Introduction à JDBC

Préparé par Saliha Yacoub

Avril 2011 (mise à jour 2016)

Table des matières

Introduction à JDBC

Introduction

JDBC, Java Data Base Connectivity est un ensemble de classes (API – Application Programming Interface --JAVA) permettant de se connecter à une base de données relationnelle en utilisant des requêtes SQL ou des procédures stockées.

L'API JDBC a été développée de manière à pouvoir se connecter à n'importe quelle base de données avec la même syntaxe; cette API est dite indépendante du SGBD utilisé.

Les classes JDBC font partie du package **java.sql** et **javax.sql**

JDBC permet entre autre :

1. L'établissement d'une connexion avec le SGBD.
2. L'envoi de requêtes SQL au SGBD, à partir du programme java.
3. Le traitement, au niveau du programme, des données retournées par le SGBD.
4. Le traitement des erreurs retournées par le SGBD lors de l'exécution d'une instruction.

Pilote de bases de données ou driver JDBC

- Un pilote ou driver JDBC est un "logiciel" qui permet de convertir les requêtes JDBC en requêtes spécifiques auprès de la base de données.
- Ce "logiciel" est en fait une implémentation de l'interface Driver, du package java.sql.

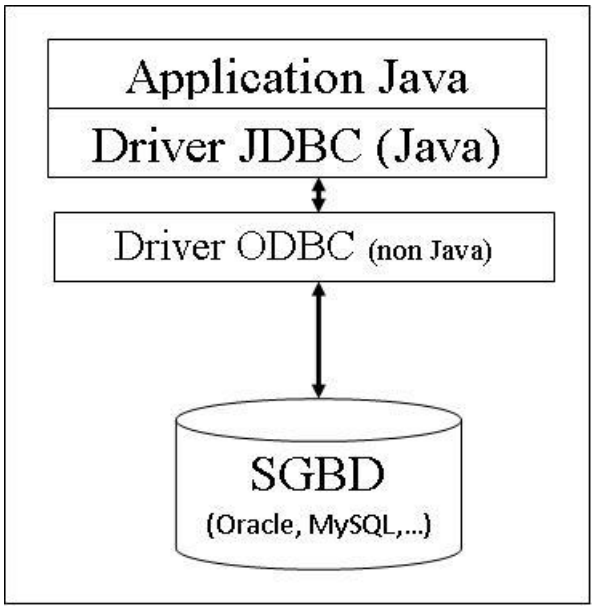
Dans le cas d'oracle, les drivers JDBC sont fournis par Oracle (en principe installés avec la base de données) téléchargeables à l'adresse.

http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html

Types de drivers JDBC

Il existe plusieurs types de pilotes JDBC

Les drivers de Type 1 : ODBC-JDBC bridges, ODBC (Open Data Base Connectivity) est une interface propre à Microsoft et qui permet l'accès à n'importe quelle base de données (Panneau de configuration /Outils d'administration/ Sources de données ODBC



Chaque requête JDBC est convertie par ce pilote en requête ODBC qui est par la suite convertie une seconde fois dans le langage spécifique de la base de donnée.

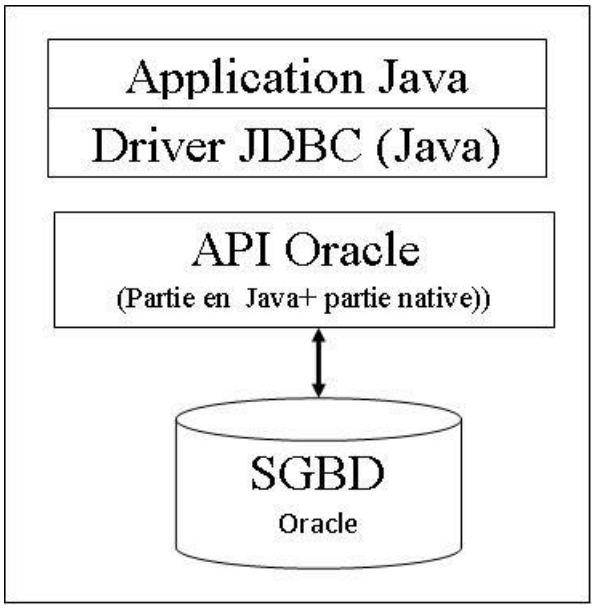
Cette technique est la moins optimale puisque les bases de données sont disponibles uniquement que par technologie ODBC.

Le SDK de Java fournit un pilote JDBC-ODBC :« sun.jdbc.odbc.JdbcOdbcDriver ».

Les drivers de Type 2

Ce type de driver traduit les appels de JDBC à un SGBD particulier, grâce à un mélange d'API java et d'API natives. (Propre au SGBD).

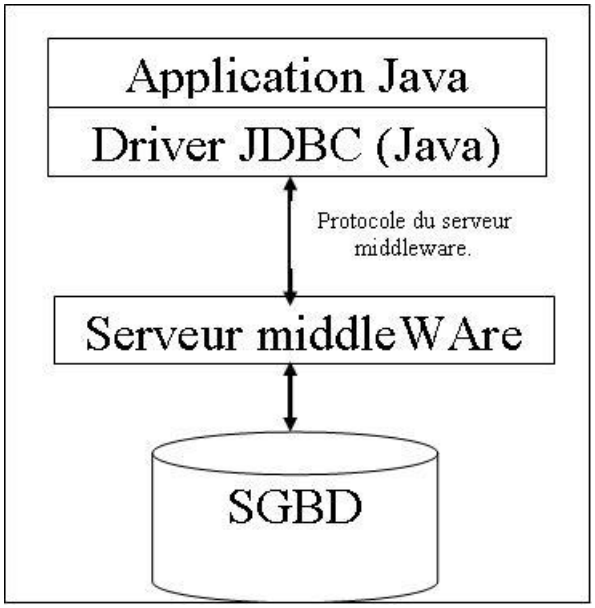
Ce Driver est fourni par l'éditeur de SGBD



Il est de ce fait nécessaire de fournir au client l'API native de la base de données.

Si on change le type de la base de données, on doit changer le pilote.

Drivers de type 3 (complètement écrit en JAVA)



Permet la connexion à une base de données via un serveur intermédiaire régissant l'accès aux multiples bases de données

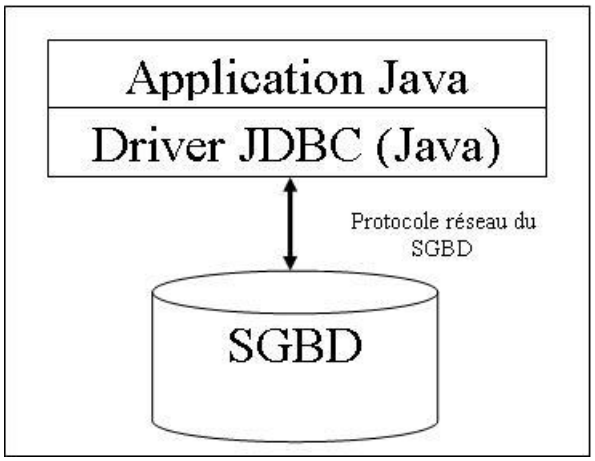
Ce type de driver est portable car écrit entièrement en java. Il est adapté pour le Web.

Cela exige une autre application serveur à installer et à entretenir.

Ce type de driver peut être facilement utilisé par une applet, mais dans ce cas le serveur intermédiaire doit obligatoirement être installé sur la machine contenant le serveur Web.

Drivers de type 4 (complètement écrit en JAVA)

Ce type de driver est connu sous le nom Direct Database Pure Java Driver), permet d'accéder directement à la base de données (sans ODBC ni Middleware). C'est le type le plus optimal.



C'est ce type de driver qui sera utilisé pour accéder aux bases de données oracle

Dans ce type de driver on retrouve le driver pour oracle (thin driver ou oracle.jdbc.driver.OracleDriver) dont le format de la chaine de connexion à une base de donnes est sous formes :jdbc:oracle:thin:@chainedeconnexion

Les drivers pour Oracle

Oracle supporte les drivers suivants :

Le JDBC Thin driver : c'est le driver par excellence. Il est de type 4, donc entièrement écrit en java. Il peut être utilisé dans les applications ou des applets. Pas besoin d'utres installation de logiciel oracle pour son fonctionnement. Il utilise le Oracle Net Services pour communiquer avec la base de données.(Oracle Net Services = ensemble de programmes permettant aux applications clientes de communiquer avec la base de données.

Oracle recommande d'utiliser le thin driver sauf s'il n'est pas supporté par l'application

L'URL de connexion incluant le type de river est :

url="jdbc:oracle:thin:@IP:PORT:ServiceName";

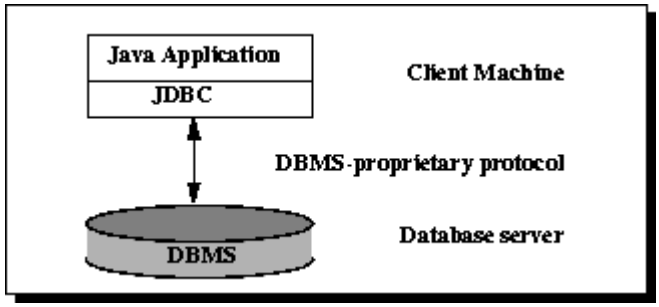
Le JDBC OCI (Oracle Call Inteface) , qui est un driver de type 2, écrit en Java et C et peut être utilisé uniquement avec des applications.

JDBC Server side thin Driver : identique au thin driver mais roule sur le serveur. Utilisé pour accéder à d'autres bases de données distantes.

Architecture

JDBC fonctionne selon les deux modèles suivants :

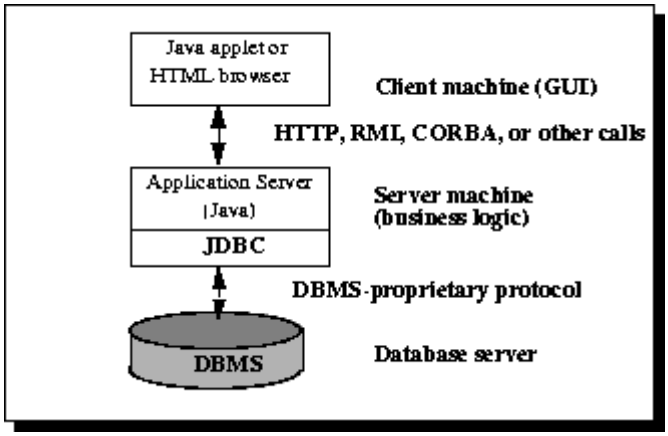
Modèle à deux couches (two-tier)



Dans le modèle two-tier, une application JAVA (ou une applet) dialogue avec le SGBD par l'intermédiaire du pilote JDBC. L'application JAVA et le pilote JDBC s'exécutent sur l'ordinateur client tandis que le SGBD est placé sur un serveur.

C'est ce type d'architecture qui nous concerne actuellement dans notre cours.

Modèles 3 couches (three-tier)



Dans le modèle three-tier, l'applet (ou l'application JAVA) ne dialogue plus directement avec un SGBD : un **middle-tier** fait le lien entre ces deux composants

Le SGBD exécute les requêtes SQL et envoie les résultats au middle tier. Ces résultats sont ensuite communiqués à l'applet sous forme d'appels http.

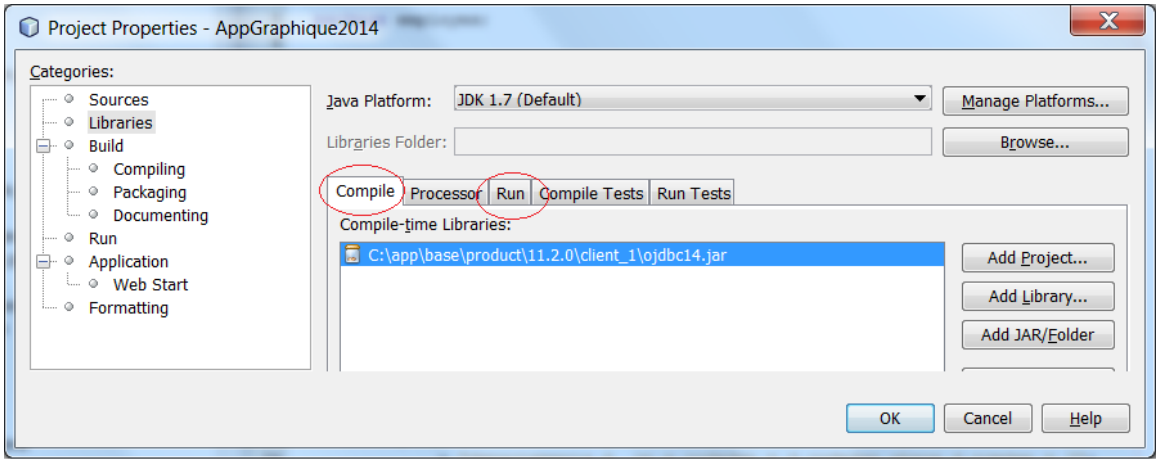
Fonctionnement

Tout programme JDBC fonctionne selon les étapes suivantes :

1. **Importer les packages nécessaires**
2. **Connexion à la base de données**
    i. Chargement du pilote de la BDD
    ii. Demande de connexion: s'identifiant auprès du SGBD et en précisant la base utilisée

3. **Traitement des commandes SQL**
4. **Traitement des résultats (Objet ResultSet)**
5. **Fermeture du RsultSet et du Statement**
6. **Mettre à jour la base de de données**
7. **Valider les mises à jour (COMMIT)**
8. **Fermeture de la connexion**.

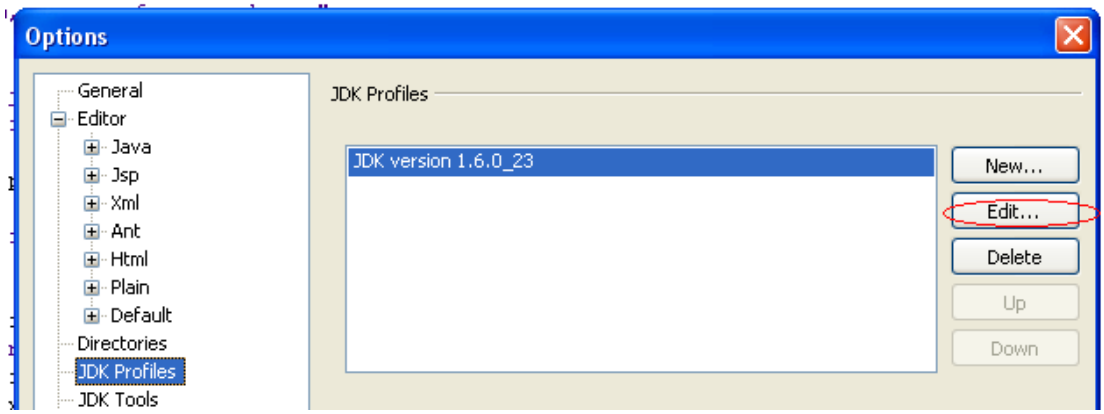
Pour utiliser JDBC avec NetBeans et oracle, vous devez inclure les librairies (.Jar) à votre projet : ces librairies sont, selon la version de votre JDK, ojdbc14.jar (pour JDK 1.6) .Pour ajouter ces librairies au projet cliquez sur bouton droit, puis propriétés



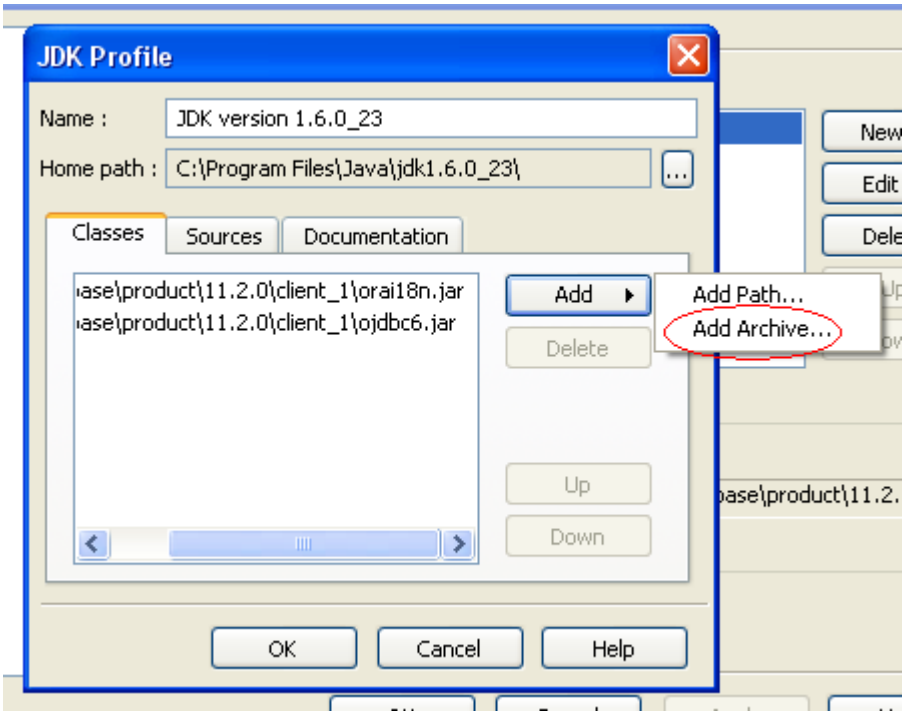
Ces librairies (pour oracle) sont dans le répertoire
C:\app\base\product\11.2.0\client_1

