Introduction à JDBC

Sébastien NEDJAR et Fabien PESCI

1 Introduction

L'objectif de ce document est de vous présenter une méthode d'accès à un SGBD à travers le langage de programmation Java. Pour cela, nous allons dans un premier temps présenter l'API JDBC (Java DataBase Connectivity) [1]. C'est un ensemble de classes permettant d'exécuter des ordres SQL de manière générique. En effet, l'API JDBC est construit autour de pilotes (Driver) interchangeables. Un pilote est un module logiciel dédié à une source de données (un SGBD-R dans la plupart des cas). Pour utiliser comme source de données MySQL au lieu d'Oracle, il suffit de de remplacer le pilote Oracle par celui de MySQL. Ce changement de pilote peut se faire directement par paramétrage sans même avoir besoin changer une seule ligne de code ni même le recompiler [2].

2 Mise en place de l'environnement de travail

L'API JDBC fait partie de Java mais le pilote propre au SGBD-R Oracle n'y est pas. Avant de pouvoir se connecter à la base de données située sur *allegro*, il faudra donc ajouter à votre projet le fichier *jar* contenant le pilote adapté. Le prochain paragraphe sera consacré à l'installation de ce fichier à partir de l'IDE Eclipse.

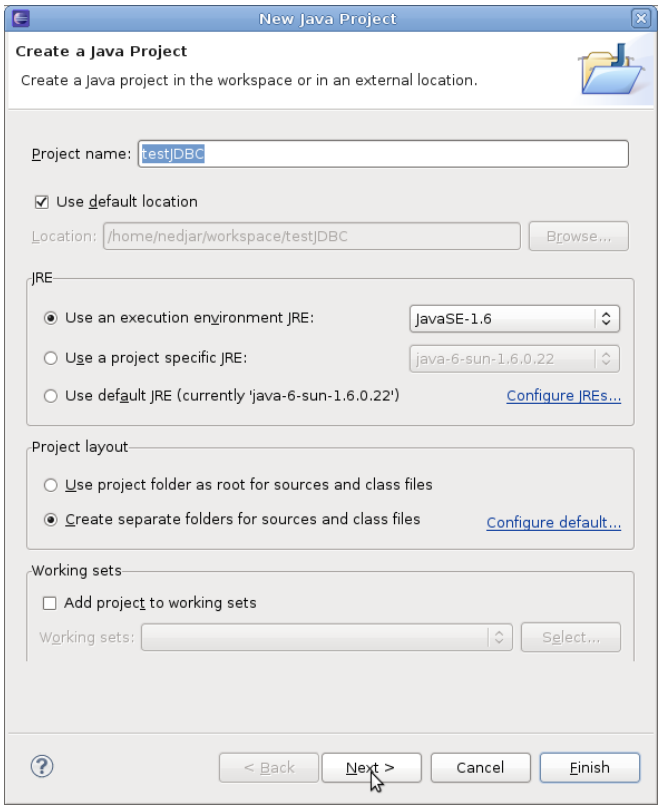
Avant de commencer, récupérez l'un des pilotes dans le repertoire local : `/commun/nedjar/ojdbc6.jar` ou `/commun/nedjar/mysql-connector-java-5.1.23-bin.jar`. Placer le fichier dans répertoire `~/net-home/tp/tpBDA/`. Puis à partir d'Eclipse, lancez l'assistant de création de nouveau projet Java (`File` → `New` → `Java Project`). Après avoir rempli les informations de ce premier écran, validez pour passer au suivant (*cf.* Figure 1a). Dans la nouvelle fenêtre, cliquez sur l'onglet *Libraries* (*cf.* Figure 1b) puis sur le bouton *Add External JARs* (*cf.* Figure 1c), sélectionnez le fichier `ojdbc14.jar` précédemment téléchargé (*cf.* Figure 1d), validez en cliquant sur *Finish*. Une fois ces étapes validées, vous avez un projet java capable d'utiliser JDBC pour interagir avec Oracle.

