JDBC

Maurice. D. Faye

JDBC

Java DataBase Connectivity

- JDBC est une API (ensemble de classes et d'interfaces) permettant de développer des applications Java capables de se connecter à des serveurs de bases de données. Le serveur peut être distant ou sur la même machine que l'application.

- L'API se charge de trois étapes indispensables à la connexion à une base de données :
  - la création d'une connexion à la base,
  - l'envoi d'instructions SQL,
  - l'exploitation des résultats provenant de la base.

Les objets et méthodes relatifs aux bases de données sont présents dans le package *java.sql*.

JDBC

- L’API est indépendante des SGDR. Donc le même programme Java peut accéder à une base de données MySQL, oracle, Derby,etc.
- Pour un SGBD donné, l'ensemble des classes qui implémentent les interfaces de l'API (java.sql.Driver) est appelé un pilote (driver) .
- Les protocoles d’accès au SGBD sont propriétaires ==> les pilotes diffèrent en fonction des SGBD ==> il faut donc en fonction des SGBD , charger les pilotes adéquats ⇒ pilote de MySQL pour notre cours
- Un programme peut avoir plusieurs pilotes s'il veut attaquer divers types de SGBD

JDBC

- Créez la BD suivante sous MySQL. **num_ins** et **code_mod** sont des entiers dans cet exemple, les autres champs des chaînes.
- Insérez quelques données dans la base.

  Etudiants (**num_ins**, nom, prenom)

  Modules (**code_mod**, intitule)

  Inscriptions (**num_ins**, **code_mod**)

JDBC

- Ajouter les pilotes (pour les deux BD) au projet [plusieurs manières]
  - Clic droit sur la partie bibliothèques [library/librairies en anglais] du projet ⇒ ajouter une bibliothèque :
    - Si Java DB Driver et Pilotes JDBC mysql ne sont pas dans la liste ⇒ bouton importer et ajouter ses pilotes
  - Si on a déjà téléchargés les fichiers du pilote, on peut les ajouter en cliquant [droit] sur la partie bibliothèques/librairies du projet ⇒ ajouter des fichiers jar ⇒ et ajouter les fichiers
  - Propriétés du projet ⇒ librairies ⇒ onglet compiler ⇒ boutons ajouter bibliothèque et ajouter fichiers jar

JDBC

- Pour se connecter à une base de données, on choisit et on charge le pilote de base de données à laquelle on désire se connecter en utilisant le gestionnaire de pilotes (classe *DriverManager*). Nous allons utiliser le pilote JDBC/MySQL:

Pilote ORACLE : oracle.JDBC.driver.OracleDriver

Pilote JDBC/ODBC : sun.jdbc.odbc.JdbcOdbcDriver

Pilote mySQL : com.mysql.jdbc.Driver

Pilote Derby : org.apache.derby.jdbc.ClientDriver

JDBC

Java DataBase Connectivity

- Choisir le pilote

// pilote MySQL récent :

```
String driver="com.mysql.jdbc.Driver";
```

// pilote BD Java DB (derby) si on veut se connecter à une BD Derby

```
//String driver="org.apache.derby.jdbc.ClientDriver";
```

- Charger le pilote
  - Se fait avec la méthode Class.forName(String Pilote) throws ClassNotFoundException

```
Class.forName(driver);
```

JDBC

Java DataBase Connectivity

- Pour se connecter à une base, il faut fournir une URL qui dit comment se connecter (le protocole) ; où se trouve la base (adresse machine et éventuellement le port), le nom de la BD et éventuellement le nom d'utilisateur et le mot de passe:

  Format URL de la BD : protocole//host:port/nomBD

- //url BD mysql, port par defaut 3306

  String url="jdbc:mysql://localhost:3306/testdb";

  ou bien //"jdbc:mysql://localhost/testdb" ⇒ si num port non précisé , port par défaut est utilisé

- //url BD DERBY, port par defaut 1527

  String url="jdbc:derby://localhost:1527/testbd_derby";

JDBC

- Autres :

  URL ORACLE : jdbc:oracle:thin:host:port:idbase

  URL ODBC : jdbc:odbc:IDDSN

Etc..