Pour utiliser JDBC avec Jcreator, il faut suivre les étapes :

Configure puis Options, choisir JDK Profiles, puis EDIT



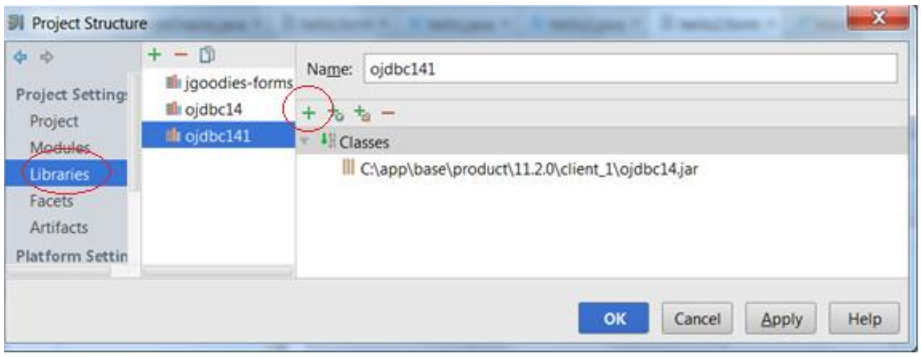
Puis Add Archive. La version actuelle est **ojdbc14.jar**



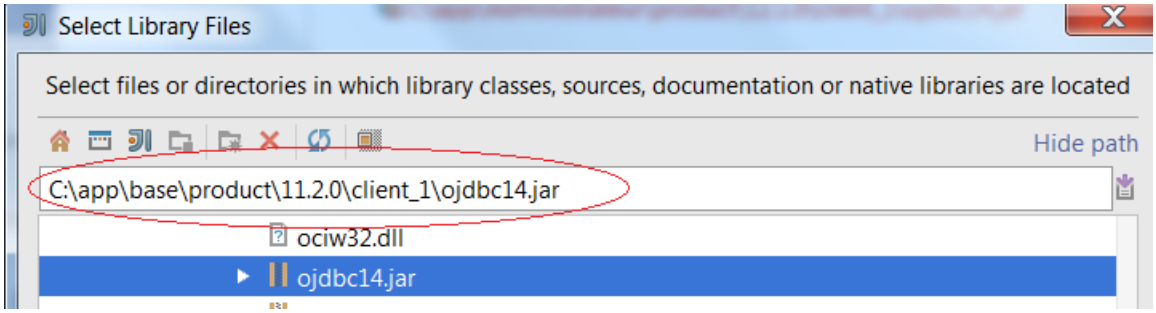
Les librairies pour Oracle sont dans **C:\app\base\product\11.2.0\client_1**

Pour utiliser JDBC avec Intellij Idea

Par le menu Fichier, Project Structure, à l'onglet Libraire, Ajouter



Puis, chercher ojdbc14.jar.



Établissement d'une connexion : versions du JDBC antérieurs à 4

1. **Importer le packages**
   import java.sql.*;
   import oracle.jdbc.driver.*;
   (d'autres packages seront nécessaires plus tard)
2. **Chargement du pilote (driver)**

Pour établir une connexion, il faut d'abord charger le driver en utilisant la méthode forName de la classe Class comme suit : **Class.forName(string driver)**. Pour oracle, l'instruction est la suivante :

```
Class.forName ("oracle.jdbc.driver.OracleDriver");
```

Quand une classe **Driver** est chargée, elle doit créer une instance d'elle même et s'enregistrer auprès du **DriverManager.**

La méthode **Class.forName(string driver)** fait partie du package **java.lang et** peut lancer une exception de type «ClassNotFoundException ».

Méthode 1 : Utilisation de la classe Class

Exemple1.java

```
package Exemple1;
import java.sql.*;
import oracle.jdbc.OracleDriver;

public class Exemple1 {

    public static void main(String[] args) {
        Connection conn = null;
        String user="user1";
        String mapasse ="user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";
        String sql1 ="select nomemp, prenomemp from employesclg where numemp =4";
        ResultSet rst;

        try    {
         Class.forName("oracle.jdbc.driver.OracleDriver");
          System.out.println("pilote chargé");
              }
        catch(ClassNotFoundException cnfe)
               {
        System.out.println("driver manquant");
               }
```

Méthode 2 : Utilisation du DriverManager.

On peut également demander à charger le driver auprès du DriverManager avec la méthode registerDriver comme suit

```
try  {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        System.out.println("pilote chargé");
      }
catch(SQLException sqldriver)
    {  System.out.println("driver manquant" );
    }
```

Remarquez que dans le premier cas, où on utilise la classe Class, le type d'exception renvoyée n'est pas SQL. C'est ClassNotFoundException. Pour la méthode forName, on passe le nom du Driver sous forme de chaîne caractère.

Dans le deuxième cas, le nom du driver est fourni directement à la méthode **registerDriver** est l'exception est de type SQL.

3. **Demander une connexion**
   Que l'on utilise la méthode 1 ou la méthode 2, une fois que le driver est chargé, il faudra demander une connexion.

   Cette connexion est obtenue grâce à la méthode **getConnection** de la classe DriverManager

   Cette méthode retourne la connexion qui est en fait, un **objet** implémentant l'interface «Connection».

   Connection connexion = DriverManager.getConnection(url); url désigne la chaine de connexion , dans le cas d'oracle la chaine de connexion est de forme :

   "jdbc:oracle:thin:@IP:port:orcl", "nomUsager","Motdepasse"

   Exemple :

```java
package Exemple1;
import java.sql.*;
import oracle.jdbc.OracleDriver;

public class Exemple1 {

    public static void main(String[] args) {
        Connection conn = null;
        String user="user1";
        String mapasse ="user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";

         try  {
                DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
                System.out.println("pilote chargé");
             }
         catch(SQLException sqldriver)
             {  System.out.println("driver manquant" );
             }
        // on ouvre une connexion avec les paramètres de connexion.
           try      {
            conn = DriverManager.getConnection(url,user,mapasse);
            System.out.println("vous êtes connectés");
                    }
          catch(SQLException sqlconnect){ System.out.println("connexion impossible");}
```

Toute connexion ouverte doit être fermée. La méthode Close de l'objet Connection permet de fermer une connexion.

```
import java.sql.*;
public class Exemple1 {

    public static void main(String[] args) {
        Connection conn = null;
        String user="user1";
        String mapasse ="user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";

         try  {
                DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
                System.out.println("pilote chargé");
              }
         catch(SQLException sqldriver)
             {  System.out.println("driver manquant" );
             }
        // on ouvre une connexion avec les paramètres de connexion.
          try      {
           conn = DriverManager.getConnection(url,user,mapasse);
           System.out.println("vous êtes connectés");
                  }
         catch(SQLException sqlconnect){ System.out.println("connexion impossible");}
          //on ferme la connexion
        finally
             {
              try
              {
              if (conn!=null)
                  conn.close();
              System.out.println("connexion fermée");
               }
              catch(SQLException se){}
             }
    }
}
```

Obtenir une connexion à une base de données Mysql

Mysql.java ×

```
package Mysql;
import java.sql.*;
public class Mysql {

    public static void main( String args[] ) {
        Connection conn = null;
        Statement stm = null;
        ResultSet rst=null;
        String ussr = "aaaaaa";
        String Bd = "jdbc:mysql://localhost:3306/depinfo2015";
        String mp = "aaaaa";
        String sql2 = "select nomgroupe from groupes where codegroupe ='inf1'";
        try {
            Class.forName("com.mysql.jdbc.Driver");

            System.out.println("Pilote Chargé");
             }
        catch (ClassNotFoundException e) {
            System.err.println(e.getMessage());
            System.exit(0);
             }
        try {
            conn = DriverManager.getConnection(Bd, ussr, mp);
            System.out.println("Pilote Chargé, Connexion ouverte");

            try{
                stm = conn.createStatement();
                rst = stm.executeQuery(sql2);
                if (rst.next())
                {
                    System.out.println("le nom du groupe est" + "   "+ rst.getString(1) );
                }
               }

            catch (SQLException sql)
            {
                System.out.println(sql.getMessage());
                System.exit(0);
            }
            finally {
                stm.close();
                rst.close();
                    }
             }
```

Suite du code : On complète avec le catch du try de connexion et le finally pour fermer la connexion

```
        catch (SQLException seq) {
            System.out.println(seq.getMessage());
            System.exit(0);
                }

        finally
        {
            try {
                if (conn != null)
                    conn.close();
                System.out.println("connexion fermée");

                }
            catch (SQLException se) {
            }
        }

    }
    }
```

Obtenir une connexion à une base de données Sqlite :

On ouvre une connexion, puis on exécute une requête de type SELECT.

SqLite.java

```
package Sqlite;
import java.sql.*;
public class SqLite {

    public static void main( String args[] )
    {
        Connection conn = null;
        Statement stm = null;
        ResultSet rst = null;
        String sql2 = "Select race from chats where nom ='Ruby'";

        try {
            Class.forName("org.sqlite.JDBC");

            System.out.println("Pilote Chargé");
             }
        catch ( ClassNotFoundException e )
            {
            System.out.println(e.getMessage());
            System.exit(0);
            }
        try {
            conn = DriverManager.getConnection("jdbc:sqlite:C:/Sqlite/BdLite.db;");
            System.out.println("Pilote Chargé, Connexion ouverte");

                    try{
                    stm = conn.createStatement();
                    rst = stm.executeQuery(sql2);
                    if (rst.next())
                    {
                   System.out.println("la race est" + "   "+ rst.getString(1) );
                    }

                    }

                    catch (SQLException sql)
                    {
                   System.out.println(sql.getMessage());
                   System.exit(0);
                       }
                   finally {
                    stm.close();
                   rst.close();
                    }
            }
```

Suite du code : On complète avec le catch du try de connexion et le finally pour fermer la connexion

```
        catch (SQLException sed )
        {
            System.out.println(sed.getMessage());
            System.exit(0);
        }

        finally
        {
            try {
                if (conn != null)
                    conn.close();
                System.out.println("connexion fermée");

            }
            catch (SQLException se) {
            }
        }


    }
}
```

Établissement d'une connexion : version 3 et plus du JDBC

Dans la version 4 du JDBC, il n'est pas nécessaire de charger explicitement le driver avec la méthode forName de la classe Class ou d'enregistrer explicitement le driver auprès du DriverManager.

Le DataSource permet d'utiliser un pool de connexion, qui est un mécanisme de réutilisation des connexions créées. Un pool de connexions ne ferme pas les connexions lors de l'appel à la méthode *close()*. Au lieu de fermer directement la connexion, celle-ci est "retournée" au pool et peut être utilisée ultérieurement. La gestion du pool est transparente pour l'utilisateur.

Un objet OracleDataSource définit un ensemble de méthodes permettant l'accès à la base de données Oracle.

Constructeur : OracleDataSource()

OracleDataSource ods = new OracleDataSource();

Méthode importantes :

getConnection()	Obtient une connexion à la base de données
setServerName	Définit le nom du serveur de la base de données utilisée. (ou son adresse IP)
setPortNumber	Définit le port du serveur de base s de données
setPassword	Définit le mot de passe avec lequel la connexion est obtenue.
setUser	Définit le username de l'usager
setServiceName	Définit le nom du service de la base de donnée : orcl
setDatabaseName	Définit le nom de la base de donnée : orcl
setURL	Définit la chaîne de connexion qui sera utilisée pour se connecter à la base de données. Elle est de la forme String url="jdbc:oracle:thin:@111.111.111.111:1521:orcl". Au lieu de définir l'IP , le protocole, le service name séparément, il est préférable d'utiliser l'URL.
getServerName	obtient le nom du serveur de la base de données utilisée.

	(ou son adresse IP)
getPortNumber	obtient le port du serveur de base s de données
getPassword	obtientt le mot de passe avec lequel la connexion est obtenue.
getUser	Obtient le username de l'usager
getServiceName	Obtient le nom du service de la base de donnée : orcl
getDatabaseName	Obtient le nom de la base de donnée : orcl

Pour les autres méthodes, veuillez consulter le site :

http://docs.oracle.com/cd/E11882_01/appdev.112/e13995/oracle/jdbc/pool/OracleDataSource.html

1. Importer les packages :

```
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.*;
```

2. La connexion elle-même

```
String user1 ="user1";
 String mdep ="oracle1";
 String url="jdbc:oracle:thin:@205.237.244.251:1521:orcl";

//déclarer un objet OracledataSource
OracleDataSource ods = new OracleDataSource();

// définir les paramètres de connexion pour l'objet OracleDataSourse ods
    ods.setURL(url);
    ods.setUser(user1);
    ods.setPassword(mdep);

// Appel de la méthode getConnection pour obtenir une connexion
Connection conn = ods.getConnection();
```

3. Exemple complet

```
package Exemple1;
import java.sql.*;
import oracle.jdbc.pool.*;
public class Exemple1 {

    public static void main(String[] args) {
        Connection conn = null;
        String user="user1";
        String mapasse ="user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";

    try {
        OracleDataSource ods = new OracleDataSource();
        ods.setURL(url);
        ods.setUser(user);
        ods.setPassword(mapasse);
        conn = ods.getConnection();
        System.out.println("vous etes connectés ");
        }

        catch(SQLException sqlods)
         {
         System.out.println("connexion impossible");
         }

        finally
             {
                try
                {
                if (conn!=null)
                    conn.close();
                System.out.println("connexion fermée");
                 }
                catch(SQLException se){}
             }

 }

}
```

Fermeture d'une connexion :

La connexion est fermée avec la méthode close de l'objet **connexion.close();**

Exécution de requêtes SQL : créer un « statement » d'une requête particulière.

Cette étape consiste à obtenir une **déclaration (zone de description de requête ou «*statement »)*** au travers de laquelle les requêtes SQL seront exécutées.

Il existe 3 types de déclarations:

1. Statement, instruction simple : permet d'exécuter directement et une fois l'action sur la base de données :

   Statement declaration1= connexion.createStatement();

2. PreparedStatement: instruction paramétrée. (cas des requêtes avec paramètres)

   - L'instruction est générique, des champs sont non remplis
   - Permet une précompilation de l'instruction optimisant les performances
   - Pour chaque exécution, on précise les champs manquants

   PreparedStatement declaration2= connexion.PreparedStatement (requetesql);

3. CallableStatement:

Une déclaration de type « CallableStatement» permet l'accès complet aux fonctions contenues dans la base de données.(cas des procédures stockées)

Exécution de requêtes SQL : executeUpdate; executeQuery, execute

A. **La méthode ExecuteUpdate** est utilisée pour les requêtes DML (INSERT, DELETE, UPDATE).

Cette méthode retourne le nombre de lignes affectées par la requête sql.

objetStatement.**executeUpdate**(String Requête_SQL]);

ou

objetPreparedStatement.**executeUpdate**(String Requête_SQL]);

Exemple:

```
package Exemple1;
import java.sql.*;
import oracle.jdbc.pool.*;
public class Exemple1 {

        public static void main(String[] args) {
        Connection conn = null;
        String user = "user1";
        String mapasse = "user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";
       String sqlIns = "insert into employesclg(numemp,
nomemp,prenomemp) values(25,'Alpha','Omega')";

        try {
            OracleDataSource ods = new OracleDataSource();
            ods.setURL(url);
            ods.setUser(user);
            ods.setPassword(mapasse);
            conn = ods.getConnection();
            System.out.println("vous êtes connectés ");

                    try
                      {
                      Statement stmins = conn.createStatement();
                      int n = stmins.executeUpdate(sqlIns);
                      System.out.println("nb de lignes ajoutée" + n);
                      }
                     catch (SQLException sqlinsertion)
                      {
                      System.out.println(sqlinsertion.getMessage());
                      }

                      finally
                      {
                      stmins.close();
                      }
```

```
        }
        catch (SQLException sqlods)
        {       System.out.println("connexion impossible");}


         finally {
            try {
                if (conn != null)
                    conn.close();
                System.out.println("connexion fermée");
            } catch (SQLException se) {
            }
        }
    }

}
```

B. **La méthode executeQuery,** permet d'exécuter une instruction SQL de type SELECT

Elle retourne un objet de type **ResultSet** contenant tous les résultats de la requête (les tuples sélectionnés).

