RAPPELS : connexion à JDBC et principe des DAO

IUT de LYON, CPOA, semestre 3 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 Pour MariaDB (new MySQL free) c'est le pilote « MariaDB Connector/J » qui est un pilote JDBC Type 4.

Cette instruction charge le pilote et crée une instance de cette classe.

Ancien mode de connexion, deprecated - Pour se connecter à une base de données particulière avec **`DriverManager`**, il faut 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:mariadb:base_de_donnees";
Connection con = java.sql.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`

Par exemple pour un accès à une BD locale MariaDB, on aura pour `getConnection()`:

```
Connection connection =
DriverManager.getConnection("jdbc:mariadb://localhost:3306/DB?user=
root&password=myPassword");
```

En pratique

(1) Récupérer le driver de votre BD : driver **MariaDB Connector/J .jar** ou **ojdbc8.jar** pour ORACLE et l'intégrer à la bibliothèque de votre projet (chercher `Librairies` soit dans le projet, soit dans menu `Run – Set Project Configuration – Customize…`), et ajouter le .jar.

https://downloads.mariadb.org/connector-java/ (chercher JDBC + nom de la BD)

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

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

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 *MariaDB* est iutdoua-web.univ-lyon1.fr.

Code de connexion à la BD avec DriverManager

```
package persistance;

import java.sql.*;

public class ConnexionBD {

   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 {

      //On teste si la connexion n'est pas deja ouverte
      if (conn == null) {
         // on cree la connexion
         try {
            // Connexion à MariaDB de l'IUT (OK juin 2018) :
              DriverManager.getConnection ("jdbc:mysql://iutdoua-web.univ-lyon1.fr:3306/pxxxxx",  // ici nom de
votre base
                        "pxxxxx", "yourPwd"); // ici vos login et pwd
```

```
            System.out.println("==> connexion à MariaDB effectuee !");
// (c'est bien la BD MariaDB mais les processus sont encore appelés mySQL dans la version APACHE de l'IUT...)

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

   }
}
```

REMARQUE : avec Java7 maintenant ce n'est plus la peine d'appeler `Class.forName ("com.mariadb.jdbc.Driver")`, qui est du 'vieux' code *(legacy code)* que vous trouverez sans doute sur internet. JDBC 4.1. qui est maintenant automatiquement associé à Java7, s'en charge.

AMELIORATION1 : utiliser un fichier de propriétés MariaDB.properties (et Oracle.properties) plutôt que d'écrire les propriétés de connexion en dur avec 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 deux améliorations sont présentées en fin du document.

Accès aux tables avec des DAO

Pour écrire les requêtes dans des classes séparées des classes Métier (et permettre des changements de BD sans bouleverser tout le code), on va créer une classe DAO (*Data Access Object*) *par objet persistant* : une classe VipDao pour les requêtes sur les VIP, une classe FilmDao pour les requêtes sur les films, etc. Et même mieux, pour permettre l'accès à des BD différentes, on crée d'abord une interface IVipDao.java, et une classe d'implémentation pour la BD cible : VipDaoOracle, VipDaoXML, etc.

On propose ici un exemple des éléments de DAO pour une table **d'articles** (attention, fonctionne avec une connexion par DataSource, cf amélioration proposée ci-après) :

(1) L'interface

```
package interfaceDAO;
import java.sql.Connection;
import java.util.List;
import javax.sql.DataSource;
import metier.Article;

/**
 * Interface DAO pour un article
 */
public interface IArticleDAO {

    public List<Article>    getLesArticles();
    public int                  setLesArticles(List<Article> lesArticles);
    public void                 insertArticle(Article a);
```

```
    public boolean          supprArticle(int refArticle);
    public void             setDataSource(DataSource ds);
    public void             setConnection(Connection connexionBD);
    public List<Article>    findById(int refArticle);
    public List<Article>    findByLibelle();
    public List<Article>    getArticlesCategorie(String uneCategorie );
}
```

(2) Une classe d'implémentation par classe Métier

Dans les classes d'accès aux données métier, on va implémenter les méthodes nécessaires de DAO, avec les ordres SQL. **On ne met jamais de requête SQL directement dans les classes Métier.**
Soit par exemple la classe Métier Article suivante, dont voici un extrait (pas les get/set) :

```
package metier;

/**
 * @author VDe – IUT LYON
 */
public class Article {

    private int id;
    private String libelle;
    private String categorie;
    private String sousCategorie;
    private Double prix;
    private int quantite;

    public Article() {
    }

    public Article(int cle, String lib, String cat, String souscat, double prix, int qtte) {

        this.id = cle;
        this.libelle = lib;
        this.categorie = cat;
        this.sousCategorie = souscat;
        this.prix = prix;
        this.quantite = qtte;
    }
// etc…
```