3 Traitement d'un ordre SQL avec JDBC

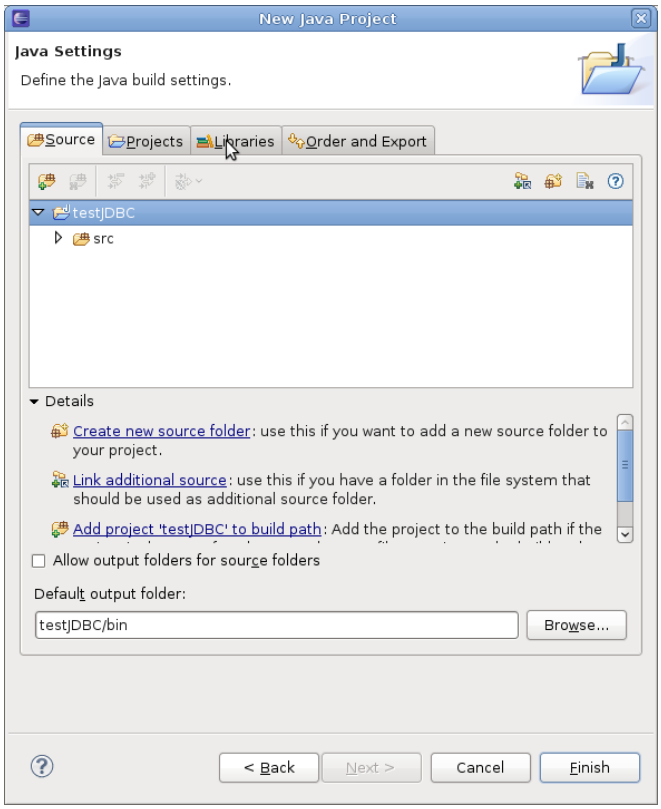
L'objectif général de cette partie est de mettre en évidence le schéma de fonctionnement classique de l'API Java d'interaction avec les bases de données relationnelles. Le principe de fonctionnement de cette API est proche de celle de PHP ou de C#. D'une manière générale, pour traiter un ordre SQL avec JDBC, il faudra suivre les étapes suivantes :

1. Connexion à la base de données.
2. Création d'une instruction SQL.
3. Exécution de la requête.
4. Traitement de l'ensemble des résultats.
5. Libération des ressources et fermeture de la connexion.

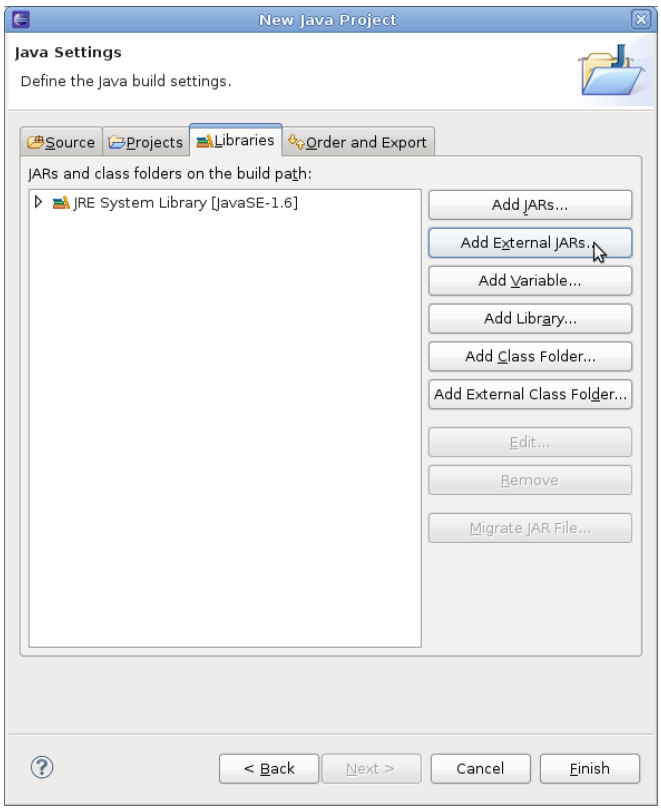
1. http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/
2. Il faut tout de même pondérer ces avantages car dans la pratique il existe de très nombreuses incompatibilités liées à des implémentations du langage SQL non respectueuses des standards.