Syntaxe

objetResultSet=objetStatement.**executeQuery(**String ordreSQL);

ou

objetResultSet=objetPreparedStatement.**executeQuery(**String ordreSQL);

Exploitation des résultats des requêtes SQL

L'interface ResultSet représente une table de lignes et de colonnes.

- Une ligne représente un enregistrement
- Une colonne représente un champ particulier de la table
- Un objet de type RsultSet possède un pointeur sur l'enregistrement courant. À la réception de cet objet, le pointeur se trouve devant le premier enregistrement.
- On y accède ligne par ligne, puis colonne par colonne dans la ligne.

Pour pouvoir récupérer les données contenues dans l'instance de **ResultSet**, celui-ci met à disposition des méthodes permettant de :

- Positionner le curseur sur l'enregistrement suivant avec la méthode next()
  - public boolean **next();** Renvoi un booléen indiquant la présence d'un élément suivant.
- Accéder à la valeur d'un champ (par indice ou par nom) de l'enregistrement actuellement pointé par le curseur avec les méthodes getString(), getInt(), getDate() .....
  - public String getString(int indiceCol);
  - public String getString(String nomCol);
  - etc.

À la création du ResultSet, le curseur de parcours est positionné avant la première occurrence à traiter. **Le premier indice étant 1**

```
package Exemple1;
import java.sql.*;
import oracle.jdbc.pool.*;
public class Exemple1 {
         public static void main(String[] args) {
        Connection conn = null;
        String user = "user1";
        String mapasse = "user1";
        String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";
         String sql1 = "select nomemp, prenomemp from employesclg ";
       Statement stm1 = null;
       ResultSet rst = null;
         try {
            OracleDataSource ods = new OracleDataSource();
            ods.setURL(url);
            ods.setUser(user);
            ods.setPassword(mapasse);
            conn = ods.getConnection();
            System.out.println("vous êtes connectés ");
               try {
                       stm1 = conn.createStatement();
                       rst = stm1.executeQuery(sql1);
                       while (rst.next())
                               {
                       //String nom1 = rst.getString(1);
                       //String prn1 = rst.getString(2);
                       String nom1 = rst.getString("nomemp");
                       String prn1 = rst.getString("prenomemp");
                       System.out.println( nom1  + "------" + prn1 );
                               }
                  }
               catch (SQLException sqlselect)
                       {
                        System.out.println(sqlselect.getMessage());
                       }
```

```
                finally {
                stm1.close();
                rst.close();
                        }


                } // premier try
        catch (SQLException sqlods)
        {       System.out.println("connexion impossible");}

         finally {
            try {
                if (conn != null)
                    conn.close();
                System.out.println("connexion fermée");
            } catch (SQLException se) {
            }
        }
    }

}
```

Remarque : Il est important de fermer le Statement et le ResultSet avec la méthode close() juste après leur utilisation pour libérer immédiatement les ressources qu'ils occupent. S'ils ne sont pas fermé par la méthode Close(),alors il le seront lorsque l'application sera fermées, mais ce n'est pas une façon recommandée.

C. **Méthode execute** (String ordre) : Retourne « **true** » si un résultat est disponible, « **false** » si non.

valeurbooléenne=objetStatement.**execute** (String ordre);

valeurbooléenne=objetPreparedStatement.**execute** (String ordre);

Exemple d'utilisation de la méthode Execute().

Dans le code suivant seul le contenu du try est indiqué.

Si la méthode Execute est utilisée alors, pour obtenir le ResultSet, il faut utiliser la méthode **getResultSet** du Statement (dans le cas qui nous concerne).

```
Statement stm = null;
ResultSet rst = null;
boolean nonvide;
String sql4 ="SELECT * FROM EMPLOYESBIDON";


stm =conn.createStatement();

nonvide = stm.execute(sql4);
 if(nonvide)
  {
      rst=stm.getResultSet();



      while (rst.next())
      {
          String nom1 = rst.getString(1);
          String prn1 = rst.getString(2);
          System.out.println( nom1  + "------" + prn1 );
      }


 }

// Suite du code ....
```

Utilisation du PreparedStatement

Ce type d'interface est utilisé pour des requêtes paramétrées. PreparedStatement est utilisé dans le cas où la requête va être exécutée plusieurs fois. De plus les requêtes sont précompilées.

Remarquez

1. Dans le requête le paramètre est représenté par **?**
2. Les paramètres sont passés dans l'ordre de leur présentation de la requête
3. Les paramètres et les valeurs sont passés comme suit :

- [Objet PreparedStatement].setString([index],[objet String]);
- [Objet PreparedStatement].setBoolean([index],[valeur]);
- [Objet PreparedStatement].setInt([index],[valeur]);
- [Objet PreparedStatement].setFloat([index],[valeur]);
- Etc…

4. La requête est exécuté par executeUpdate() ou executeQuery
5. On peut effacer le contenu des paramètres par la méthode ClearParameters()

Exemple : : Nous sommes déjà connétés.

```
conn = ods.getConnection();
```

```
String sql3 = "select nomemp, prenomemp from employesbidon where
emploi =?";

PreparedStatement stm = null;
ResultSet rst = null;

        try {
           stm = conn.prepareStatement(sql3);
            stm.setString(1,"ANALYSTE");
            rst = stm.executeQuery();
            while (rst.next())
            {
                String nom1 = rst.getString("nomemp");
                String prn1 = rst.getString("prenomemp");
                System.out.println( nom1  + "------" + prn1 );
            }
            stm.clearParameters();
            }
```

```
 catch (SQLException sqlselect)
  {
 System.out.println(sqlselect.getMessage());
  }
finally {
      stm.close();
      rst.close();
        }
```

Type de parcours du ResultSet

Il est possible de parcourir le ResultSet de trois façons différentes selon le type de ce dernier. Par défaut, le parcourt est farward only.

ResultSet.TYPE_FORWARD _ONLY : accès séquentiel

ResultSet.TYPE_SCROLL_INSENSITIVE, accès direct sans modification (les occurrences ne reflètent pas les mises à jour qui peuvent intervenir durant le parcours) Permet de parcourir les résultats dans les deux sens.

1. Permet de parcourir les résultats dans les deux sens grâce aux méthodes:
    - Public boolean next();
    - public boolean previous();
    - Public boolean first();
    - Public boolean last();
2. Permet de connaitre la position courante du curseur à l'intérieur du ResultSet
    - Public boolean isBeforFirst();
    - public boolean isAfterLast();
    - Public boolean isFirst();
    - Public boolean isLast();

ResultSet.TYPE_SCROLL_SENSITIVE accès direct avec modification

Même principe que le type précédent sauf que, les occurrences reflètent les mises à jour qui peuvent intervenir durant le parcours

Modification des données du ResultSet:

Par défaut, un ResultSet contient des données en lecture seulement, mais il est possible d'obtenir un ResultSet modifiable.

ResultSet.CONCUR_READ_ONLY : lecture seule

ResultSet.CONCUR_UPDATABLE : mise à jour

Le type de ResultSet et le mode d'utilisation (read only ou updatable) doit se faire lors de la création du Statement ou du PreParedStatement

```
String sql3 = "select nomemp, prenomemp from employesbidon where
emploi =?";

PreparedStatement stm = null;
ResultSet rst = null;

        try {
           stm = conn.prepareStatement(sql3,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

            stm.setString(1,"ANALYSTE");
            rst = stm.executeQuery();
            while (rst.next())
            {
                String nom1 = rst.getString("nomemp");
                String prn1 = rst.getString("prenomemp");
                System.out.println( nom1  + "------" + prn1 );
            }
            stm.clearParameters();
            }



    catch (SQLException sqlselect)
     {
    System.out.println(sqlselect.getMessage());
     }
   finally {
         stm.close();
         rst.close();
           }
```

Exemple

```
Statement stm2
=connexion.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
```

Dans le cas d'une modification (updatable), voici les opérations pour une mise à jour ou une insertion

Modifier la valeur du type et de la colonne donnée (par indice ou par nom) de l'enregistrement actuellement **pointé** :

- public void updateString(int indiceCol, String value);
- public void updateString(String nomCol, String value);
- public void updateInt(int indiceCol, Int value);
- public void updateInt(String nomCol, Int value);
- etc.

Appliquer dans la base de données les changements effectués sur l'enregistrement actuellement pointé : public void **updateRow**();

Exemple; (String sql2 = "select nom, prenom from employes";)

```
Statement stm4 =connexion.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
    ResultSet rst4=stm4.executeQuery(sql2);
    rst4.next();
    rst4.updateString("nom", "Coluche");
    rst4.updateString("prenom", "Moses");
    rst4.updateRow();
    connexion.commit();
```

Dans le cas d'une insertion :

Aller sur un emplacement vide permettant d'insérer un nouvel enregistrement : public void **moveToInsertRow**();

Insérer dans la base de données l'enregistrement actuellement pointé : public void **insertRow**();

Exemple; (String sql2 = "select nom, prenom from employes";)

```
Statement stm3 =connexion.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
    ResultSet rst3=stm3.executeQuery(sql2);
    rst3.moveToInsertRow();
    rst3.updateString("nom", "Samuel");
    rst3.updateString("prenom", "Yacoub");
    rst3.insertRow();
    connexion.commit();
```

Méthode de déplacement dans le ResultSet

Pour se déplacer à l'intérieur du ResultSet, on utilisera la méthode **next()** pour le parcours avant et la méthode **previous()** pour le parcours inverse. Les autres méthodes sont données dans le tableau suivant.

Méthode	**Rôle**
boolean isBeforeFirst()	booléen qui indique si la position courante du curseur se trouve avant la première ligne
boolean isAfterLast()	booléen qui indique si la position courante du curseur se trouve après la dernière ligne
boolean isFirst()	booléen qui indique si le curseur est positionné sur la première ligne
boolean isLast()	booléen qui indique si le curseur est positionné sur la dernière ligne
boolean first()	déplacer le curseur sur la première ligne
boolean last()	déplacer le curseur sur la dernière ligne
boolean absolute()	déplace le curseur sur la ligne dont le numéro est fourni en paramètre à partir du début si il est positif et à partir de la fin si il est négatif. 1 déplace sur la première ligne, -1 sur la dernière, -2 sur l'avant dernière ...
boolean relative(int)	déplacer le curseur du nombre de lignes fourni en paramètre par rapport à la position courante du curseur. Le paramètre doit être négatif pour se déplacer vers le début et positif pur se déplacer vers la fin. Avant l'appel de cette méthode, il faut obligatoirement que le curseur soit positionné sur une ligne.
boolean previous()	déplacer le curseur sur la ligne précédente. Le booléen indique si la première occurrence est dépassée.

Boolean next()	déplacer le curseur sur la ligne suivante. Le booléen indique si la dernière occurrence est dépassée.
void afterLast()	déplacer le curseur après la dernière ligne
void beforeFirst()	déplacer le curseur avant la première ligne
int getRow()	renvoie le numéro de la ligne courante

Source :http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html

Méthodes pour obtenir les données et la structure

Méthode	**Rôle**
getInt(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme d'entier.
getInt(String)	retourne le contenu de la colonne dont le nom est passé en paramètre sous forme d'entier.
getFloat(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de nombre flottant.
getFloat(String)	
getDate(int)	retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de date.
getDate(String)	
next()	se déplace sur le prochain enregistrement : retourne false si la fin est atteinte
Close()	ferme le ResultSet
getMetaData()	retourne un objet ResultSetMetaData associé au ResultSet.

Type de données JDBC (correspondance SQL et JAVA)

Type SQL	Méthode ResultSet	Type Java
ARRAY	getArray	java.sql.Array
BIGINT	getLong	long
BINARY	getBytes	byte[]
BIT	getBoolean	boolean
BLOB	getBlob	java.sql.Blob
CHAR	getString	java.lang.String
CLOB	getClob	java.sql.Clob
DATE	getDate	java.sql.Date
DECIMAL	getBigDecimal	java.math.BigDecimal
DINSTINCT	getTypeDeBase	typeDeBase
DOUBLE	getDouble	double
FLOAT	getDouble	double
INTEGER	getInt	int
JAVA_OBJECT	(type)getObject	type
LONGVARBINARY	getBytes	byte[]
LONGVARCHAR	getString	java.lang.String
NUMERIC	getBigDecimal	java.math.BigDecimal
OTHER	getObject	java.lang.Object
REAL	getFloat	float
REF	getRef	java.sql.Ref
SMALLINT	getShort	short
STRUCT	(type)getObject	type

TIME	getTime	java.sql.Time
TIMESTAMP	getTimestamp	java.sql.Timestamp
TINYINT	getByte	byte
VARBINARY	getBytes	byte[]
VARCHAR	getString	java.lang.String

Utilisation du CallableStatement : JDBC et les procédures stockées

JDBC vous permet d'appeler une procédure stockée sur la base de données depuis une application écrite en Java. La première étape est de créer un objet CallableStatement. Comme avec les objets Statement et PreparedStatement, ceci est fait avec une connexion ouverte.

Un objet CallableStatement contient l'appel d'une procédure, il ne contient pas la procédure elle-même.

La classe CallableStatement est une classe dérivée de PrepareStatement, donc un objet CallableStatement peut avoir des paramètres d'entrées tout comme l'objet PreparedStatement. En plus, un objet CallableStatement peut avoir des paramètres de sorties ou des paramètres qui sont fait pour l'entrée et la sortie.

Syntaxe :

```
CallableStatement  nomStatement  =connexion.prepareCall("{ call NomProcedures (?,?,?....) }",[type de parcours resultset], [Type de resultset]);
```

Les points d'interrogation représentent les paramètres de la procédure

Exemple:

```
CallableStatement stm2 =connexion.prepareCall("{ call GestionEmployes.LISTE(?) }", ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

Lorsque la procédure à appeler retourne des paramètres (c'est-à-dire une fonction) la syntaxe à utiliser est la suivante :

```
CallableStatement nomStatement  =connexion.prepareCall("{ ? = call Nomfonction(?,?...) }")
```

Exemple

```
CallableStatement stm2 =connexion.prepareCall("{ ? = call GestionEmployes.lISTER(?) }",ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

Lorsque la procédure n'a pas de paramètres en entrée, mais qu'elle retourne quelque chose, la syntaxe est tout simplement : (fonction)

```
CallableStatement nomStatement  =connexion.prepareCall("{ ? = call Nomprocedure }")
```

Exécution du CallableStatement

Lorsque le statement appelle une procédure stockée qui contient une requête et produit un resultset, la méthode utilisée est executeQuery () Si la procédure contient une mise à jour ou une des instructions DDL, la méthode executeUpdate() aurait été utilisée. Comme c'est parfois le cas, une procédure stockée contient plus d'une instruction SQL, qui pourrait produire plus d'un résultset, plus d'une mise à jour, ou une combinaison de resultset et de mise à jour. Dans ce cas, lorsqu'il y a de multiples résultats, la méthode **execute()** est utilisée pour exécuter un CallableStatement.

Les paramètres de la procédure ou de la fonction sont en IN.

Lorsque les paramètres de la procédure ou de la fonction sont en IN, pour les passer à la procédure, on utilisera le CallableStatement.

- [Objet CallableStatement].setString([index],[objet String]);
- [Objet CallableStatement].setBoolean([index],[valeur]);
- [Objet CallableStatement].setInt([index],[valeur]);
- [Objet CallableStatement].setFloat([index],[valeur]);
- Etc….

Utilisation d'une procédure stockée avec des paramètres OUT.