La classe d'implémentation DAO pour cette classe Article pourrait être par exemple :

```
package persistanceSQL;

import interfaceDAO.IArticleDAO;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.sql.DataSource;
import metier.Article;
```

```
/**
* TP JDBC - DAO - @author VDe, IUT de Lyon
*/
public class ArticleDaoSql implements IArticleDAO {

    private static Connection connexionBD;                    // (permet de changer de BD/connexion)

    // constructeur
    public ArticleDaoSql (Connection conn) {
        ArticleDaoSql.connexionBD = conn;
    }

    @Override
    public List<Article> getLesArticles() {
        connexionBD = ConnexionBD.getConnection();
        ResultSet rset = null;
        Statement stmt = null;
        List<Article> listeArticles = null;
        String query = "SELECT id, libelle, categorie, souscategorie, prix, quantite "
                + "FROM Produit.tableproduits order by categorie";
        Article art;
        try {
            stmt = connexionBD.createStatement();
            listeArticles = new ArrayList<>();
            rset = stmt.executeQuery(query);

            while (rset.next()) {
                // rappel du constructeur pour ne pas avoir d'erreur :
                // public Article(int cle, String lib, String cat, String souscat, double prix, int qtte)
                art = new Article(rset.getInt(1), rset.getString(2), rset.getString(3), rset.getString(4), rset.getDouble(5), rset
.getInt(6));
                listeArticles.add(art);
            }

        } catch (SQLException ex) {
            Logger.getLogger( ArticleDaoSql.class.getName() ).log(Level.SEVERE, null, ex);
        } finally {
            try {
                if (connexionBD != null) {
                    connexionBD.close();
                }
            } catch (SQLException ex) {
                System.out.println(ex.getMessage());
            }
        }
        return listeArticles;
    }

    @Override
    public void insertArticle(Article a) {
        connexionBD = ConnexionBD.getConnection();

        PreparedStatement pstmt = null;
        String sc = null;
        String query = "INSERT INTO Produit.tableproduits (id, libelle, categorie, souscategorie, prix, quantite)  VALUES
(?,?,?,?,?,?)";                    // 6 valeurs
```

```
        try {
            pstmt = connexionBD.prepareStatement(query);
            pstmt.setInt(1, a.getId());
            pstmt.setString(2, a.getLibelle());
            pstmt.setString(3, a.getCategorie());
            sc = a.getSousCategorie();
            if (sc != null) {
                pstmt.setString(4, sc);
            } else {
                pstmt.setNull(4, java.sql.Types.VARCHAR);
            }
            pstmt.setDouble(5, a.getPrix());
            pstmt.setInt(6, a.getQuantite());

            int n = pstmt.executeUpdate();
            if (n == 1) {
                System.out.println(" Insertion reussie d'un Article en BD ");
            } else {
                System.out.println(" *** Echec Insertion de l'Article en BD : cet ID existe déjà ");
            }

        } catch (SQLException ex) {
            Logger.getLogger( ArticleDaoSql.class.getName() ).log(Level.SEVERE, null, ex);
        } finally {
            try {
                if (connexionBD != null) {
                    connexionBD.close();
                }
            } catch (SQLException ex) {
                System.out.println(ex.getMessage());
            }
        }
    }

    @Override
    public void setConnection(Connection conn) {
        ArticleDaoSql.connexionBD = conn;
    }
    /**
     * Méthode qui enregistre une liste d'articles dans la BD d'un coup
     * c'est quasiment le même code qu'insererArticle() mais avec une boucle sur le
     * preparedStatement()
     */
    @Override
    public int setLesArticles(List<Article> lesArticles) {

        int nbArticlesInseres = 0;
        PreparedStatement pstmt = null;
        String sc = null;
        Article a = null;
        String query = "INSERT INTO Produit.tableproduits (id, libelle, categorie, souscategorie, prix, quantite)  VALUES
(?,?,?,?,?,?)";                    // 6 valeurs

        int nbArticles = lesArticles.size() ;
```

```
    try {
       for (int i = 0; i < nbArticles; i++) {

          a = lesArticles.get(i);
          pstmt = connexionBD.prepareStatement(query);
          pstmt.setInt(1, a.getId());
          pstmt.setString(2, a.getLibelle());
          pstmt.setString(3, a.getCategorie());
          sc = a.getSousCategorie();
          if (sc != null) {
             pstmt.setString(4, sc);
          } else {
             pstmt.setNull(4, java.sql.Types.VARCHAR);
          }
          pstmt.setDouble(5, a.getPrix());
          pstmt.setInt(6, a.getQuantite());

          int n = pstmt.executeUpdate();
          if (n == 1) {
             nbArticlesInseres++ ;
          } else {
             System.out.println(" *** Echec Insertion de l'Article en BD : cet ID existe déjà ");
          }
       }
    } catch (SQLException e) {
       System.out.print("Meth setLesArticles() : problème avec la BD..." + e.getMessage());
       Logger.getLogger( ArticleDaoSql.class.getName() ).log(Level.SEVERE, null, e);

    } finally {
       try {
          if (connexionBD != null) {
             connexionBD.close();
          }
       } catch (SQLException ex) {
          System.out.println(ex.getMessage());
       }
    } // bloc finally
    return nbArticlesInseres;

 } // de setArticles()
 // + code des autres méthodes… }
```