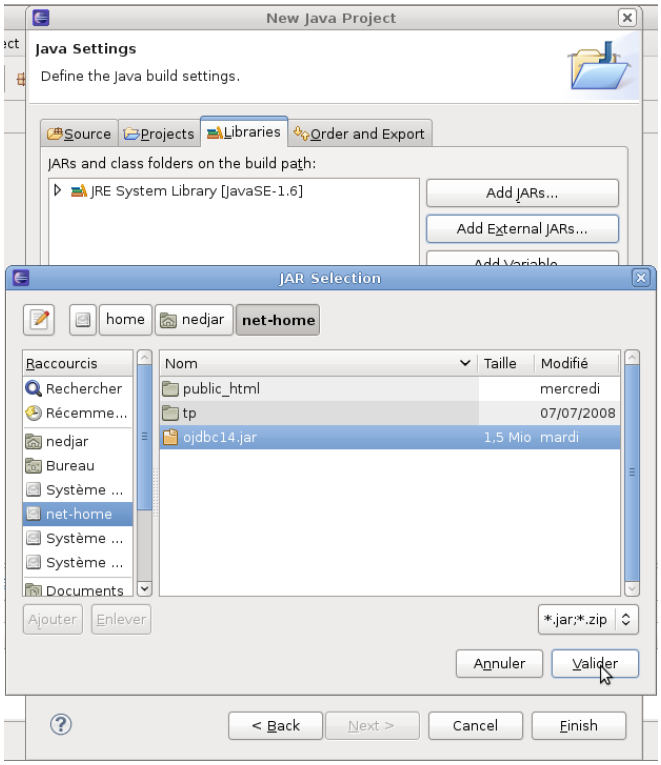
(a) Premier écran de l'assistant création de nouveau projet Java