Lorsque la procédure a des paramètres en OUT, il est important d'enregistrer les paramètres de sortie en fonction **du type du SGBD** et de son index, et ce avec la méthode **registerOutParameter** de l'objet CallableStatement. Il faut alors inclure oracle.jdbc.OracleTypes;

```
import oracle.jdbc.OracleTypes;
```

Exemple :

Vous avez un package empgest qui contient une procédure Insertion (tous les paramètres en IN, une fonction qui retourne un REF CURSOR et une procédure qui affiche un REF CURSOR (OUT).

```
CREATE OR REPLACE PACKAGE empgest as

TYPE requete is REF CURSOR;

PROCEDURE  insertion(

        pnum in employesclg.numemp%type,

        pnom in employesclg.nomemp%type,

        Ppren IN employesclg.prenomemp%type

                );

FUNCTION  lister (pcodedep in employesclg.codedep%type) return requete;

procedure afficheremp (pcodedep in employesclg.codedep%type, res out requete);

END empgest;
```

```
CREATE OR REPLACE PACKAGE BODY empgest AS
 FUNCTION  lister (pcodedep in employesclg.codedep%type) return requete as
  resultat requete;
 BEGIN
  OPEN RESULTAT FOR SELECT * FROM EMPLOYESCLG where codedep =pcodedep;
    RETURN resultat;
 END lister;


PROCEDURE afficheremp (pcodedep in employesclg.codedep%type, res out requete) as
   BEGIN
   OPEN RES FOR SELECT * FROM EMPLOYESCLG where codedep =pcodedep;
   END afficheremp;


PROCEDURE insertion(
            pnum in employesclg.numemp%type,
              pnom in employesclg.nomemp%type,
              Ppren IN employesclg.prenomemp%type
                                ) AS
BEGIN
INSERT INTO employesclg(numemp,nomemp,prenomemp) values (pnum,pnom,ppren);
COMMIT;
END insertion;
END empgest;
```

```
import java.sql.*;
import oracle.jdbc.OracleTypes;
import oracle.jdbc.pool.*;

public class Jdbc1 {

        public static void main(String[] args) {
        String user ="user1";
        String  mpasse ="user1";
        String bd=
"jdbc:oracle:thin:@205.233.344.351:1521:orcl";
        Connection conn = null;

            try
            {
             OracleDataSource ods = new OracleDataSource();
             ods.setURL(bd);
             ods.setUser(user);
             ods.setPassword(mpasse);
             conn = ods.getConnection();
             System.out.println("vous êtes connectés");


// ---------------------Insertion, tous les paramètres sont
en IN
                    try {
               CallableStatement Callins =
conn.prepareCall(" { call empgest.insertion (?,?,?)}");
                        Callins.setInt(1, 124);
                        Callins.setString(2, "PATOCHE");
                        Callins.setString(3,"ALAIN");
                        Callins.executeUpdate();
                        Callins.clearParameters();
                        Callins.close();
                        System.out.println("insertion");
                        }
                        catch(SQLException ins)
                            {
                     System.out.println(ins.getMessage());
                            }
//-------------------------------------------------------
//
```

```
 ------------------------fonction lister ------
// La fonction  a son premier paramètre en RETURN (out).Ce
paramètre est de type CURSOR.
                // un deuxième paramètre en IN
                    try {
                   CallableStatement Callist =
conn.prepareCall(" { ?= call empgest.lister(?)}");
                    Callist.setString(2,"INF");
  Callist.registerOutParameter(1,OracleTypes.CURSOR);
                    Callist.execute();
                   ResultSet rstlist = (ResultSet)
                   Callist.getObject(1);

                     while(rstlist.next())
                       {
                    String nomemp = rstlist.getString(2);
                    String prenomemp =rstlist.getString(3);
                    System.out.println(nomemp + prenomemp);
                        }
                    Callist.clearParameters();
                    Callist.close();
                    rstlist.close();
                    System.out.println("affichage");
                    }
                  catch(SQLException list)
                   {
              System.out.println(list.getMessage());
                    }
//-------------------------------------------------
//--------------------Procedure afficheremp -
 // La procédure a son premier paramètre en IN et
  // un deuxième paramètre en OUT. Ce paramètre est de type
CURSOR.
                try {
      CallableStatement Callaff = conn.prepareCall(" { call
empgest.afficheremp(?,?)}");
                    Callaff .setString(1, "RSH");
Callaff .registerOutParameter(2, OracleTypes.CURSOR);
                    Callaff .execute();
                    ResultSet rstaff = (ResultSet)
                    Callaff .getObject(2);
                     while(rstaff.next())
                    {
                     String nomemp = rstaff.getString(2);
                    String prenomemp =rstaff.getString(3);
                    System.out.println(nomemp + prenomemp);
                    }
```

```
                    Callaff .clearParameters();
                    Callaff .close();
                    rstaff.close();
                    System.out.println("affichage");
                   }
                catch(SQLException aff)
                {
               System.out.println(aff.getMessage());
                }
//---------------------------
            }
            catch(SQLException se)
            {
                System.out.println(se);            }

      finally
            {
                try
                {
                    if (conn!=null)
                        conn.close();
              System.out.println("connexion fermée");
                }
                catch(SQLException se){}
            }
    }
}
```

Explications :

- ✓ La procédure en insertion n'a que des paramètres en IN, il n'est donc pas nécessaire d'enregistrer les paramètres en OUT.
- ✓ Le CallableStatement contient l'appel d'une procédure INSERTION, il ne contient pas la procédure elle-même.
- ✓ L'interface CallableStatement hérite de l'interface PreparedStatement; toutes les méthodes du PreparedStatement s'appliquent au CallableStatement
- ✓ Lorsque les paramètres de la procédure appelée sont en OUT (pour une fonction c'est RETURN) alors il faut enregistrer les paramètres en question.
- ✓ Tous les paramètres en OUT doivent être enregistrés avant d'exécuter la procédure.

- ✓ L'enregistrement du paramètre utilise :

- Le type de données Oracle. Il faut alors inclure le package `oracle.jdbc.OracleTypes`
- la méthode `registerOutParameter` du CallableStatement. Cette méthode prend comme paramètres l'index de colonne du paramètre à enregistrer et le type SQL du paramètre en question.

```
void registerOutParameter(int parameterIndex,
                          int sqlType)
                           throws SQLException
```

- ✓ la méthode `getObject(int)` du callableStatement permet d'obtenir le résultat contenu dans la paramètre en OUT.
- ✓ lorsque le paramètre en OUT est un CURSOR (contient plusieurs enregistrements, il est nécessaire de faire un **cast** en **Resultset.**

```
ResultSet rst =(ResultSet)CallableStatement.getObject(1);
```

Autres Exemples

Utilisation d'une procédure avec deux paramètres en OUT et un paramètre en IN :

```
CREATE OR REPLACE PROCEDURE UNEMPLOYE (PNUM IN NUMBER,  PNOM OUT VARCHAR2,

PSALAIRE OUT NUMBER) AS

BEGIN

SELECT NOMEMP,SALAIREEMP INTO PNOM,PSALAIRE FROM EMPLOYESCLG WHERE
NUMEMP=PNUM;

END UNEMPLOYE;
```

```
try {
    CallableStatement emp = conn.prepareCall(" { call
UNEMPLOYE(?,?,?)}");
    emp.setInt(1,7);
    emp .registerOutParameter(2, OracleTypes.VARCHAR);
    emp .registerOutParameter(3, OracleTypes.FLOAT);
    emp .execute();
    System.out.println(" voici l'employes");
    String nom = (String)emp.getObject(2);
    float salaire = emp.getFloat(3);
    System.out.println(nom + salaire);
    emp.clearParameters();
    emp.close();
 }
 catch(SQLException em)
 {
    System.out.println(em.getMessage());
 }
```

Dans l'exemple précédent, La procédure UnEmploye a deux paramètres en OUT et un paramètre en IN.

- ✓ Le paramètre en IN est le premier. Pas besoin de l'enregistrer.
- ✓ Les deux paramètres en OUT, il faudra les enregistrer dans l'ordre et dans leur types SQL.
- ✓ une fois les paramètres enregistrés, on fait appel à la méthode Execute() de CallableStatement.

Appel d'une fonction (sans paramètre) qui retourne un simple int

La fonction dans le package :

```
FUNCTION TOTAL RETURN NUMBER;
```

Détail de la fonction dans le Package Body

```
FUNCTION  TOTAL RETURN NUMBER AS
  TOTALEMP NUMBER;
  BEGIN
    SELECT COUNT(*)  INTO TOTALEMP FROM EMPLOYES;
  RETURN TOTALEMP;
  END;
```

Appel de la fonction avec JDBC

```
CallableStatement stm3 =connexion.prepareCall("{ ? = call GestionEmployes.total() }");

stm3.registerOutParameter(1, OracleTypes.INTEGER);

stm3.execute();

int nbTotal =stm3.getInt(1);

System.out.println("nombre total employes est " + " " + nbTotal);
```

Appel d'une fonction qui retourne plus d'un enregistrement (retourne une variable de type curseur)

Fonction dans le package :

```
TYPE EMPENR IS REF CURSOR;

FUNCTION LISTER (codedepartement char )return empenr;
```

Détail de la fonction dans le Package Body

```
FUNCTION  LISTER (codedepartement char) return EMPENR AS

   sortie EMPENR;

   BEGIN

   OPEN sortie FOR SELECT nom, prenom FROM employes WHERE  codedep =
codedepartement;

   RETURN  sortie;

   END;
```

Appel de la fonction avec JDBC

```
CallableStatement stm2 =connexion.prepareCall("{ ? = call GestionEmployes.lISTER(?)
}");

stm2.registerOutParameter(1,OracleTypes.CURSOR);

stm2.setString(2, "inf");

stm2.execute(); //execution de la fonction

// Caster le paramètre de retour en ResultSet

ResultSet rest = (ResultSet) stm2.getObject(1);
```

```
        while (rest.next()) // affichage des résultat
        {
           String NOMR = rest.getString("nom");
           String PRENOMR = rest.getString("prenom");
           System.out.print(NOMR + " "  + PRENOMR);
        }
```

Appel d'une procedure stockée avec le pramètre OUT de type curseur

La procédure dans le package :

```
TYPE EMPENR IS REF CURSOR;
procedure Liste(requete2 out empenr);
```

Détail de la procédure dans Package Body

```
PROCEDURE  Liste(requete2 OUT empenr) AS
    BEGIN
  OPEN requete2  FOR SELECT NOM,PRENOM FROM EMPLOYES;
   END;
```

Appel de la fonction avec JDBC (se connecter en premier)

```
CallableStatement stm2 =connexion.prepareCall("{ call GestionEmployes.LISTE(?)
}",ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
stm2.registerOutParameter(1,OracleTypes.CURSOR);
stm2.execute(); //execution de la procédure
// Caster le paramètre de retour en ResultSet
ResultSet rest = (ResultSet) stm2.getObject(1);
        while (rest.next())
        {
           String NOMR = rest.getString("nom");
```

```
            String PRENOMR = rest.getString("prenom");
            System.out.print(NOMR + " "  + PRENOMR);
                    System.out.println();
        }
```

Notion de transactions avec JDBC

Le traitement de transactions est une condition obligatoire pour toutes les applications qui doivent garantir la cohérence de leurs données permanentes. Le pilote JDBC d'Oracle, travaille en mode Autocommit. Il est possible de changer ce mode grâce à la méthode SetAutocomite et un boolean (true ou false) de l'objet connection

Exemple :

```
connexion.setAutoCommit(false);
PreparedStatement stmupdate = connexion.prepareStatement(requete3);
stmupdate.setString(1, "Martin");
stmupdate.setString(2, "Alpha");
stmupdate.executeUpdate();
connexion.commit();
```

Pour annuler une transaction, il faut utiliser la méthode rollback() de l'objet connection
connexion.rollback();

Parfois, il est souhaitable de faire un rollback() jusqu'à un point de sauvegarde : savepoint :

Exemple :

```
String sqlup ="update employesclg set salaireemp = ? where numemp
=?";

    PreparedStatement stmupdate = conn.prepareStatement(sqlup);

                      stmupdate.setFloat(1,200);

                      stmupdate.setInt(2,2 );

                      stmupdate.executeUpdate();

                      stmupdate.clearParameters();

                      conn.setAutoCommit(false);

                      Savepoint save1 = conn.setSavepoint();

                      stmupdate.setFloat(1,10);

                      stmupdate.setInt(2,3 );

                      stmupdate.executeUpdate();

                      conn.rollback(save1);

                      stmupdate.clearParameters();

                      conn.commit();

                    stmupdate.close();
```

Extraire les clés générées automatiquement :

Certaines tables, utilisent des séquences pour générer automatiquement les clés primaire (ou autre) lors des insertions. Il est possible de récupérer ces clés grâce à la méthode getGeneratedKeys() de l'objet Statement ou du PreparedStatement. il faudra alors indiquer au Statement (Ou au PreparedStatement) la colonne qui utilise les clés générées.

Exemple

```
String sqli = "insert into employesclg (numemp,nomemp
,salaireemp) values(SEQ1.nextval,?,?)";

ResultSet rst = null;

try {
```

```
    String colonne[] = {"numemp"};
  // on indique la colonne qui utilise la séquence.
    PreparedStatement stm= conn.prepareStatement(sqli,colonne);
           int i =0;
           while (i<=10)
               {
                 stm.setString(1, "'alloqqq'");
                stm.setFloat(2,21);
               stm.executeUpdate();
               i++;
               rst = stm.getGeneratedKeys();
                           while( rst.next())
                              {
                          System.out.println(rst.getInt(1));
                              }
                }
    rst.close();
   stm.close();
  }
               catch(SQLException seup)
          {
              System.out.println(seup.getMessage());
          }
```

Limitations :

On ne peut accéder aux informations du ResultSet obtenu que par l'index de colonne.

Autres informations du Resultset :

Le ResultSet présente une interface qui permet d'avoir de l'information sur la structure des données qu'il contient. Cette interface est : **ResultSetMetaData**

- Obtenir le "metadata" avec la méthode **getMetaData() du resultSet**

Exemple:

```
ResultSetMetaData colonnes = rst.getMetaData();
```

- Obtenir le nombre de colonnes d'un objet de type ResultSet: avec la méthode **getColumnCount();**
  Exemple :

```
int nbcolonne = colonnes.getColumnCount();
```

- **obtenir le nom des colonnes d'indice connu.**

getColumnName(int [indice colonne]);

Exemple

```
colonnes.getColumnName(2)
```

- **obtenir le type d'une colonne avec la méthode getColumnTypeName(int [indice colonne]);**

```
colonnes.getColumnTypeName(3)
```

Exemple :

```
String sql4 ="SELECT * FROM EMPLOYESBIDON";
Statement stm = null;
ResultSet rst = null;
boolean nonvide;
Connection conn = null;

// après la connexion
try {

Statement stmx = conn.prepareStatement(sql3,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
       stm =conn.createStatement();
       nonvide = stm.execute(sql4);
             if(nonvide)
               {
                rst=stm.getResultSet();
                //Afficher les metadonnées ici
                ResultSetMetaData colonnes = rst.getMetaData();
                int nbcolonne = colonnes.getColumnCount();

              for(int i=1; i<=nbcolonne;i++)
                   {
               System.out.print(colonnes.getColumnName(i)+ "\t\t");
       System.out.print(colonnes.getColumnTypeName(i)+ "\t\t");
                    }
                            System.out.println();
                        //Afficher le resultSet
                            while (rst.next())
                            {
                                String nom1 = rst.getString(1);
                                String prn1 = rst.getString(2);
                                System.out.println( nom1+ prn1 );
                            }

                }
             }

             catch (SQLException sqlselect)  {
                 System.out.println(sqlselect.getMessage());
                                          }
             finally {
                      stm.close();
                      rst.close();
                     }
```

Bases de données et Web : Servlet – Brève définition

Une servlet est une application Java écrite sur le serveur et qui s'exécute sur le serveur lui-même et utilise donc ses ressources dont l'accès à une base de données. Le lancement d'une servlet se fait par la demande d'une page HTML.

Une servlet va se présenter comme une classe dérivée de la classe HttpServlet et fournie dans le package javax.servlet. La méthode doGet possède deux arguments :

Le premier de type HttpRequest, permet de récupérer les informations fournies lors de l'appel par la méthod.getParameter(NomDuParametre);

String getParameter(String name) :

Retourne la valeur d'un paramètre de la requête sous forme de chaîne, ou null. Le paramètre est contenu dans un formulaire et correspond à l'attribue name de celui-i

Le deuxième HttpResponse, créé automatiquement par le serveur permet d'identifier le client.

Transmission des paramètres à une servlet : Il suffit de récupérer les informations du client par le biais d'un formulaire. HTML et lancer le servlet via ce formulaire. Dans ce cas on peut choisir entre deux méthodes GET et POST.