Ensuite selon les traitements, on appelle les méthodes de l'instance de DAO créée, par exemple :

```
// Création d'une instance de DAO Article avec les infos de connexion :
        articleDAO = new ArticleDAOSql( connexionBD );

// Insertion :
        Article a1 = new Article(7, "BoClavier", "Clavier", null, 29.99, 3);
        articleDAO.insertArticle(a1);

// Récupérer les articles puis affichage :
        lesArticles = articleDAO.getLesArticles();
        int n = lesArticles.size();              // nb d'articles

        if (n > 0) {
           System.out.println(n + " article(s) chargé(s)");
```

```
        for (int i = 0; i < n; i++) {
            System.out.println( lesArticles.get(i) );  // on a reecrit toString()
        }
    } else {
        System.out.println("ATTENTION ! Aucun article chargé !");
    }

 // Fermer la connexion :
connexionBD.close();
```

AMELIORATION de la connexion à la BD

On propose ici 2 améliorations :

- Faire la connexion à la BD via un **DataSource**, qui est la méthode à utiliser aujourd'hui. Un objet DataSource est un fichier de propriétés propre à la BD cible (Mysql.properties et Oracle.properties) cela évite d'écrire les propriétés de connexion en dur avec DriverManager. . Les éléments de connexion sont ainsi clairement dissociés du code qui utilise la connexion, et on peut facilement changer de BD si jamais c'est nécessaire. D'autre part, un DataSource gère un pool de connexions et optimise les transactions.

- **Saisir les *login* et mots de passe par une boîte de dialogue** plutôt que de les écrire de manière visible dans le code, à l'aide de la classe java.net.PasswordAuthentication.

Connexion à la BD avec un DataSource

(1) Définir le (ou les) *fichiers de propriétés* pour la (ou les) BD cible.

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

`connexionOracle.properties`

```
# Sample ResourceBundle properties file pour la connexion à Oracle serveur IUT de LYON, jamais de "
port=1521
# service=orcl suffit si vous êtes sur le domaine univ-lyon1 (wifi EDUROAM, ou connexion filaire)
service=orcl.univ-lyon1.fr
serveur=iutdoua-oracle.univ-lyon1.fr
pilote=thin
# vos login/pwd d'Oracle :
user=pxxxx
pwd=secret
# (on ne définit pas la BD car à l'IUT, la BD Oracle porte votre nom pxxxx)
```

FICHIER de propriétés : connexion à une BD locale via un serveur Apache WAMP / MAMP

`connexionMySQL.properties`

```
port=8889
serveur=localhost
```

```
#driver = com.mysql.jdbc.Driver
base=Produit
user=root
pwd=root
```

FICHIER de propriétés : connexion à une BD MariaDB sur le serveur web de l'IUT

`connexionMariaIUT.properties`

```
# (attention il faut être sur le réseau de l'Université : wifi EDUROAM
# ou connexion filaire, ou proxy Lyon1 bien défini)
port=3306
serveur=iutdoua-web.univ-lyon1.fr
base=pxxxxxx
user=pxxxxx
pwd=votre pwd ou code BIP (si pwd initial)
```

(2) Définir **la source de données pour la connexion**. On va ici définir la connexion pour MariaDB de l'IUT. Importer le **`DataSource`** de MariaDB (après avoir chargé le driver) :

```
import org.mariadb.jdbc.MariaDbDataSource;
```

