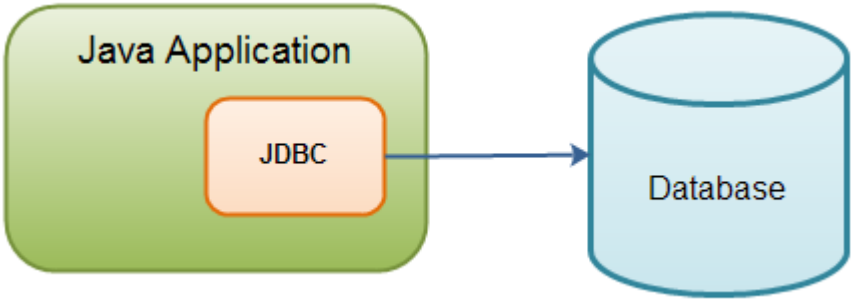


Figure 1 - (source jenkov.com)

Pour plus d'information sur le fonctionnement de JDBC vous

2.2 CREATION D'UN PROJET

Afin de mettre en place un nouveau projet comprenant les dépendances JDBC vous pouvez utiliser l'achétype Maven « **quickstart** » et ajouter la dépendance suivante au fichier Maven (attention de bien ajuster le numéro de version) :

```
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>42.6.0</version>
</dependency>
```

Cette dépendance ajoute un « driver » (aussi appelé « connector ») à votre application permettant de communiquer avec un SGBD (système de gestion de base de données) de type PostgreSQL.

Un fois votre projet créé et votre dépendance ajoutée vous pourrez utiliser JDBC à partir du code de votre projet.

2.3 MISE EN PLACE D'UNE BASE DE DONNEES DANS UN ENVIRONNEMENT DE TEST

Pour rappel, l'application que vous allez reprendre est basée basées sur JavaFX en utilisant une **architecture MVC**.

Voici comment s'organise un projet en MVC et cette bibliothèque :

- Fichiers FXML pour les **vues** ;
- Classes du package « **controllers** » pour les contrôleurs ;
- Classes du package « **models** » pour les **objets métier**.

Les **objets métiers sont les données** que nous allons chercher à persister en base de données. Jusqu'à présent ces objets étaient sauvegardés en utilisant une méthode de **sérialisation binaire**.

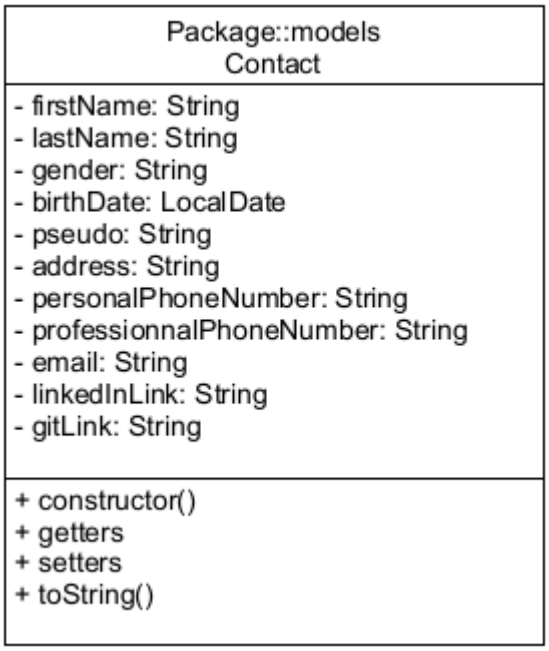
Vous allez remplacer cette **méthode de sérialisation** par une sauvegarde dans une base de données SQL.

Important

Il va vous falloir créer des tables de base de données à l'image de vos **objets métiers du modèle**.

Dans le cas du projet de gestion de projet il vous faut donc stocker en base de données les informations de la classe représentant les contacts.

Ci-dessous un exemple d'UML de cette classe :



Nous pouvons établir une correspondance MCD de cet UML, par exemple :

Important

Vous devrez adapter ces diagrammes en fonction de la classe métier de votre application de gestion de contact.

Les **classes du modèle Java devront être à l'image des classes de la base de données**, afin de simplifier leur développement vous pouvez transformer un modèle conceptuel de données

Merise en un diagramme de classe UML en suivant les règles suivantes : https://merise.developpez.com/faq/?page=MCD#Comment-traduire-un-MCD-en-un-diagramme-de-classe-UML

A faire

Déployez une base de données PostgreSQL dans un conteneur Docker.

Cette base de données devra contenir une table contenant toutes les colonnes permettant de stocker les informations de la classe « **Contact** »

2.4 ECRITURE DE REQUETES

2.4.1 Requêtes à tester

Attention

Dans un premier temps, testez vos requêtes à partir de fonctions contenues dans votre classes principales.

Avant de le faire il vous faut mettre en place la base de données.