Pour la méthode **POST**, La transmission les paramètres, ne sera plus visible dans le navigateur. Dans ce cas la servlet doit faire appel à une méthode différente : **doPost.**

Exemples : Les exemples suivants sont faits conjointement avec François Boileau

Exemple1 : Utilisation d'une servlet pour envoyer des résultats à la base de données : Cas d'une insertion

```
package Employes;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.sql.*;
```

```
// on accède au servlet avec l'URL "localhost:8080/employes/insertion"
@WebServlet( name = "Insertion", urlPatterns = {"/insertion"} )
public class Insertion extends HttpServlet
{
   // méthode appelée suite à une requête GET (barre d'adresse du
navigateur
  // ou hyperlien
   @Override
   protected void doGet(
         HttpServletRequest request, HttpServletResponse response )
         throws ServletException, IOException
   {
      // avertit le navigateur qu'il va recevoir du code HTML
      response.setContentType( "text/html;charset=UTF-8" );
      // obtention d'un flux pour écrire vers le navigateur
      PrintWriter out = response.getWriter();

      try
      {
//début du document HTML(pourrait être fait dans une méthodespécifique)
         out.println( "<!DOCTYPE html>" );
         out.println( "<html lang='fr'>" );
         out.println( "<head>" );
         out.println( "<meta charset='utf-8'/>" );
         out.println( "<title>Insertion</title>" );
         out.println( "</head>" );
         out.println( "<body>" );
         out.println( "<h1>Insertion d'un employé</h1>" );
         // formulaire de saisie pour insertion
         out.println( "<form action='insertion' method='post'>" );
         out.println( "<table>" );
```

```
         out.println( "<tr>" );
         out.println( "<td>Nom</td>" );
         out.println( "<td><input type='text' name='nom'/></td>" );
         out.println( "</tr>" );
         out.println( "<tr>" );
         out.println( "<td>Prénom</td>" );
         out.println( "<td><input type='text' name='prenom'/></td>" );
         out.println( "</tr>" );
         out.println( "<tr>" );
         out.println( "<td>Code de département</td>" );
         out.println( "<td><input type='text' name='dep'/></td>" );
         out.println( "</tr>" );
         out.println( "<tr>" );
         out.println( "<td colspan='2' style='text-align: center'>" +
               "<input type='submit' value='Insérer'/></td>" );
         out.println( "</tr>" );
         out.println( "</table>" );
         out.println( "</form>" );
         // barre de navigation
         out.println( "<a href='insertion'>Insertion</a>" + " | " +
" " + "<a href='recherche'>Recherche</a>" + " " + " | " + "<a
href='Afficher'>Lister</a>");
         out.println( "</body>" );
         out.println( "</html>" );
      }
      finally
      {
         out.close();
      }
   }
   // méthode appelée suite à une requête POST (envoi d'un formulaire)
```

```
   @Override
   protected void doPost(
         HttpServletRequest request, HttpServletResponse response )
         throws ServletException, IOException
   {
      // récupération des paramètres du formulaire avec la méthode
      String nomemploye = request.getParameter( "nom" );
      String prenomemploye = request.getParameter( "prenom" );
      String departement = request.getParameter( "dep" );
      // avertit le navigateur qu'il va recevoir du code HTML
      response.setContentType( "text/html;charset=UTF-8" );
      // obtention d'un flux pour écrire vers le navigateur
      PrintWriter out = response.getWriter();
      try
      {
         // début du document HTML (pourrait être fait dans une méthode
         out.println( "<!DOCTYPE html>" );
         out.println( "<html lang='fr'>" );
         out.println( "<head>" );
         out.println( "<meta charset='utf-8'/>" );
         out.println( "</head>" );
         out.println( "<body>" );
         out.println( "<h1>Insertion d'un employé</h1>" );
         // connexion à la base de données
         ConnexionOracle oradb = new ConnexionOracle();
         oradb.connecter();
String sqlins = "insert into employesclg
(numemp,nomemp,prenomemp,codedep) values (SEQ1.nextval,?,?,?)";
      try
         {
```

```
PreparedStatement stmins =oradb.getConnexion().prepareStatement( sqlins
);
            // on affecte les valeurs aux paramètres de la requête
            stmins.setString( 1, nomemploye);
            stmins.setString( 2, prenomemploye);
            stmins.setString( 3, departement );
            stmins.executeUpdate();
            stmins.close();
            oradb.deconnecter();
         }
         catch( SQLException se )
         {
            System.err.println( se.getMessage() );
         }
         // confirmation de l'insertion
         out.println( "<p>Vous avez ajouté l'employé suivant:</p>" );
         out.println( "<p>" + nomemploye + " " + prenomemploye + " (" +
departement + ")</p>" );
//barre de navigation
         out.println( "<a href='insertion'>Insertion</a>" + " | " +
" " + "<a href='recherche'>Recherche</a>" + " " + "|" + "<a
href='Afficher'>Lister</a>");
         out.println( "</body>" );
         out.println( "</html>" );
      }
      finally
      {
         out.close();
      }
   }
}
```

Exemple 2 : On affiche le contenu d'une table (aucun paramètre n'est fourni à la servlet

```
@WebServlet( name = "affiche", urlPatterns = {"/Afficher"} )
public class Afficher extends HttpServlet
{
   // méthode appelée par doGet() et doPost()
   protected void processRequest(
         HttpServletRequest request, HttpServletResponse response )
         throws ServletException, IOException
   {
      // avertit le navigateur qu'il va recevoir du code HTML
      response.setContentType( "text/html;charset=UTF-8" );
      // obtention d'un flux pour écrire vers le navigateur
      PrintWriter out = response.getWriter();
      // début du document HTML (pourrait être fait dans une méthode
      out.println( "<!DOCTYPE html>" );
      out.println( "<html lang='fr'>" );
      out.println( "<head>" );
      out.println( "<meta charset='utf-8'/>" );
      out.println( "<title>Recherche</title>" );
      out.println( "</head>" );
      out.println( "<body>" );
      out.println( "<h1>Afficher tous les  d'employés</h1>" );
      // connexion à la base de données
      ConnexionOracle oradb = new ConnexionOracle();
      oradb.connecter();
      String sql= "select  nomemp,prenomemp from employesclg ";
```

```
      try
      {
         Statement stm = oradb.getConnexion().createStatement();
         ResultSet rst = stm.executeQuery(sql);
//récupérer les nom de colonnes de la base de données. On les mets dans
un tableau.
          ResultSetMetaData colonnes = rst.getMetaData();
         int nbcolonne = colonnes.getColumnCount();
         out.println( "<table>" );
         out.println( "<tr>" );
         for(int i=1; i<=nbcolonne;i++)
                {
            out.println( "<td>" +colonnes.getColumnName(i) + "</td>");
                }
          out.println( "</tr>" );
         out.println( "</table>" );
//Afficher le contenu du resultSet, dans un autre tableau
         out.println( "<table border =1>" );
         while( rst.next() )
         {
            out.println( "<tr>" );
            String nom = rst.getString(1);
            String prenom = rst.getString(2);
              out.println( "<td>" + nom + "</td>" );
              out.println( "<td>" + prenom + "</td>" );
            out.println( "</tr>" );
           }
         out.println( "</table>" );
         stm.close();
         rst.close();
         oradb.deconnecter();
```

```
      }
      catch( SQLException se )
      {
         System.err.println( se );
      }
      // barre de navigation
   out.println( "<a href='insertion'>Insertion</a>" + " | " + " "
+"<a href='recherche'>Recherche</a>" + " " + " | " + "<a
href='Afficher'>Lister</a>");
      out.println( "</body>" );
      out.println( "</html>" );
   }

   // méthode appelée suite à une requête GET (barre d'adresse du
navigateur
   // ou hyperlien
   @Override
   protected void doGet( HttpServletRequest request,
HttpServletResponse response)
         throws ServletException, IOException
   {
      processRequest( request, response );
   }
   // méthode appelée suite à une requête POST (envoi d'un formulaire)
   @Override
   protected void doPost(HttpServletRequest request,
HttpServletResponse response)
         throws ServletException, IOException
   {
      processRequest( request, response );
   }
}
```

Exemple 3 : Cas d'une requête paramétrée : Le formulaire contient un paramètre, et la BD nous renvoi des résultat suite à la valeur du paramètre.

```
@WebServlet( name = "Recherche", urlPatterns = {"/recherche"} )
public class Recherche extends HttpServlet
{
   // méthode appelée par doGet() et doPost()
   protected void processRequest(
         HttpServletRequest request, HttpServletResponse response )
         throws ServletException, IOException
   {
      // avertit le navigateur qu'il va recevoir du code HTML
      response.setContentType( "text/html;charset=UTF-8" );
      // obtention d'un flux pour écrire vers le navigateur
      PrintWriter out = response.getWriter();
      // début du document HTML (pourrait être fait dans une méthode
      out.println( "<!DOCTYPE html>" );
      out.println( "<html lang='fr'>" );
      out.println( "<head>" );
      out.println( "<meta charset='utf-8'/>" );
      out.println( "<title>Recherche</title>" );
      out.println( "</head>" );
       out.println( "<body>" );
       out.println( "<h1>Recherche d'employés</h1>" );
       out.println ( "<form action='recherche' method='post'>" );
       out.println("<input type='text' name='dep'/>");
       out.println("<input type='submit' value='Chercher'/>");
        out.println( "</frorm>" );
      // connexion à la base de données
```

```
      ConnexionOracle oradb = new ConnexionOracle();

      oradb.connecter();

      String sql= "select  nomemp,prenomemp from employesclg where
codedep = ?";

      try

      {

         //récuperer le paramètre

        String dept = request.getParameter("dep");

  PreparedStatement stm = oradb.getConnexion().prepareStatement( sql );

          stm.setString(1,dept);

          ResultSet rst = stm.executeQuery();

         // parcours du ResultSet

         out.println( "<ol>" );

                  while( rst.next() )

                  {

                     String nom = rst.getString( "nomemp" );

                     String prenom = rst.getString( "prenomemp" );

               out.println( "<li>" + nom + ", " +  prenom + "</li>" );

                  }

         out.println( "</ol>" );

         stm.close();

         rst.close();

         oradb.deconnecter();

      }

      catch( SQLException se )

      {

         System.err.println( se );

      }

      // barre de navigation

    out.println( "<a href='insertion'>Insertion</a>" + " | " + " "
+  "<a href='recherche'>Recherche</a>" + " " + " | " + "<a
href='Afficher'>Lister</a>");
```

```
      out.println( "</body>" );

      out.println( "</html>" );

   }

   // méthode appelée suite à une requête GET (barre d'adresse du
navigateur   // ou hyperlien

   @Override

   protected void doGet( HttpServletRequest request,
HttpServletResponse response)

         throws ServletException, IOException

   {

      processRequest( request, response );

   }



   // méthode appelée suite à une requête POST (envoi d'un formulaire)

   @Override

   protected void doPost(HttpServletRequest request,
HttpServletResponse response)

         throws ServletException, IOException

   {

      processRequest( request, response );

   }

}
```

Compléments de cours : L'interface RowSet

Une interface qui hérite de l'interface ResultSet. Contrairement au ResultSet, le RowSet est updateble et scrollable. De plus il possède les propriétés d'un JavaBean

Le RowSet peut être utilisé en mode connecté ou non.

Il existe plusieurs types de RowSet: OracleJDBCRowSet, OracleCachedRowSet , OracleWebRowSet.

OracleJDBCRowSet : Il peut être utilisé de plusieurs façons :

1. En utilisant le constructeur par défaut de mise en œuvre de référence : la connexion est obtenue au moment de l'exécution de la commande.
    - La connexion est obtenue au moment de l'exécution de la commande, grâce aux méthodes SetUrl() SetUsername(), SetPassword ()du OracleJDBCRowSet
    - La requête est envoyée grâce à la méthode SetCommand(string SQL)
    - La Requête est exécutée grâce à la méthode Execute().

Exemple1 1

```
try

        {

             OracleJDBCRowSet rowset = new OracleJDBCRowSet();

             rowset.setUrl(bd);

             rowset.setUsername(user);

             rowset.setPassword(mpasse);

rowset.setCommand("SELECT numemp, nomemp, prenomemp FROM
employesclg");

            rowset.execute();

                  while (rowset.next())

                  {

            System.out.println("empno: " + rowset.getInt(1));

            System.out.println("ename: " + rowset.getString(2));
```

```
            System.out.println("Prenom: " + rowset.getString(3));
                  }
                  System.out.println("----Le troisieme---");
                  rowset.absolute(3);
                  System.out.println(rowset.getString(2));
      }
          rowset.close();
 catch(SQLException select)
            {
                System.out.println(select.getMessage());
            }
```

2. La connexion est obtenue comme d'habitude (par ods ou le DriverManager)
   - La connexion est faîte avec ods ou DriverManager . Elle est faîte avant la création de l'objet OracleJDBCRowSet
   - La requête est passée par la méthode SetCommand (string SQL)
   - La requête est exécutée par la méthode Excecute()

Exemple 2 :

```
try
        {
              OracleDataSource ods = new OracleDataSource();
                ods.setURL(bd);
                ods.setUser(user);
                ods.setPassword(mpasse);
                conn = ods.getConnection();
                try
                {
            OracleJDBCRowSet rowset = new OracleJDBCRowSet(conn);
```

```
     rowset.setCommand("SELECT numemp, nomemp, prenomemp FROM
employesclg where codedep = ?");

                rowset.setString(1, "INF");

                rowset.execute();

                rowset.absolute(3);

                System.out.println(rowset.getString(2));

                rowset.close();

                     }

                catch(SQLException list)

              {

                 System.out.println(list.getMessage());

              }

// suite : catch pour la connexion
```

Remarque :

Le fait que le rowset passe la commande, il se comporte donc comme un Statement ou un PreparedStatement. Le rowset peut passer également les valeurs des paramètre de la requête SQL passée par la méthode SetCommand();

Le CachedRowset, Il a les mêmes caractéristiques que le JDBCRowSet, et peut travailler en mode déconnecté (~ Data Set pour ADO.NET).. En plus de l'utiliser comme un JdbcRowSet on peut le peupler avec un ResultSet. Pour peupler un CachedRowSet on utilise la méthode populate(ResultSet). Le résultat obtenu est scrollable quel que soit le type de ResultSet.

Exemple 3

```
try
{
      OracleDataSource ods = new OracleDataSource();
        ods.setURL(bd);
        ods.setUser(user);
        ods.setPassword(mpasse);
        conn = ods.getConnection();

        try
        {
      Statement stm = conn.createStatement();
      rst=stm.executeQuery(sql);

   OracleCachedRowSet rowset = new OracleCachedRowSet();
          rowset.populate(rst);
          conn.close();
          while (rowset.next())
          {
            System.out.println( rowset.getString(1));
            System.out.println(rowset.getString(2));
                          }
          System.out.println("----Le troisieme---");
          rowset.absolute(3);
         System.out.println(rowset.getString(2));
        stm.close();
```

```
                rst.close();
                rowset.close();
                     }
                catch(SQLException rowsetex)
               {
                  System.out.println(rowsetex.getMessage());
                }
```

Exemple 4 : Le RésultSet est issus d'une procédure stockée

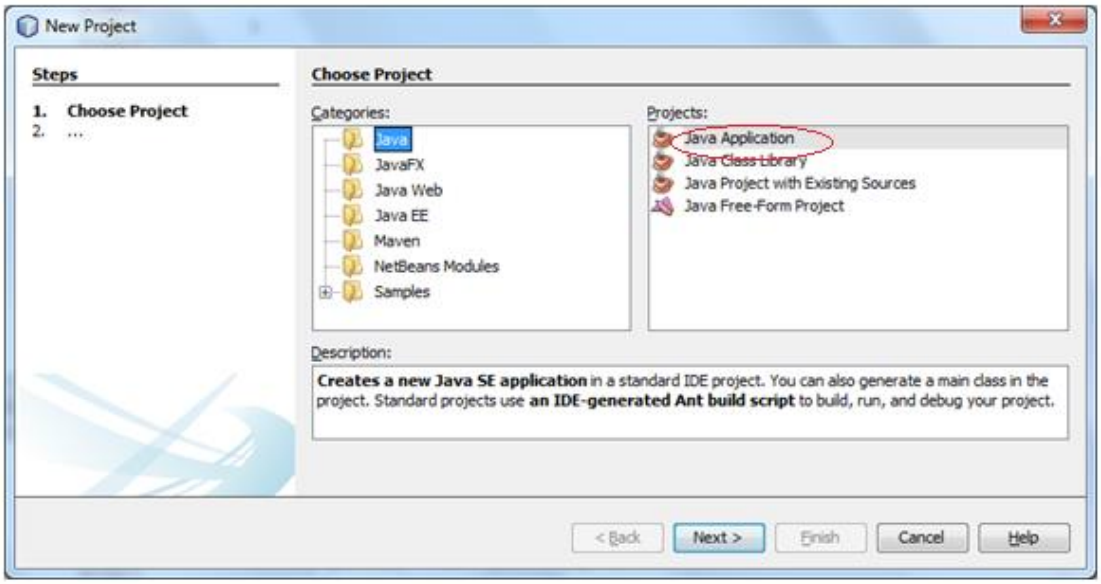
```
try {
            CallableStatement Callist = conn.prepareCall(" { ?=
call empgest.lister(?)}");
            Callist.setString(2,"INF");
            Callist.registerOutParameter(1, OracleTypes.CURSOR);
            Callist.execute();
            ResultSet rstlist = (ResultSet) Callist.getObject(1);
        OracleCachedRowSet orarowset = new OracleCachedRowSet ();
                orarowset.populate(rstlist);
                System.out.println("_____ du début___");
                 while(orarowset.next())
                   {
                         String nomemp = orarowset.getString(2);
                      String prenomemp =orarowset.getString(3);
                       System.out.println(nomemp + prenomemp);
                     }
                System.out.println("__De la fin_____");
```

```
orarowset.afterLast();
while(orarowset.previous())
  {
 String nomemp = orarowset.getString(2);
 String prenomemp =orarowset.getString(3);
 System.out.println(nomemp + prenomemp);
     }
  Callist.clearParameters();
  Callist.close();
  rstlist.close();
  orarowset.close();
  System.out.println("Liste complété ");
  }
catch(SQLException list)
 {
  System.out.println(list.getMessage());
 }
```