JDBC

- La connexion à une BD se fait Se connecter à la méthode

  Connection DriverManager.getConnection("URL","user","pass") throws SQLException ⇒ retourne un objet de classe Connection

String url="jdbc:mysql://localhost:3306/testdb"; //"jdbc:mysql://localhost/maBD"

String login="testuser"; //

String pswd="password";

Connection co = DriverManager.getConnection(url,login,pswd);

JDBC

- Après l'établissement de la connexion avec la base de données, on peut :
  - exécuter des requêtes SQL pour récupérer des données ou effectuer une mise à jour.
  - Il est aussi possible de demander des informations sur la base telles les noms des tables ou leurs structures (méta-données).

JDBC

- Dans les deux cas (requête ou méta-données), il faut d'abord créer un objet de la classe *Statement*, obtenu à partir de l'objet de la classe Connection.

Statement stmt = co.createStatement(); // lance SQLException

- Ensuite on peut exécuter la requête de sélection avec la méthode executeQuery(String requete) throws SQLException

  de l'objet de la classe Statement

- cette méthode retourne un objet de type ***ResultSet***

- ***Pour les requêtes de MAJ (INSERT, UPDATE, DELETE), il faut utiliser la méthode int executeUpdate(String sql) de l'objet Statement.***

JDBC

- Exemple :

```
// requête de sélection

String reqSelect="select * from Etudiants";

Statement st = co.createStatement(); //co= objet de type Connection

ResultSet rs = (ResultSet) st.executeQuery(reqSelect);

 // pour requête de MAJ (insert/delete/update) c'est la méthode executeUpdate(String sql)

 String reqSqlMaj="insert into Etudiants(num_ins,nom,prenom)"+ "values (30,'Fatou','sarr')";

 int reussi=st.executeUpdate(reqSqlMaj) ;
```

JDBC

- Ensuite, on peut se servir des objets suivants pour interroger la base de données : ***ResultSet*** , **ResultSetMetaData**

ResultSet :

- il contient les informations sur une table (noms des colonnes par exemple) ou le résultat d'une requête.
- L'accès aux données se fait colonne par colonne, mais il est possible d'accéder indépendamment à chaque colonne par son nom.
- C'est l'objet le plus important, la plupart des objets et requêtes retournent les données sous forme d'un objet ResultSet.

JDBC

- Les principales méthodes de la classe ResultSet sont:
  - next() : permet d'obtenir chacune des lignes (un enregistrement d'une table) de l'objet et retourne false lorsqu'il ne reste plus aucune ligne. Accède à la prochaine ligne

    s'il y en a
  - String getString(String columName) : interroger un champ String par son nom
  - int getInt(...) : interroger un champ Integer par son nom ou index

    Exple:

    While (rs.next()) { System.out.println(rs.getString("nom")+" "+rs.getInt("age")); }
  - close() : ferme l'objet
  - getMetaData(): retourne les métadonnées de l'objet (un objet ResultSetMetaData).
  - ….

JDBC

- Les principales méthodes de la classe ResultSet sont:
  - String getString(int columnIndex) : interroger un champ String par son index
  - Exemple avec l'exécution de la requete "select * From Etudiants"

```
while (rs.next()) {
String numIns = rs.getString(1); // à traduire en int avec parseInt()
String nom = rs.getString(2);
String prenom = rs.getString(3);
}
```

JDBC

- La classe ResultSetMetaData : il contient les informations concernant le nom et le type des colonnes d'une table. Pour exploiter un objet ResultSet, il est parfois nécessaire de récupérer le nombre de colonnes qu'il contient :
- Méthodes :
  - ResultSetMetaData rsmd = rs.getMetaData();
  - int nbre_colonnes = rsmd.getColumnCount() ;
  - String nom_col = rsmd.getColumnName() ;
  - DataBaseMetaData : il contient les informations sur la base de données : noms des tables, index, etc.