Voici une liste des requêtes que vous allez devoir mettre en place :

1. **Récupération de toutes les informations de tous les contacts**
2. **Récupération d'un contact par identifiant**
3. **Suppression d'un contact en fonction de son identifiant**
4. **Suppression d'un contact en fonction de son nom et de son prénom**
5. **Ajout d'un nouveau contact**
6. **Modification de toutes les informations d'un contact**

Vous pourrez créer une fonction pour chacune des requêtes demandées. A vous de trouver les **signatures** des fonctions qui conviennent.

Par exemple, pour la requête 1, vous pourrez utiliser la déclaration de fonction suivante :

```
public List<Contact> getAllContacts() {
    /** Instructions **/
}
```

Attention

Il vous faudra instancier et manipuler des objets de la classe « Contact » au sein de ces fonctions.

2.4.2 Requêtage en Java

Il vous est fortement conseillé de lire l'excellente ressource de David Gayerie expliquant en détails le requêtage en utilisant JDBC : https://gayerie.dev/epsi-b3-orm/javaee_orm/jdbc.html?highlight=jdbc

Vous trouverez ci-dessous un ensemble de requêtes dont vous pourrez vous inspirer.

2.4.2.1 Exemple de requête « SELECT »

```
try {
            // chaîne de connexion à la base de données
            String url = "jdbc:postgresql://localhost:<port-bdd>/<nom-bdd>";
            // création d'un objet de la classe "Connection" en utilisant DriverManager
            Connection con = DriverManager.getConnection(url, "<login>", "<password>");

            // création d'un "Statement" (objet qui permet d'exéctuer une requête SQL)
            Statement stm = con.createStatement();

            // récupération de toutes les lignes de résultat (objet de la classe "ResultSet")
            ResultSet result = stm.executeQuery("SELECT * FROM <table>");

            // on passe en revue toutes les lignes
            while (result.next()) {
                // récupération des valeurs des colonnes
                int valueCol1 = result.getInt("<nom-colonne>");
                String valueCol2 = result.getString("<nom-colonne>");
                String valueCol3 = result.getString("<nom-colonne>");
                // affichage du résultat
                System.out.format("[%d] %s %s\n", valueCol1, valueCol2, valueCol3);
            }

            // fermeture des ressources
            stm.close();
            result.close();
            con.close();

} catch (Exception e) {
            System.err.println("Error");
            System.err.println(e.getMessage());
}
```

2.4.2.2 Exemple de requête préparée « INSERT »

```
        try {
```

```
            String url = "jdbc:postgresql://localhost:<port>/<nom-bdd>";
            Connection con = DriverManager.getConnection(url, "<user>", "<password");

            PreparedStatement stm = con.prepareStatement("INSERT INTO contact (first_name,
last_name, email, <...>) VALUES (?, ?, ?, <...>)");

            stm.setString(1, "Ada");
            stm.setString(2, "Lovelace");
            stm.setString(3, "ada@computing.co.uk");

            stm.execute();

            stm.close();
            con.close();

        } catch (Exception e) {
            System.out.println("Error");
            System.out.println(e.getMessage());
        }
```

A faire

Analyser le code et implémentez les différentes requêtes.

Vous trouverez un ensemble de ressource en lignes

3. MISE EN PLACE DU DESIGN PATTERN DAO

3.1 IMPLEMENTATION DANS L'APPLICATION CONSOLE

Maintenant que les requête ont été implémentée vous allez pouvoir réorganiser votre code afin de l'architecturer au mieux (et ne plus tout avoir dans votre classe principale).

Pour nous aider dans l'architecture logicielle nous pouvons adopter des « **design pattern** **»** (ou « patron de conception »).

Un « design pattern » peut être défini de la façon suivante (source : Rajib Mall, *Fundamentals of Software Engineering*, PHI Learning Pvt. Ltd.) :

« arrangement caractéristique de modules, **reconnu comme bonne pratique** en **réponse à un problème de conception** d'un logiciel. Il décrit une **solution standard**, utilisable dans la conception de différents logiciels »

L'adoption de « design patterns » permet de concevoir et développer un code bien organisé (et donc maintenable).

Dans notre cas le problème rencontré peut être résumé de la façon suivante : **dans une application avec approche orientée objet, nous souhaitons transformer les enregistrements d'une base de données relationnelle en objet métiers exploitable.**

Parmi les « design pattern » qui permet de trouver une solution à ce problème il existe le DAO (pour « Data Access Object »).

A faire

Implémentez le DAO dans votre application console en suivant le tutoriel suivant (vous pouvez vous arrêter avant le paragraphe portant sur le design pattern « factory ») :

https://cyrille-herby.developpez.com/tutoriels/java/mapper-sa-base-donnees-avec-pattern-dao/

3.2 IMPLEMENTATION DANS VOTRE APPLICATION EXISTANTE

Une fois le DAO fonctionnel dans votre application console, vous pourrez reprendre votre application JavaFX de gestion de contacts et ajouter votre nouveau code.

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Date de mise à jour : 18/09/2024