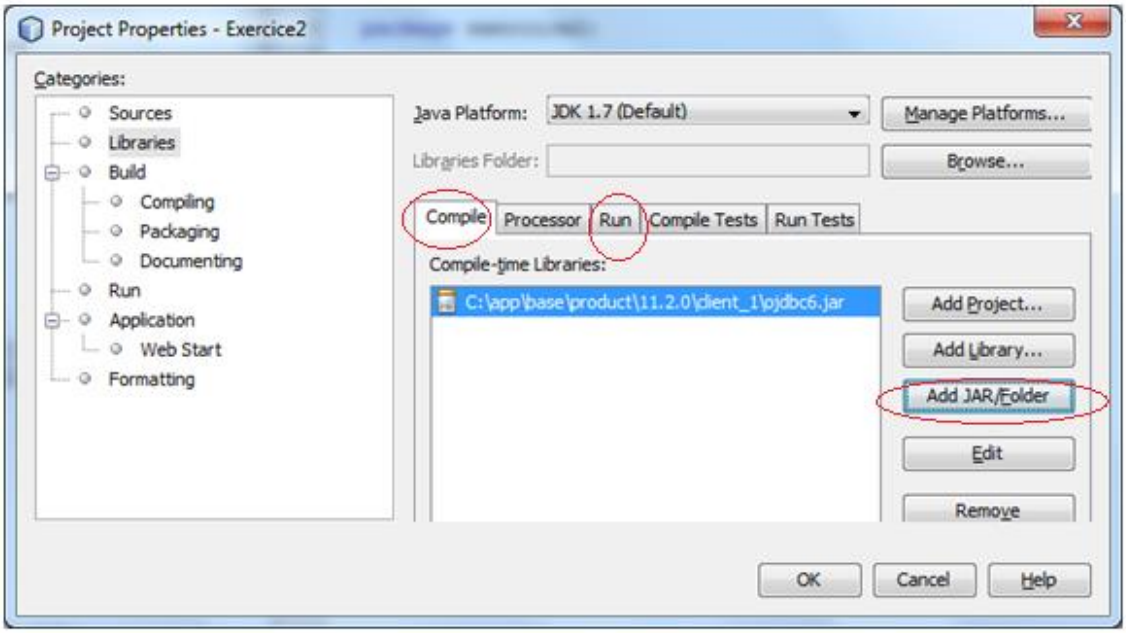
Annexe 1 : Comment démarrer un projet NetBeans utilisant une interface graphique?

Voici une description rapide du contenu de l'application qui vous a été envoyée.

1- Vous créez votre projet normalement (par le menu Fichier nouveau projet)

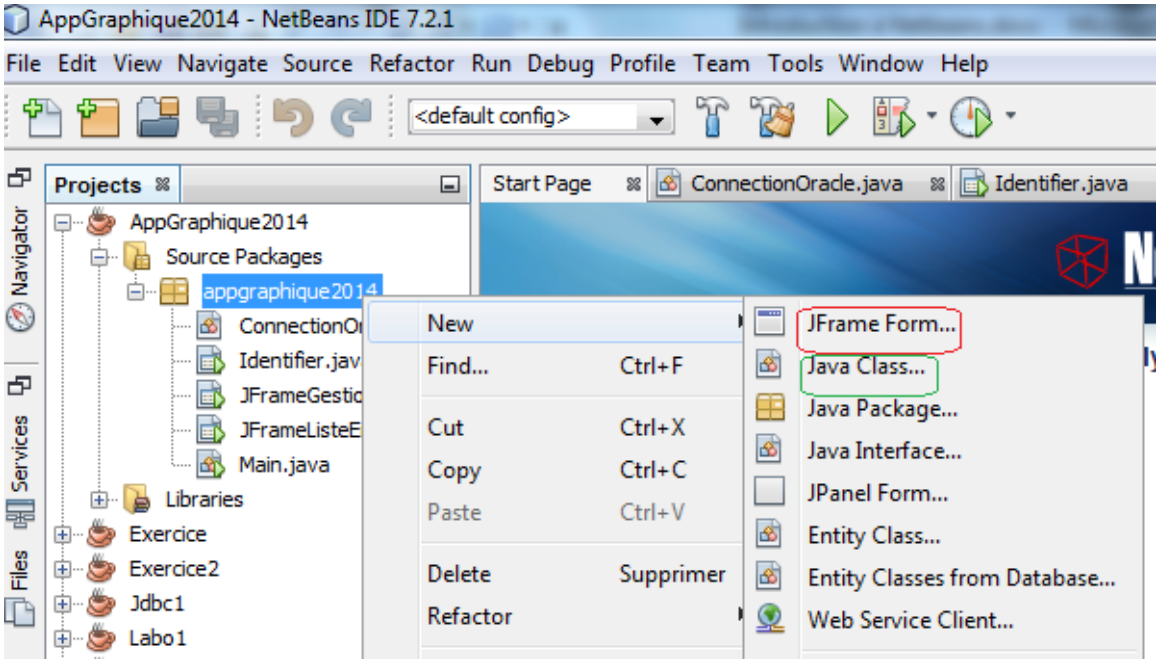


2- une fois crée, n'oubliez pas d'inclure les librairies .JAR (la version 14 de ojdbc)



3- Par la suite, nous allons créer les classes suivantes :

Une **classe ConnectionOracle** de type **Java Class**
Une classe **Main** de type **Java Class**
Une Classe **Identifier** de Type Jframe Form
Une Classe **JFrameGestion** de type JFrame Form
Une Class **JFrameLisEmp** de type jFrame Form



Rôle de la classe ConnectionOracle.

Cette classe a pour rôles :

- Charger le driver s'i y a lieu
- De définir les paramètres de connexion
- De définir deux méthodes connecter() et deconnecter()

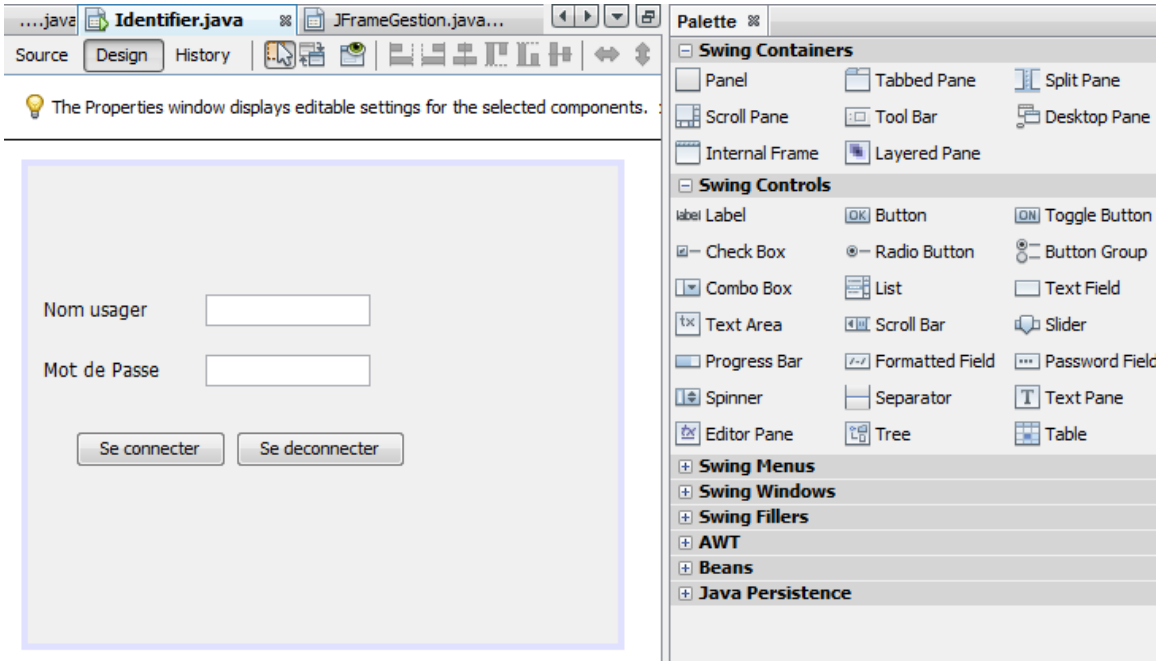
Cette classe doit utiliser les package : java.sql.* et import oracle.jdbc.pool.*;

```
import java.sql.*;

import oracle.jdbc.pool.*;
```

Rôle de la classe Identifier de type JFrameForm

En mode Design :

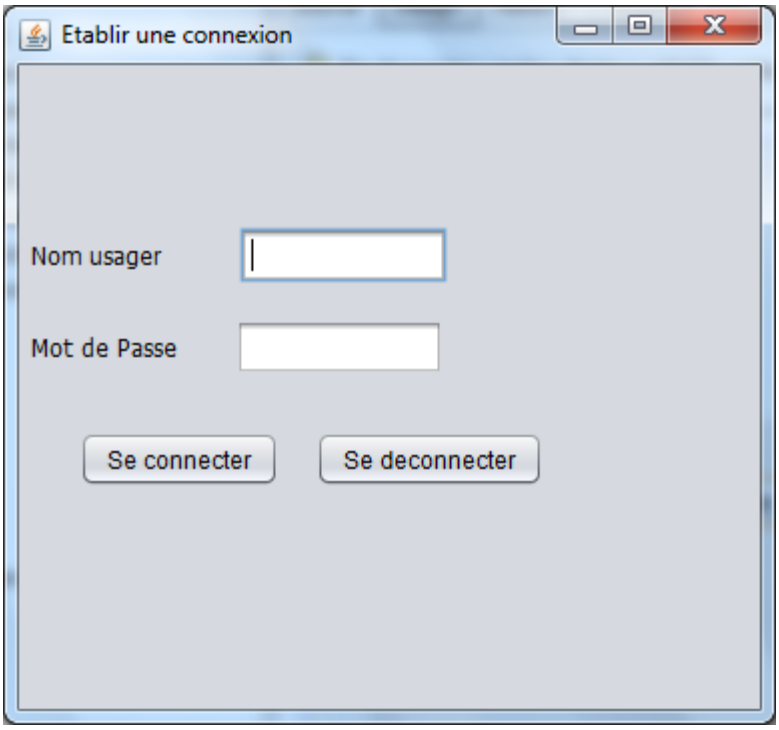
À Droite vous avez votre Jform et à gauche vous avez vos contrôles NetBeans.



En mode Source, vous devez importer les packages suivants :

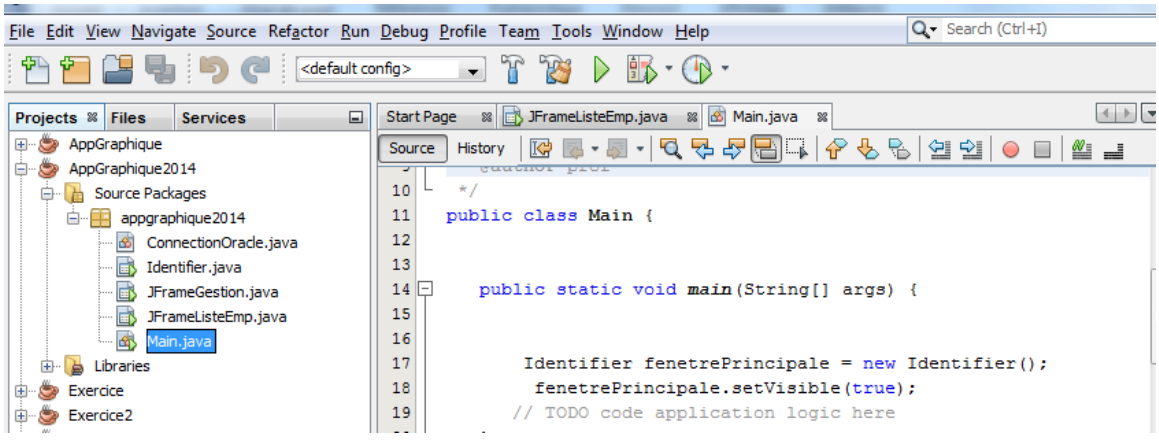
```
import javax.swing.*;  // pour les interface graphique

import java.awt.*;// pour les interface graphique (version antérieur)

import java.sql.*;
```

À travers cette classe on saisit le nom d'usager et le mot de passe pour obtenir une connexion à la base de données.



Rôle de la classe Main :

C'est cette Classe qui va appeler la classe Identifier. Voir code source dans la fenêtre suivante.

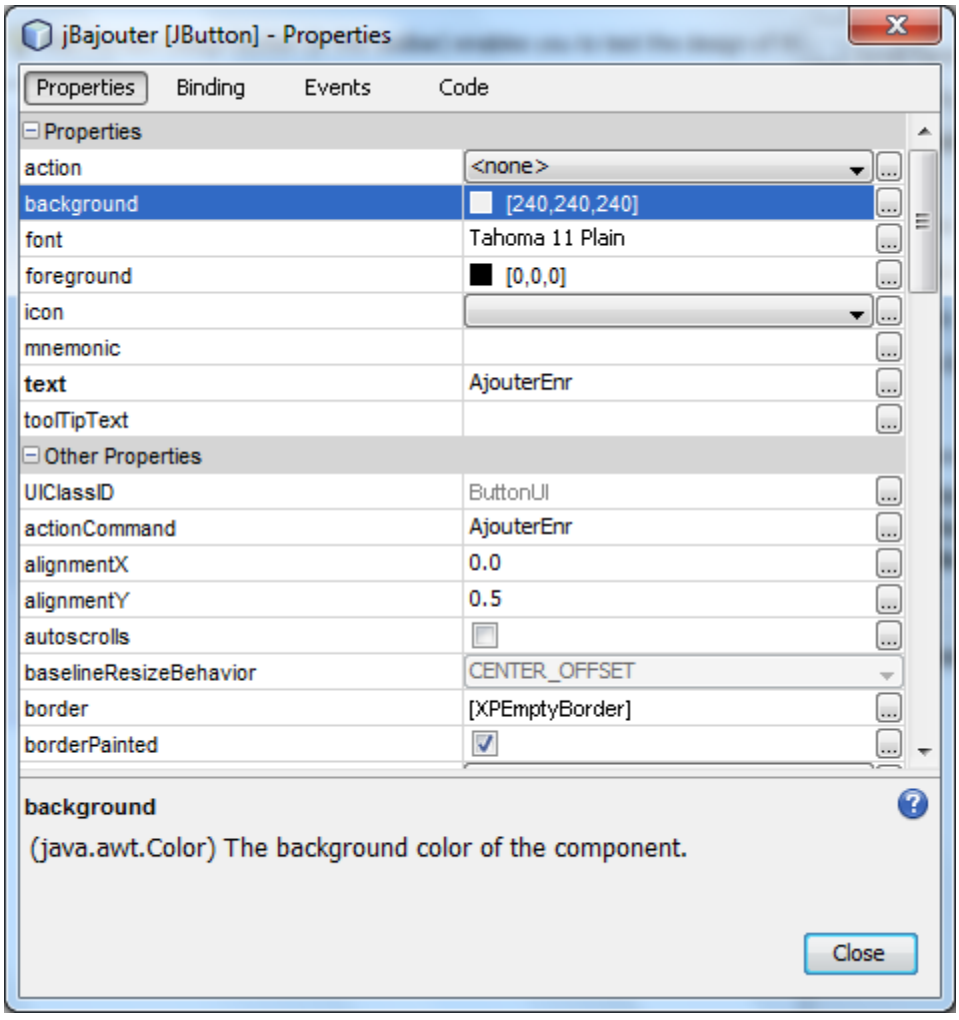


Rôle Classe JFrameGestion.