Créer la classe **`MonMariaDbDataSource`** qui va hériter de la classe **`MariaDbDataSource.`** C'est un **`singleton`** : on a besoin d'une seule instance. Le constructeur est donc privé et ne s'exécute qu'avec l'appel d'une méthode publique, **`getMariaDbDataSource()`**, qui va vérifier s'il n'existe pas déjà une instance de la classe (statique).

```
public class MonMariaDbDataSource extends MariaDbDataSource {

    private static MonMariaDbDataSource mds;

    /**
     * constructeur privé vide, utilisé dans getMdbDataSource(), c'est juste pour dire qu'il est privé
     */
    private MonMariaDbDataSource () {
    }

    /**
     * Méthode statique qui renvoie l'unique instance de MonMariaDbDataSource construite à partir du fichier de proprietes
     * @return une instance de MariaDbDataSource:
     */
    public static MonMariaDbDataSource getMdbDataSource() {

        // contrôle si un datasource n'existe pas deja
        if (mds == null) {
            Properties prop = new Properties();
            FileInputStream fichier = null;

            try {
                fichier = new FileInputStream("src/persistanceIUT/connexionMariaIUT.properties");

            } catch (FileNotFoundException ex1) {
                System.out.println("Fichier de proprietes non trouvé");
            }
            try {
                prop.load(fichier);

            } catch (IOException ex) {
                System.out.println("Erreur lors du chargement du fichier de proprietes mySQL");
```

```
        } finally {
            try {
                fichier.close();
            } catch (IOException ex) {
                System.out.print("Problème d'entree/sortie" + ex.getMessage());
            }
        }

        mds = new MonMariaDbDataSource ();

        mds.setPortNumber(new Integer(prop.getProperty("port")));
        mds.setServerName(prop.getProperty("serveur"));
        mds.setDatabaseName(prop.getProperty("base"));
        mds.setUser(prop.getProperty("user"));
        mds.setPassword(prop.getProperty("pwd"));
        // pas de service à définir pour MariaDB

      } else System.out.println("---(la source de data existe deja)") ;

      return mds;

   } // de getMdbDataSource()

}
```

La connexion à la BD s'effectue en appelant la méthode `getConnection()` des classes `datasource`.

C'est ainsi dans le *main* (ou dans le constructeur de la fenêtre) qu'on définit l'accès à la Base de données avec le `datasource` qui convient :

```
        // Création de la connexion :
        dataSourceDAO = MonMySQLDataSource.getMySQLDataSource();
        connexionBD = dataSourceDAO.getConnection();

        // Création d'une instance de DAO Article avec les infos de connexion :
        articleDAO = new ArticleDAOSql( connexionBD );

        // on pourra ainsi changer de connexion :
        //articleDAO.setConnection(uneAutreConnexion);
```

Classe JDialog pour une authentification propre

On crée une fenêtre modale avec un composant de l'API Java qui permet de gérer les informations de login de façon cachée, `java.net.PasswordAuthentication` :

```
package vue;

import java.net.PasswordAuthentication;

public class JDAuthentificationBD extends javax.swing.JDialog {

    private javax.swing.JButton jButtonOK;
    private javax.swing.JLabel lblUser;
    private javax.swing.JLabel lblPwd;
    private javax.swing.JPasswordField tfPwd;
```

```
    private javax.swing.JTextField tfUser;

    /**
     * Creates new form JDAuthentificationBD
     */
    public JDAuthentificationBD (java.awt.Frame parent) {
        super(parent, "Identification BD", true);
        initComponents();
    }

    private void initComponents() { //…    }

    public PasswordAuthentication recuperer() {
        setVisible(true);
        return  new PasswordAuthentication( tfUser.getText(), tfPwd.getPassword() );
    }

    private void Quitter(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }

    private void jButtonOKActionPerformed(java.awt.event.ActionEvent evt) {
        this.dispose();     // rend la main à la la fenêtre parent
    }
}
```

Boucle pour capturer les *login/pwd* à partir de cette `JDialog`, par ex. à la première action nécessitant une connexion (ouverture de la fenêtre par ex.) :

```
        boolean etatConnecte = false;
        do {
            // declaration d'un bloc d'identification en fenêtre modale (this ici = fen. principale)
            JDAuthentificationBD saisieMdp = new JDAuthentificationBD (this);
            PasswordAuthentication identification = saisieMdp.recuperer();
            // gestion de la connexion
            try {
                // connexion
                    connexionBD = dataSourceDAO.getConnection( identification );
                // si connexion reussie
                etatConnecte = true;
                // chargement du conteneur depuis la base
                connexionBD.charger( listeArticles) ;
            } catch (Exception e) {
                // si impossibilite de se connecter
                JOptionPane.showMessageDialog(this, e.getMessage(),
                        "Probleme d'identification", JOptionPane.WARNING_MESSAGE);
            }
        } while (!etatConnecte);
```

Il faut alors modifier le code du datasource pour prendre en compte l'identification des login/mdp :

```
public static MonMariaDbDataSource getMdbDataSource( PasswordAuthentication identification ) {
…
        mds.setUser( identification.getUserName() );
        // conversion nécessaire d'un tableau de char en String :
        mds.setPassword( new String( identification.getPassword() ) );
…
```