(b) Deuxième écran de l'assistant création de nouveau projet Java



(c) Onglet *Libraries*



(d) Fenêtre de selection du fichier `~/net-home/tp/tpBDA/ojdbc6.jar`

FIGURE 1

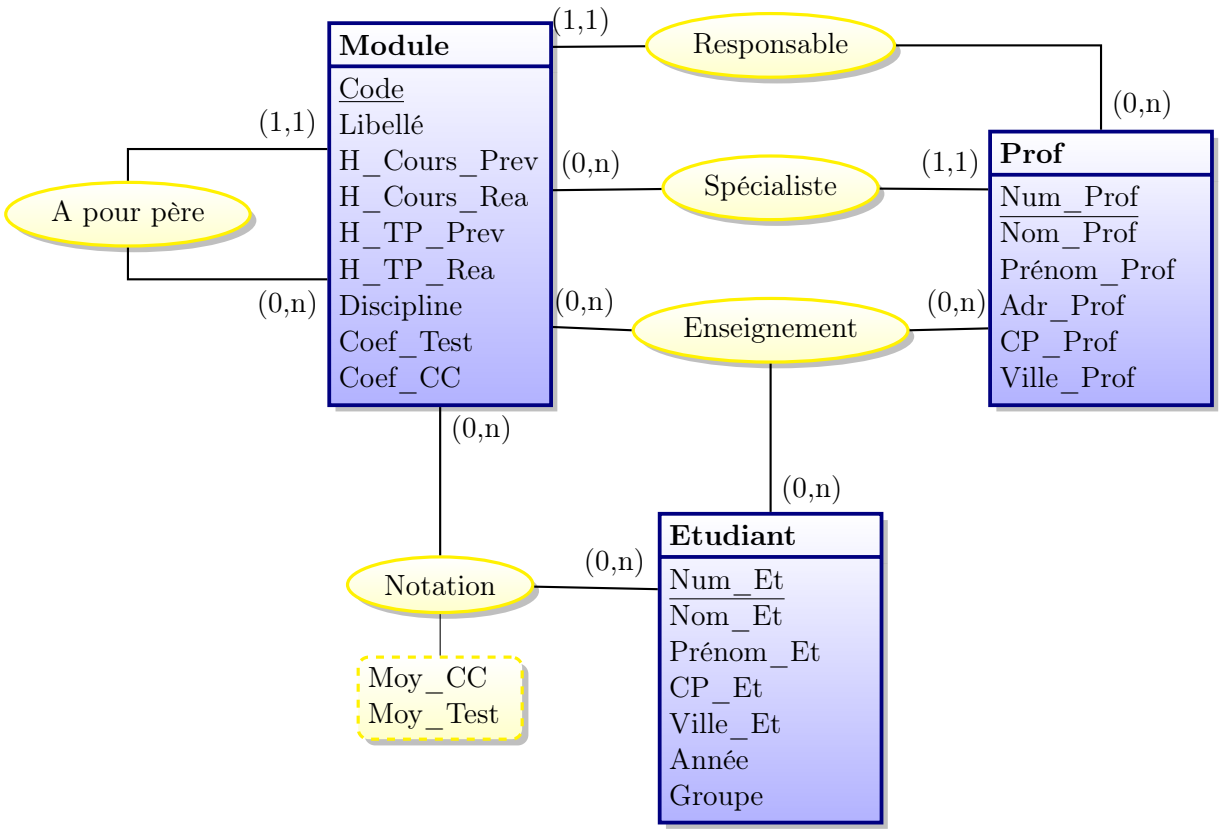


FIGURE 2 – Modèle conceptuel des données de la base « Gestion Pédagogique »

Étant donné que chacune de ses étapes est susceptible de rencontrer des erreurs, il faudra donc rajouter une étape supplémentaire de gestion des exceptions.

Pour illustrer ce propos, nous utiliserons la base de données « Gestion Pédagogique[3] » que vous avez utilisée lors de vos TP de PL/SQL en début d'année. Le modèle conceptuel des données est rappelé par la figure 2.

Le programme Java ci-dessous[4] va être utilisé pour illustrer le fonctionnement de chacune de ces étapes. L'objectif de ce programme est de récupérer la liste des numéros, noms et prénoms de tous les étudiants habitant à Aix-en-Provence pour l'afficher à l'écran. Cet exemple utilise un driver propre à Oracle. Il faudrait le remplacer par un autre pour utiliser un SGBD différent.

3. Script de régénération disponible à l'adresse suivante : `/commun/nedjar/gestion_peda_oracle.sql` ou `/commun/nedjar/gestion_peda_mysql.sql`

4. Code source disponible à l'adresse suivante : `/commun/nedjar/testJDBC.java`

```java
// Ne pas faire un copier/coller du pdf...

// Importer les classes jdbc
import java.sql.*;

public class testJDBC {
    // Chaine de connexion
    static final String CONNECT_URL = "jdbc:mysql://localhost:3306/
       maBD";
    static final String LOGIN = "monLogin";
    static final String PASSWORD = "monPaswd";
    // La requete de test
    static final String req = "SELECT NUM_ET, NOM_ET, PRENOM_ET " +
                              "FROM ETUDIANT " +
                              "WHERE VILLE_ET = 'AIX-EN-PROVENCE'";
    public static void main(String[] args) throws SQLException {
        // Objet materialisant la connexion a la base de donnees
        Connection conn = null;
        try {
            // Connexion a la base
            System.out.println("Connexion a " + connectString );
            conn = DriverManager.getConnection(CONNECT_URL,LOGIN,
               PASSWORD);
            System.out.println("Connecte\n");
            // Creation d'une instruction SQL
            Statement stmt = conn.createStatement();
            // Execution de la requete
            System.out.println("Execution de la requete : " + req );
            ResultSet rset = stmt.executeQuery(req);
            // Affichage du resultat
            while (rset.next()){
                System.out.print(rset.getInt("NUM_ET") + " ");
                System.out.print(rset.getString("NOM_ET") + " ");
                System.out.println(rset.getString("PRENOM_ET"));
            }
            // Fermeture de l'instruction (liberation des ressources)
            stmt.close();
            System.out.println("\nOk.\n");
        } catch (SQLException e) {
            e.printStackTrace();// Arggg!!!
            System.out.println(e.getMessage() + "\n");
        } finally {
            if (conn != null) {
                // Deconnexion de la base de donnees
                conn.close();
            }
        }
    }
}
```

Les différentes étapes détaillées ci-dessous mentionnent de nombreuses classes contenues dans les paquetages `java.sql.*` et `javax.sql.*`. Pour connaître les détails sur chacune de ces classes vous êtes invités à lire la Javadoc que vous trouverez à l'adresse suivante : http://download.

oracle.com/javase/6/docs/api/.

3.1 Connexion à la base de données

La première étape qui permet d'interagir avec une base de données est la connexion. Il faut initialiser un objet du type `Connection` grâce à la méthode `getConnection()` de la classe `DriverManager`.