Cette classe appelée dès que la connexion est établie (par la classe identifier)

```
private void jBconnecterActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
        String n = jTextUser.getText();
        String m =jTextMPasse.getText();
        connBD.setConnection(n,m);
        connBD.connecter();
        // appel de la classe JFrameGestion (en utilisant un objet Connectio
        JFrameGestion fenetre = new JFrameGestion(connBD);
        fenetre.setVisible(true);

}
```

C'est au niveau de cette classe que nous avons décidé d'afficher et d'ajouter des enregistrements.



On accède aux propriétés des Contrôles graphiques par le bouton droit/propriétés (figure précedente).

Pour accéder au nom du contrôle, utiliser l'onglet CODE

IMPORTANT :

Dans les classes JFrame qui utilisent une connexion à la base de données qui a été définie dans une autre classe (ConnectionOrale), il est important de définir un objet (une variable) de type ConnectionOracle comme suit : ConnectionOracle connBD;

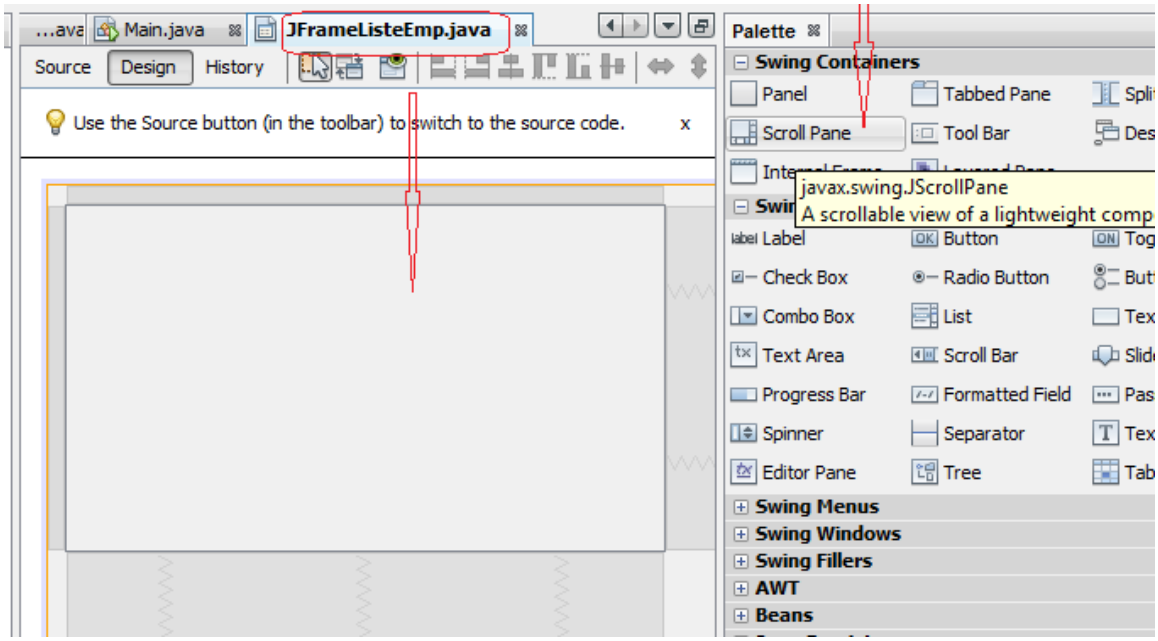
Lors de la définition d'un Statement ou d'u PreparedStatement ou un CallableStament, on utilise l'instruction suivante

PreparedStatement stminsert=connBD.getConnection().prepareStatement(sqlajout);

Au lieu de

PreparedStatement stminsert=connBD.prepareStatement(sqlajout);

Rôle de la classe JFrameListeEmp



On utilise un Jtable (Tableau de lignes et de colonnes) pour contenir

Toutes les lignes et toutes lec colonnes.un constructeur de Jtable est de la forme:

JTable(Object[][] rowData, Object[] columnNames)

On utilise le JscrollPann pour visulaiser le JTable.

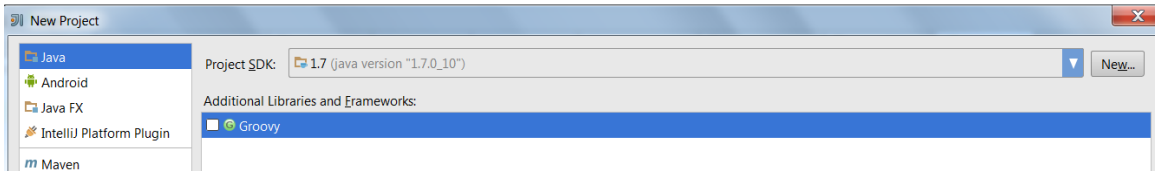
La méthode qui permet de faire lister tous les employés est **public void lister()**

Il ne faut pas formater le Jscroll Pane en mode Design. C'est le vecteur qui est définit par le code dans le Jtable qui va formater vos données. Le Jscroll Pane est vu comme une fenêtre de visualisation seulement.

Annexe 2 : Comment démarrer un projet Intellij Idea utilisant une interface graphique?

Intellij Idea est un éditeur Java qui permet de réaliser des interfaces graphiques en utilisant les librairies de swing. Voici une explication rapide de comment démarrer un projet Intellij Idea graphique et JDBC

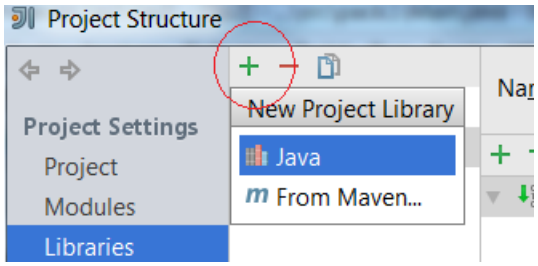
Nouveau projet.(avec une version du JDK (1.7 et plus).(C:\Program File\Java\jdk1.8.0.25)

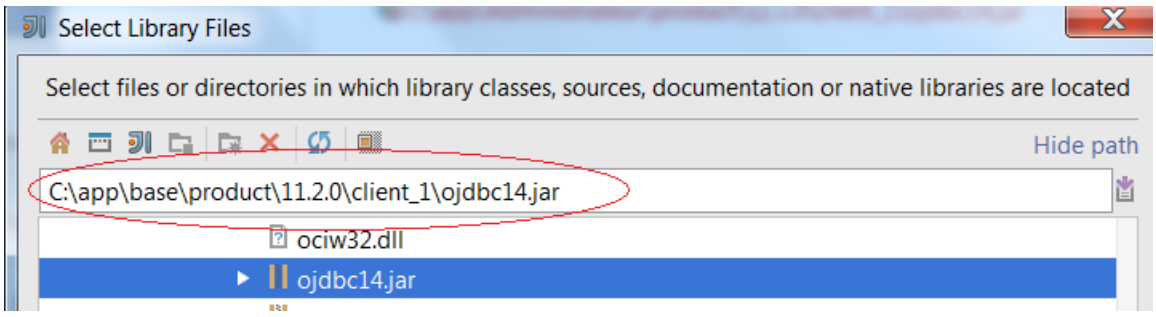


Comme c'est un projet JDBC, il faudra ajouter les classes **.JAR ojdbc14**.jar qui se trouve dans le dossier C:\app\base\product\11.2.0\client_1. Procéder comme suit :

Si vous n'avez pas la dernière version, vous pouvez la télécharger à l'adresse :
http://www.oracle.com/technetwork/apps-tech/jdbc-10201-088211.html

Par le menu Fichier/Project Structure/Libraire, puis ajouter.

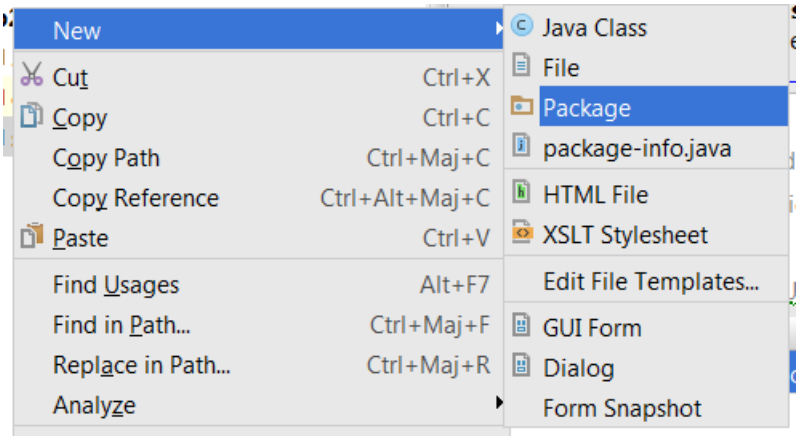




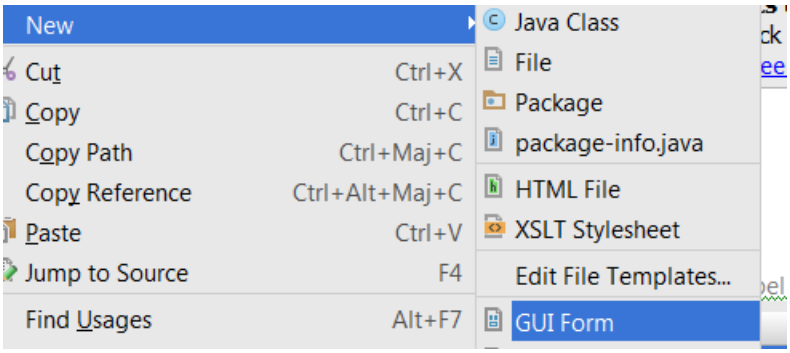
Maintenant que l'environnement de travail est bien configuré, alors on peut commencer

Il est également important de créer un **package** qui va contenir toutes les classes en rapport avec le projet.

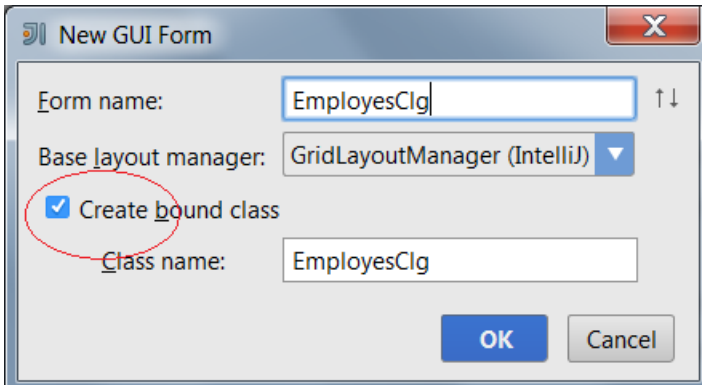
1. Bouton droit sur src, puis nouveau package. Donnez un nom significatif à votre package.



2. Puis bouton droit sur le nom **du package**, puis nouveau GUI Form



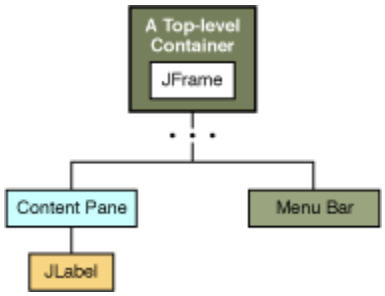
Ceci va vous permettre de créer deux classes



- La classe Java, contenant le code source de votre programme.
- Une classe de même nom avec l'extension **.form** Cette classe va contenir le code correspondant à la création et définition des contrôles swing (bouton, zone de text ..) associés à votre application.

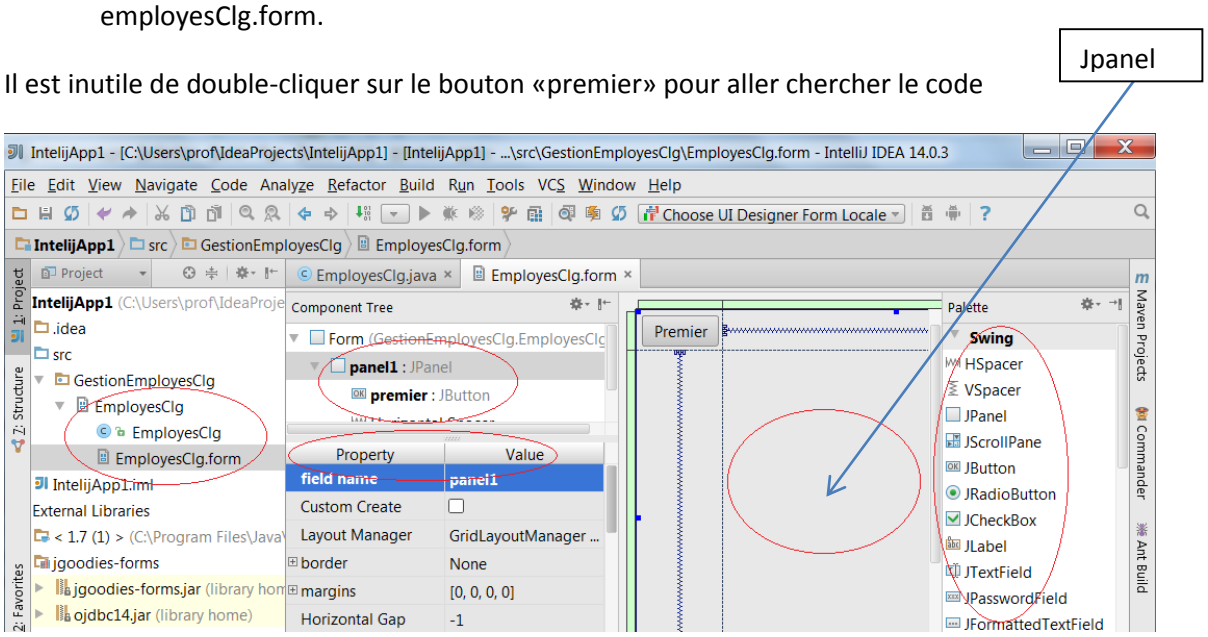
Après la création de la GUI form, vous obtiendrez l'espace de travail suivant :

- ✓ Votre espace de travail est le Jpanel dont le nom est panel1.
- ✓ C'est ce conteneur qui va contenir l'ensemble des autres contrôles : Jbutton, JTextFild, JLabel etc ...). Par la suite, votre Jframe, doit contenir votre Jpanel.



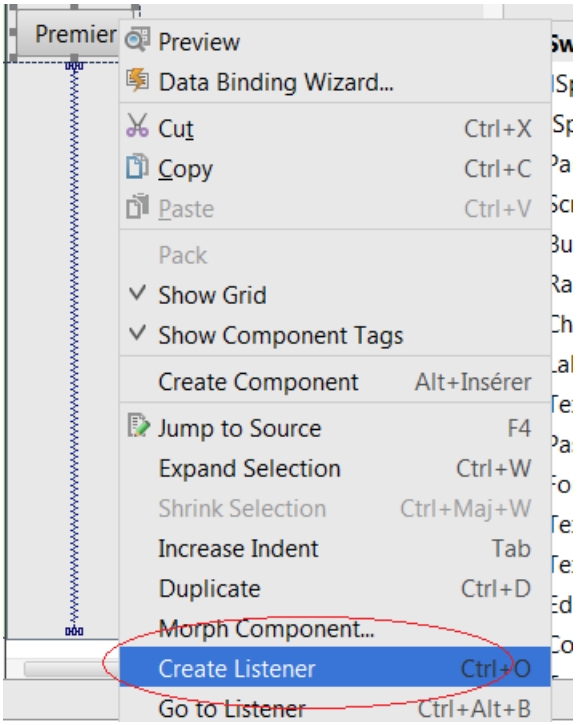
- ✓ Les contrôles swing ou la boite à outil est complétement à droite.
- ✓ Complétement à gauche vous remarquerez que vous avez deux classes : EmployesClg et employesClg.form.

