RAPPELS : connexion à JDBC (sans fichier de propriétés) et principe des DAO

IUT de LYON, V. Deslandres

Notions de base : le package java.sql.*

Tous les objets et les méthodes relatifs aux bases de données sont présents dans le package *java.sql*, il est donc indispensable d'importer *java.sql.** dans tout programme se servant de la technologie JDBC.

Le package *java.sql* contient les éléments suivants :

Classes	Interfaces	Exceptions
Date DriverManager DriverPropertyInfo Time Timestamp Types	Array Blob CallableStatement Clob Connection DatabaseMetaData Driver PreparedStatement Ref ResultSet ResultSetMetaData SQLData SQLInput SQLOutput Statement Struct	BatchUpdateException DataTruncation SQLException SQLWarning

Connexion à la base de données - Pour se connecter, il faut charger le pilote de la base de données à laquelle on désire se connecter grâce à un appel au `DriverManager` (gestionnaire de pilotes) :

```
Class.forName("nom.de.la.classe");
```

Cette instruction charge le pilote et crée une instance de cette classe. Ainsi pour se connecter à une base de données déclarée dans l'administrateur ODBC (pour ACCESS par exemple), il faut charger le pilote JDBC-ODBC (appelé *pont JDBC-ODBC*) :

```
Class.forname("sun.jdbc.odbc.JdbcOdbcDriver");
```

pour MySQL c'est le pilote **"com.mysql.jdbc.Driver".** Certains compilateurs refusant cette notation, il faut parfois appeler le driver de la façon suivante :

```
Class.forname("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

Pour se connecter à une base de données particulière, il s'agit ensuite de créer une instance de la classe *Connection* grâce à la méthode *getConnection* de *DriverManager* en indiquant la base de données à charger à l'aide de son URL :

```
String url = "jdbc:mysql:base_de_donnees";
Connection con = DriverManager.getConnection(url);
```

La syntaxe de l'URL peut varier légèrement selon le type de la base de données. Il s'agit généralement d'une adresse de la forme: `jdbc:sousprotocole:nom`

En pratique

(1) Récupérer le driver de votre BD : driver **MySQL Connector/J.jar** ou **ojdbc14.jar** pour ORACLE et l'intégrer à la bibliothèque de votre projet

http://dev.mysql.com/downloads/connector/j/

(2) Créer un package dédié à l'accès aux BD : Donnees par ex.

Dans ce package, créer la classe de connexion en singleton. Ci après on présente l'accès à une BD MySQL du serveur de l'IUT (c'est identique pour Oracle). On verra plus tard comment procéder de façon plus propre avec une **fabrique de connexion**, permettant d'atteindre différents SGBD.

On rappelle qu'un **singleton** est une classe qui n'a qu'une seule instance (le constructeur est privé, une méthode publique getInstance() retourne l'instance déjà créée, ou appelle le constructeur si l'instance n'existe pas déjà).

NOTA : Le serveur IUT où se trouve mySQL est iutdoua-webetu.univ-lyon1.fr.

```
package Donnees;

import java.sql.*;

public class MyConnection {

   private Connection conn;                          // l'instance en champ privé
   /**
    * Méthode qui va retourner l'instance de connexion ou la créer si elle n'existe pas
    * @return Connection
    */
   public Connection getConnection() throws SQLException {

      //This condition will check if the connection is not already open
      if (conn == null) {
         // on cree la connexion
         try {
            Class.forName("com.mysql.jdbc.Driver");
            // ici connexion à MySQL de l'IUT :
                  String userid = "monLogin";
                  String password = "monPwd";
                  String URL = "jdbc:mysql://iutdoua-webetu.univ-lyon1.fr/maBD";
                  conn = DriverManager.getConnection(URL, userid, password);
```

```
                System.out.println("==> connexion à MySQL effectuee !");

            } catch (ClassNotFoundException e) {
                System.out.println("**** La connexion a la BD a echoue....");
                e.printStackTrace();
            }
        }
    // si la connexion existe, on la renvoie
    return conn;