3.2 Création d'une instruction Sql

Une fois la connexion établie, il faut créer un objet matérialisant l'ordre Sql à exécuter. Cet objet du type `Statement` est obtenu en appelant la méthode `createStatement()` de notre connexion. Il existe trois types d'ordre :

1. Les `Statement` : Ils permettent d'exécuter n'importe quelle requête sans paramètre. La requête est interprétée par le Sgbd au moment de son exécution. Ce type d'ordre est à utiliser principalement pour les requêtes à usage unique.
2. Les `PreparedStatement` : Ils permettent de précompiler un ordre avant son exécution. Ils sont particulièrement importants pour les ordres destinés à être exécutés plusieurs fois comme par exemple les requêtes paramétrées.
3. Les `CallableStatement` : Ils sont destinés à l'appel des procédures stockées.

3.3 Exécution de la requête

Afin d'exécuter une requête, il suffit de faire appel à l'une des méthodes `executeXXXX()` de l'objet `Statement` que l'on vient de créer. Dans l'exemple ci-dessus on utilise la méthode `executeQuery()` en lui passant en paramètre une chaîne de caractères (`string`) contenant la requête (comme Execute Imediate de Pl/Sql). Cette méthode retourne un objet du type `ResultSet` contenant l'ensemble des résultats de la requête. Il faut noter que si l'ordre Sql est une mise à jour des données (Insert, Update, Delete), il faudra alors l'exécuter avec la méthode `executeUpdate()` qui retourne un entier correspondant au nombre de lignes impactées par la mise à jour.

3.4 Traitement de l'ensemble des résultats

La manipulation du résultat d'une requête se fait à travers un objet du type `ResultSet`. Le résultat se manipule, comme avec les curseurs de Pl/Sql, ligne après ligne. Ainsi, l'objet `ResultSet` maintient un pointeur vers la ligne courante. La manipulation de ce pointeur se fait avec la méthode `next()` qui permet d'avancer le pointeur sur la ligne suivante. Lors de la création du `ResultSet` ce pointeur est positionné sur une ligne spéciale appelée le *gap*. Cette ligne est située une ligne avant la première ligne du résultat. De ce fait, la première ligne n'est pointée qu'après le premier appel à `next()`. Lorsque le pointeur est positionné après la dernière ligne, `next()` retourne la valeur `false`. Pour parcourir linéairement l'intégralité d'un `ResultSet`, on utilise donc une boucle `while` avec `next()` comme prédicat de continuation. Le corps de la boucle est dédié à la manipulation de la ligne (tuple) couramment pointée.

Afin de récupérer les valeurs des attributs du tuple courant, on utilise l'une des différentes méthodes `getXXXX()` (où `XXXX` désigne le type de l'attribut que l'on souhaite récupérer). Par exemple, pour récupérer un entier on utilise `getInt()` et pour récupérer un booléen on utilise `getBoolean()`. Le paramètre passé à cet accesseur permet de choisir l'attribut à récupérer. Il existe deux façons pour désigner un attribut. La première (celle de l'exemple) consiste à utiliser une `string` contenant le nom de la colonne souhaitée. La seconde quant à elle, passe en paramètre un entier (`int`) contenant la position (dans le Select) de l'attribut à récupérer. Attention, contrairement à l'habitude en programmation, les attributs sont numérotés à partir de 1 (et non

de 0). Par exemple, si l'on souhaite récupérer la valeur de l'attribut `NUM_ET` (le premier dans notre SELECT), il faudra faire : `rset.getInt(1)`.

3.5 Libération des ressources et fermeture de la connexion

Tant que l'on utilise un `Statement` ou une `Connection`, le système nous alloue un certain nombre de ressources. Maintenir ces ressources disponibles a un coût non négligeable. Ainsi, comme toujours en informatique, pour éviter le gaspillage (et donc un ralentissement inutile) il faut libérer les ressources dès qu'elles ne sont plus nécessaires. Pour ce faire, il suffit d'appeler la méthode `close()` des objets `Statement` et `Connection`.

3.6 Gestion des exceptions

La grande majorité des classes de JDBC sont susceptibles de lever des exceptions lorsqu'elles rencontrent des erreurs. C'est pour cela qu'il faut toujours encadrer le code JDBC par un bloc `try/catch`. Les exceptions levées sont toutes des classes filles de `SQLException`.

Question 1 : Mettre en place un projet TestJDBC pour tester la classe donnée en exemple. N'oubliez pas de configurer votre base de données pour qu'elle contienne les données nécéssaires.