JDBC : exemple

Testez le chargement des pilotes et la connexion à la base (TestJdbc.java)

```
import java.sql.*;
import javax.swing.JOptionPane;
public class TestJDBC {
        public static Connection initConnection () {
        Connection con=null;
       //pilote + récent :
         String driver="com.mysql.jdbc.Driver";
String url="jdbc:mysql://localhost:3306/testdb";  /*port par defaut
pour MySql ⇒ "jdbc:mysql://localhost/maBD"  oracle ⇒ 1521, ….*/
String login="testuser";              // par defaut  "root"
String pswd="password";        // par défaut ""
```

JDBC

```
-----suite----
try
  {
            //2- charge pilote
            Class.forName(driver);
             System.out.println(" Pilote chargé= OK: " );
  }
catch(Exception ex)  /*ClassNotFoundException exception généré par Class.forName n'est pas du type SQLException car non lie a la DB mais au pilotes*/
   {
          System.out.println(" Problème de chargement de driver: " + ex.getMessage());
           return null;
   }
```

JDBC

```
----suite-----
try
   {
       con = DriverManager.getConnection(url,login ,pswd );
        //con = DriverManager.getConnection(url );
       System.out.println("Connexion établie");
              return con;
   }
catch(SQLException ex)   // exception liée à la BD
   {
       System.out.println(" Problème de connexion à la BD: " + ex.getMessage());
              return null;
   }
}
```

JDBC

----suite-----

```
 public static void main(String[] args) {
        Connection MaConnection= null;
                //MaConnection=Connection initConnection (driver,url,login,passwd);
          MaConnection=initConnection();
           if(MaConnection==null) return;          //si echec connexion
           System.out.println("connexion réussie");
}
}
```

JDBC : tester une requête de sélection (TestJdbc1.java)

----Ajoutez au main (exemple précédent)

```
//traitement à faire si la connexion est ok: envoyer requetes+ traitement, etc..
        try{
        System.out.println(" Exemple requete selection ");
        String reqSelect="select * from Etudiants";
        Statement st = MaConnection.createStatement();
        ResultSet rs = (ResultSet)st.executeQuery(reqSelect);

        //traiter le ResultSet  rs.last()==true si le rs est vide
        while(rs.next()){
            //acces au contenu par nom des colones du resultat en fonction de leur types
              //System.out.println(rs.getString("NomcolonneString")+ " "+rs.getInt("NomcolonEntier"));
         //JOptionPane.showMessageDialog(null, "req selection OK: reste a traiter le resultat");
         System.out.println(rs.getInt("num_ins")+ " "+rs.getString("nom")+ " "+ rs.getString("prenom"));
        }
        st.close();
        MaConnection.close();   // si on a plus besoin de connection
        }
        catch(SQLException ex){
            System.out.println(" erreur requete "+ex.getMessage());
        }
```

JDBC : tester une requête de MAJ (TestJdbc2.java)

----Ajoutez au main

```
System.out.println(" Exemple requete MAJ: insert, update,delete ");
 MaConnection=initConnection();
 if(MaConnection==null) return;//MaConnection=initConnection();
 try{
 String reqMaj="insert into Etudiants(num_ins,nom,prenom)"
        + "values (30,'big','fap')";    // insert/delete/update
 Statement st = MaConnection.createStatement();
 st.executeUpdate(reqMaj);
 JOptionPane.showMessageDialog(null, "insertion OK");
 st.close();
 MaConnection.close();   // si on a plus besoin de connection
 }
 catch(SQLException ex){
    System.out.println(" erreur requete "+ex.getMessage());
    JOptionPane.showMessageDialog(null, "Erreur Req MAJ");
 }
```

JDBC

- Exercice :
- Afficher la liste des étudiants, chacun sur une ligne
- Afficher pour un étudiant l'ensemble des modules où il est inscrit
- Utiliser une interface graphique pour insérer un étudiant
- Utiliser une interface graphique, disposant d'une zone de texte pour afficher l’ensemble des étudiants.