     }
}
```

REMARQUE : avec Java 7 maintenant ce n'est plus la peine d'appeler `Class.forName ("com.mysql.jdbc.Driver")`. L'Automatic Resource Management (ARM) ajouté à JDBC 4.1, qui est automatiquement associé à Java 7, s'en charge.

AMELIORATION1 : utiliser un fichier de propriétés Mysql.properties (et Oracle.properties) plutôt que d'écrire les propriétés de connexion en dur avec ForName et DriverManager.

AMELIORATION2 : utiliser boite de dialogue et la classe java.net.PasswordAuthentication pour saisir les login et mot de passe plutôt que de les écrire de manière visible dans le code.

Ces améliorations sont présentées en fin du document.

(3) On va créer une classe DAO (Data Access Object) par type d'objet persistant. Pour cela on créer une classe abstraite DAO.java, qui repose sur un type générique, et qui sera utilisée pour toutes les classes Métier.

Classe DAO.java

```
package Donnees;

import java.util.ArrayList;

public abstract class DAO<Type> {
// ici Type signifie type générique, on aurait pu mettre E pour élément
   protected ArrayList<Type> liste;

   public DAO(ArrayList<Type> liste) {
      this.liste = liste;
   }

   public ArrayList<Type> getListe() {
      return liste;
   }

   public void setListe(ArrayList<Type> liste) {
      this.liste = liste;
   }

   public abstract int ajouter(Type t);
   public abstract int modifier(int index, Type t);
   public abstract int charger();                    // depuis la BD
```

```
        public abstract int supprimer(int index);
    }
```

Dans les classes d'accès aux données métier, on n'implémentera que les méthodes nécessaires de DAO. On donne ici l'illustration du DAO pour une classe Film :

package Donnees;

```
import Metier.Film;
import java.sql.*;
import java.util.ArrayList;
```

public class DAOFilm extends DAO<Film> {

```
    public DAOFilm(ArrayList<Film> liste) {
        super(liste);
    }

    @Override
    public int charger()
    {
        try {
            Connection connect = MyConnection.getConnection();
            Statement stmt = connect.createStatement();
            ResultSet rs = stmt.executeQuery("select * from Film");
                while(rs.next()){
                    //System.out.println(rs.getString("nom") + " " + rs.getInt("ref") + "sorti le " +
rs.getString("date"));
                    liste.add(new Film(rs.getString("nom"), rs.getInt("ref"), rs.getString("date")));
                }
                rs.close();
                stmt.close();
                return 0;
        } catch (Exception e) {
            System.out.println("Erreur au chargement des fims : " + e.getMessage());
            return -1;
        }
    }

    @Override
    public int supprimer(int index) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public int ajouter(Film t) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public int modifier(int index, Film t) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Par ailleurs on a la classe Métier Film :

```
package Metier;

import java.util.*;

public class Film {
   private int ref;
   private static int derniereRef = 0;
   private String titre;
          ….
   private String date;  // de sortie

    public Film(String titre, String date) {
       this.titre = nom;
       this.date = date;
       this.ref = derniereRef++;
    }

    public static int getDerniereRef() {
       return derniereRef;
    }

    public static void setDerniereRef(int derniereRef) {
       Film.dernierRef = dernierRef;
    }

    public String getDate() {
       return date;
    }

    public void setDate(String date) {
       this.date = date;
    }

    public String getTitre() {
       return titre;
    }

    public void setTitre(String titre) {
       this.titre = titre;
    }

    public int getRef() {
       return ref;
    }

    public void setRef(int ref) {
       this.ref = ref;
    }

    @Override
    public String toString() {
       return titre + " sorti le " + date;
```

```
        }
    }
```

AMELIORATION de l'ACCES à JDBC

Ici on utilise un fichier de propriétés Mysql.properties (et Oracle.properties) plutôt que d'écrire les propriétés de connexion en dur avec ForName et DriverManager.

Avantage : on peut disposer de plusieurs SGBD et leurs fichiers de propriétés, et varier celui qui sera utilisé dans le code facilement, selon les besoins. C'est plus propre, les éléments de connexion sont clairement dissociés du code qui utilise la connexion.

AMELIORATION2 : utiliser boite de dialogue et la classe java.net.PasswordAuthentication pour saisir les login et mot de passe plutôt que de les écrire de manière visible dans le code.

```
package accesBD;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import oracle.jdbc.pool.OracleDataSource;

/**
 *
 * Cree la connexion à Oracle par une méthode de classe (static)
 *  avec login / pwd saisies, et infos de connexion dans un fichier de proprietes
 *
 **/

public class ConnexionOracle {

    // constructeur vide
    public ConnexionOracle() {}

   // connexion a la BD ORACLE via un DATA SOURCE
    public static Connection creerConnexion( PasswordAuthentication identification ) {

      Properties props = new Properties();
      FileInputStream fichier = null;

      try {
        //fichier = new FileInputStream("src/accesBD/connexionOracle.properties");
        fichier = new FileInputStream("src/accesBD/connexionMysql.properties");
        props.load(fichier);

       }
        catch (FileNotFoundException e) {

         System.out.println("**** Fichier decrivant les proprietes d'acces a la BD non trouve !"
```

```
            +e.getMessage());
    } catch (IOException ex) {
        Logger.getLogger(ConnexionOracle.class.getName()).log(Level.SEVERE, null, ex);
     }
    finally {
        try {
            fichier.close();
        } catch (IOException ex) {
            Logger.getLogger(ConnexionOracle.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    try {
        OracleDataSource ods = new OracleDataSource();

        ods.setDriverType(props.getProperty("pilote"));
        ods.setPortNumber(new Integer(props.getProperty("port")).intValue());
        ods.setServiceName(props.getProperty("service"));
        // saisie des login et mot de passe via une fenetre de dialogue
        ods.setUser( identification.getUserName() );
        // conversion nécessaire d'un tabl de char en String :
        ods.setPassword( new String(identification.getPassword()) );
        ods.setServerName(props.getProperty("serveur"));

        return (ods.getConnection());


    } catch (Exception e) {

       System.err.println("Erreur lors de la connexion Oracle...");
       return null;
    }
  }

 }  // fin de ConnexionOracle
```

FICHIER de propriétés décrivant la connexion à une BD Oracle de l'IUT

```
# Sample ResourceBundle properties file
# pour la connexion a Oracle serveur IUT de LYON
port=1521
service=orcl
serveur=iutdoua-oracle.univ-lyon1.fr
pilote=thin
```

Classe singleton regroupant les requêtes ORACLE

package accesBD;

```
import java.sql.PreparedStatement;
import java.net.PasswordAuthentication;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
import metier.LigneTable;


public class RequetesOracle {

    private static RequetesOracle instance = null;        // instance unique d'accès, en static
    private Connection connexionOracle;
    private Statement stmt;
    private ResultSet rset;

    // constructeur privé du singleton :
    private RequetesOracle() throws IOException {
        connexionOracle = ConnexionOracle.creerConnexion( new PasswordAuthentication() );
        System.out.println("==> connection ORACLE etablie...");
    }
    // methode qui créer l'unique instance d'accès à la BD

    public static RequetesOracle getInstance() throws IOException {

        if (instance == null) {
            instance = new RequetesOracle();
        }
        return instance;
    }

    public void close() throws Exception {
        connexionOracle.close();
        System.out.println("==> Deconnection ORACLE OK");
    }

    // LISTE REQUETES SQL ======================================

    // récupère tous les départements présents dans la table des dpts
    // fonction appelée pour remplir les listes
    // on utilise un Vector<> (deprecated) car le seul constructeur de JList et ComboBox
    // commun repose sur cette seule collection !

    public Vector<String> listDept() {

        String query = "SELECT dname from scott.dept";
        Vector<String> listeDept = new Vector<String>();

        try {
            stmt = connexionOracle.createStatement();
            rset = stmt.executeQuery(query);
            while ( rset.next() ) {
                listeDept.add( rset.getString(1) );
            }
            stmt.close();
            rset.close();

        } catch (SQLException e) {
            System.out.println(e.getMessage());
```

```
        } finally {
            try {
                if (rset != null) {
                    rset.close();
                }
            } catch (SQLException ex) {
                // Ignorer
            }
        }
        return listeDept;
    }
//…. Autres requêtes utiles pour l'application
} // fin de classe RequetesOracle
```

Classe de saisie pour le login et mot de passe :

```
package vue;

import java.net.PasswordAuthentication;

public class JDAuthentificationBD extends javax.swing.JDialog {
    /**
     * Creates new form JDAuthentificationBD
     */
    public JDAuthentificationBD(java.awt.Frame parent) {
        super(parent, "Identification", true);
        initComponents();
    }

    public PasswordAuthentication recuperer() {
        setVisible(true);
        return   new PasswordAuthentication( jTextUtilisateur.getText(), jPasswordMdp.getPass
word() );
    }
    private void Quitter(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }

    private void jButtonOKActionPerformed(java.awt.event.ActionEvent evt) {
        setVisible(false);
    }

    private javax.swing.JButton jButtonOK;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JPasswordField jPasswordMdp;
    private javax.swing.JTextField jTextUtilisateur;

  }
```