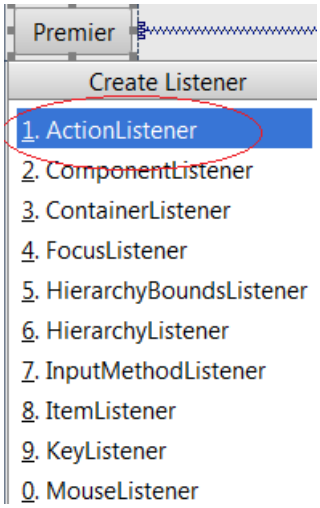
Il est inutile de double-cliquer sur le bouton «premier» pour aller chercher le code

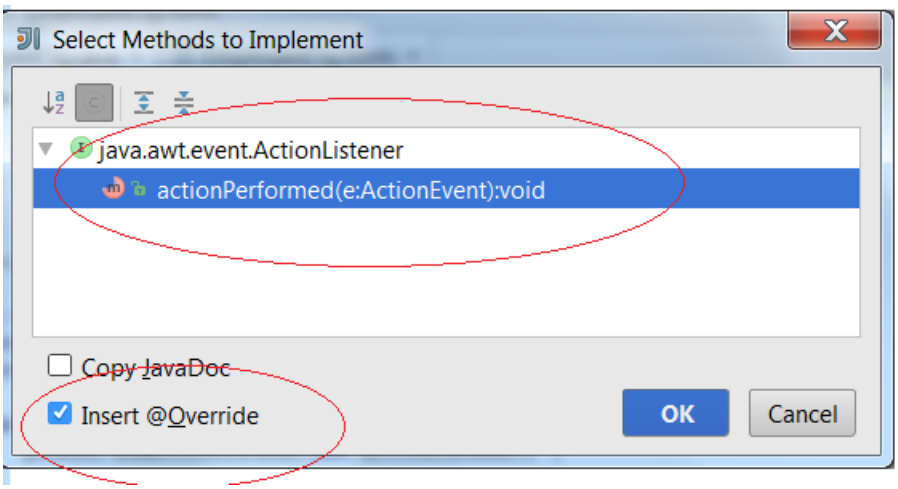


Quand on parle du développement graphique avec Java swing , on parle de Listener La méthode associer est ActionListener(). Pour accéder à cette méthode :

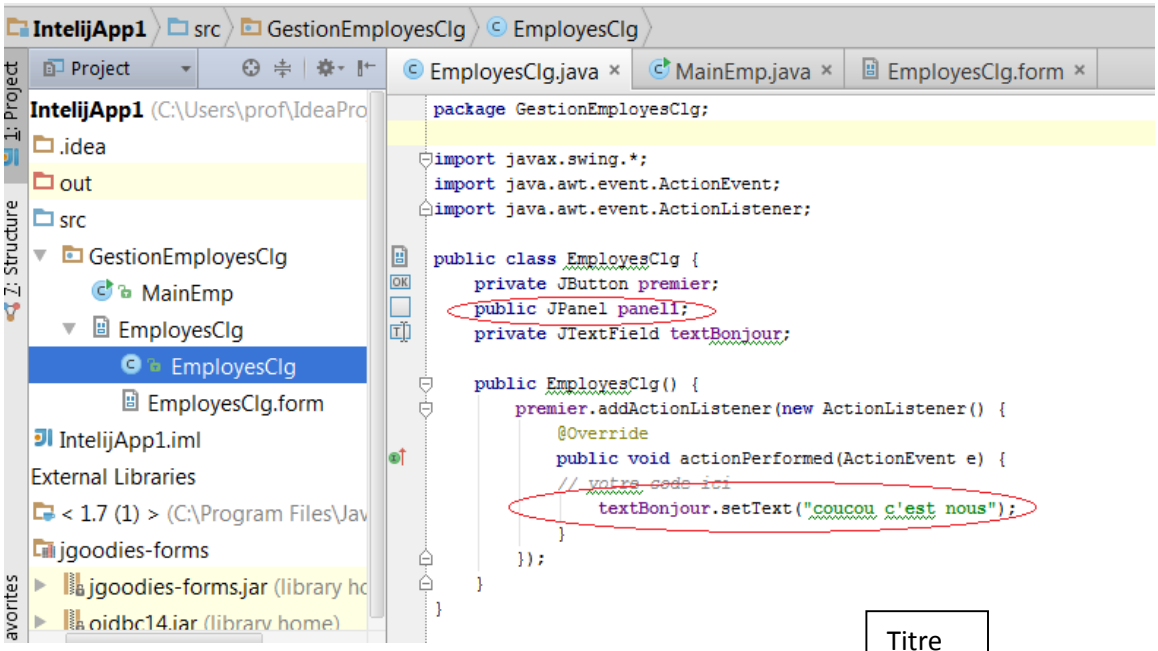
- Create Listener pour créer le code pour la première fois
- Go To Litener pour modifier le code déjà crée.



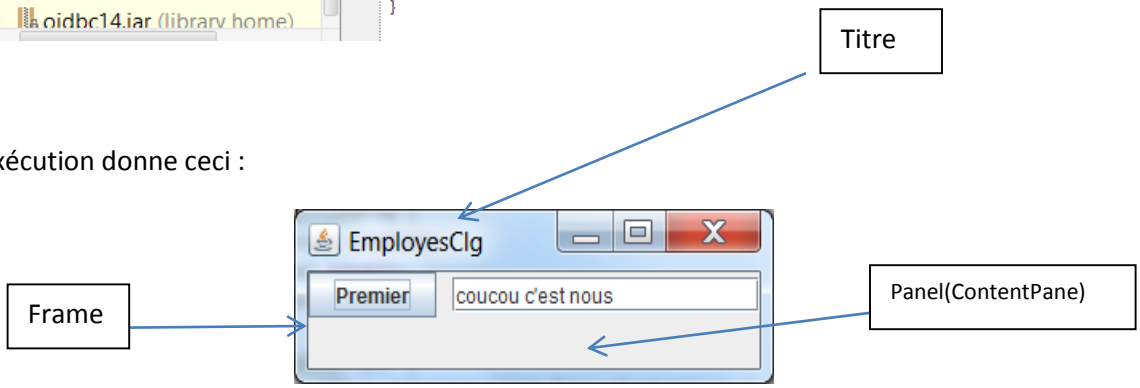




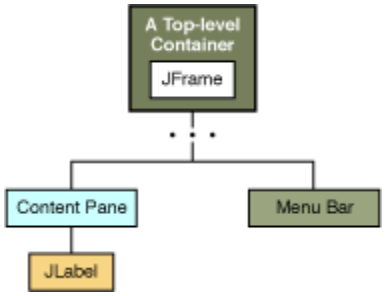
Voici ce que donne l'exemple dans la classe correspondant au code du Form



L'exécution donne ceci :

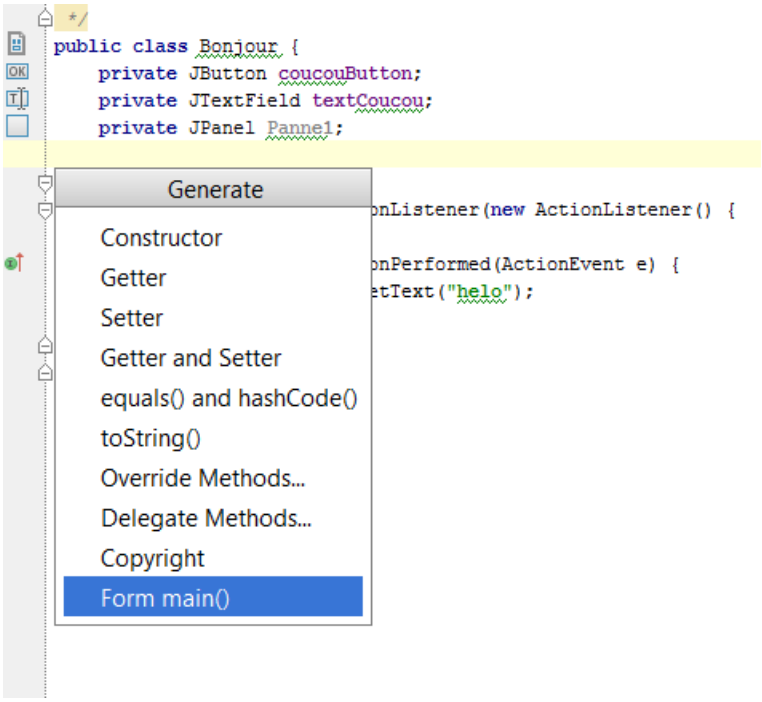


Panel(ContentPane) est contenu dans un Jframe.



Pour pouvoir exécuter le programme, il faudra lui inclure une classe main, deux options s'offrent à vous :

1- inclure une classe main dans le code source :
   a. Ouvrir la class du code source (bound class pour édition – modification)
   b. Presser ALT + Insert
   c. choisir Formmain()



Le code suivant va alors s'insérer :

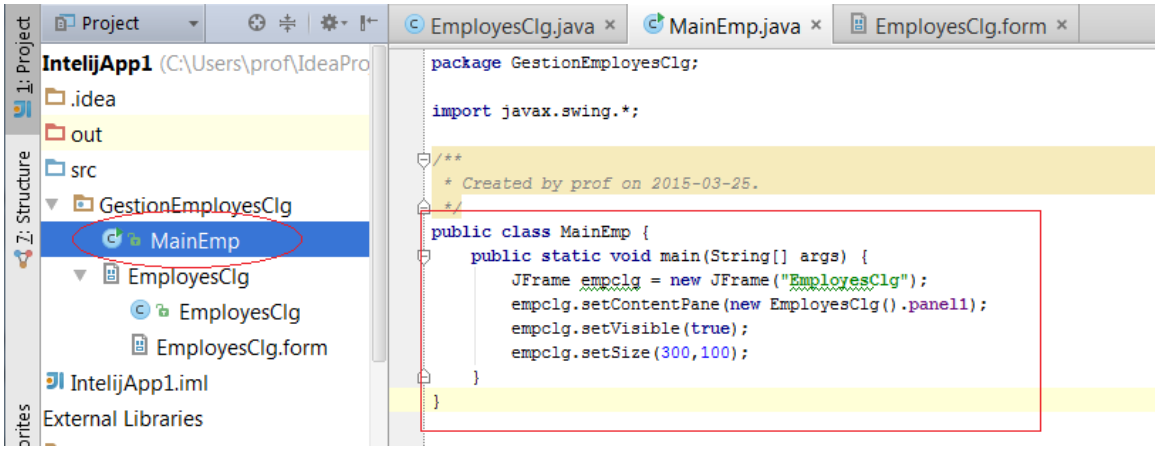
```
public static void main(String[] args) {
// on crée une fenêtre dont le titre est "Bonjour"
JFrame frame = new JFrame("Bonjour");
// on ajoute le contenu du Panne1
frame.setContentPane(new Bonjour().Pannel);
//la fenêtre se ferme quand on clique sur la croix rouge
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//on attribue la taille minimale au frame
frame.pack();
// on rend le frame visible
frame.setVisible(true);
```

Pour le frame.pack(), il est préférable de le remplacer par
frame.**setSize (int widht, int height)**

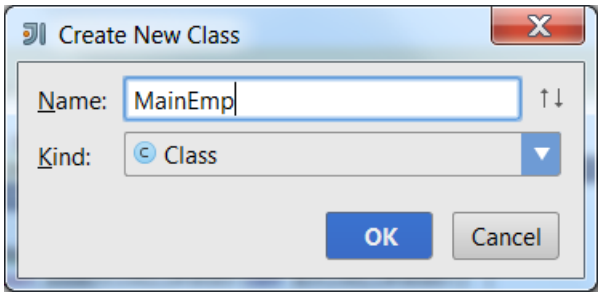
2- créer votre propre classe main, c'est ce qui est recommandé en suivant les étapes :
définir un Frame Java, dont le nom est empclg. et dont le titre est EmployesClg

1. affecter les propriétés du JPanel panel1 au JFrame
2. affiche le JFrame
3. affecte une dimension au frame avec la methode **setSize (int widht, int height)**

(Voir exemple)

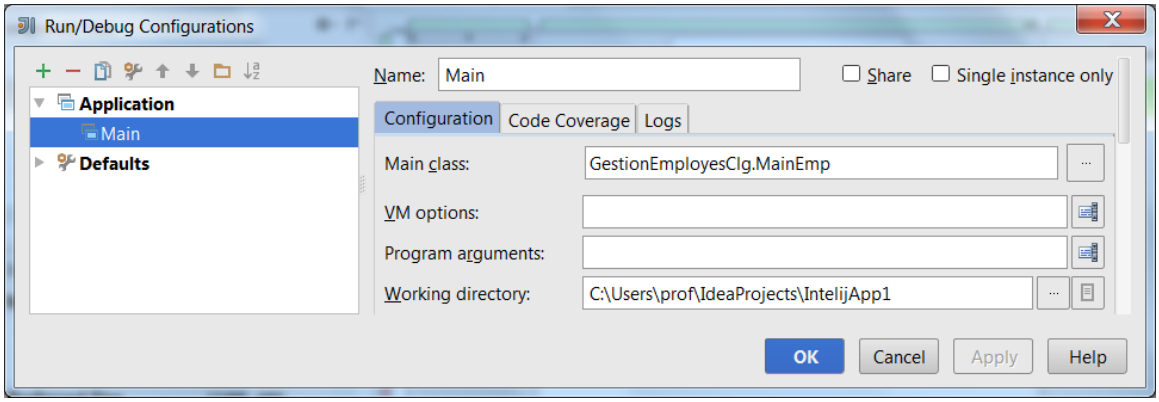


Vous pouvez créer votre classe MainEmp qui va contenir la classe main, comme suit :



Autre chose à configurer : Quelle est la classe qui va contenir la class main pour l'exécution du programme.

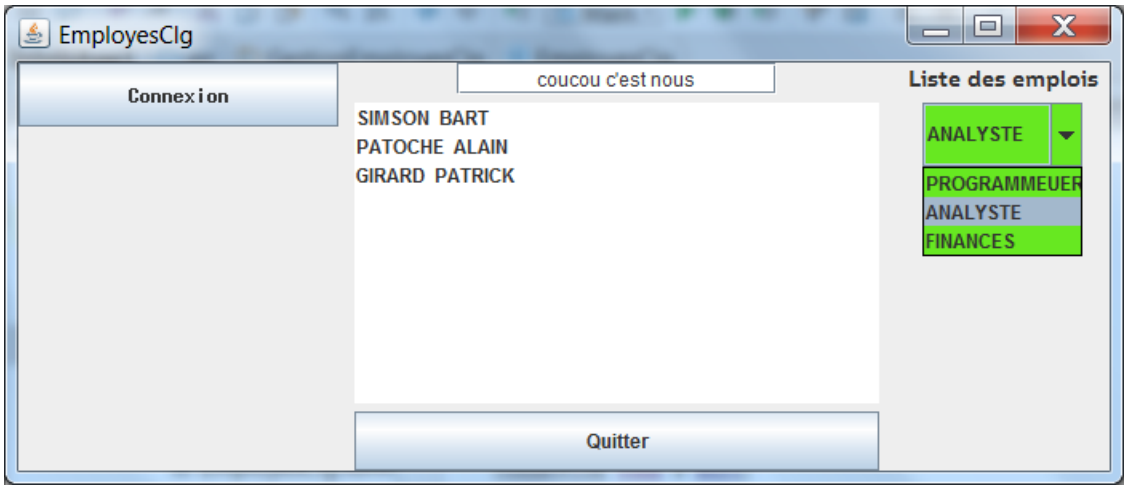
Par le menu Execution (Run), choisir Edit Configurations et choisir la classe qui va contenir votre classe **main**.



Exemple :

Pour obtenir l'interface graphique suivante, nous avons eu besoin des classes suivantes dans le package **GestionEmployesClg**

1- Une classe EmployesClg de type GUI Form, qui correspond aux classes (EmployesClg et EmployesClg,form)
2- Une classe MainEmp qui contient la classe main



Contenu de la classe MainEmp

```
package GestionEmployesClg;
import javax.swing.*;

public class MainEmp {
    public static void main(String[] args) {
        JFrame empclg = new JFrame("EmployesClg");
        empclg.setContentPane(new EmployesClg().panel1);
        empclg.setVisible(true);
        empclg.setSize(700,300);
    }
}
```

Contenu de la classe EmployesClg.form : Contien tous les controles (Jpanel, JtextField, Jbutton etc ..)de notre application

La classe EmployesClg contient :

```
package GestionEmployesClg;
import oracle.jdbc.pool.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;

public class EmployesClg {
    // Declartion des controles de la forme
    private JButton premier;
    public JPanel panel1;
    private JTextField textBonjour ;
    private JButton quitter;
    private JList list1;
    private JComboBox comboBox1;

    // Declartion des requetes SQL, resultSet et autres
    Connection conn = null;
    String user="user1";
    String mpasse ="user1";
    String url = "jdbc:oracle:thin:@205.237.244.251:1521:orcl";
   String sql1 ="SELECT  nomemp, prenomemp FROM employesbidon where emploi =
?";
   String sqlemp ="select distinct emploi from employesbidon";
    ResultSet rst;

// Fonction qui liste les emplois et les mets dans le ComboBox
    public void listeemploi()
    {
        try{
            Statement stm1 = conn.createStatement();
            ResultSet rst2 = stm1.executeQuery(sqlemp);
            while (rst2.next())
            {
                comboBox1.addItem(rst2.getString(1));
            }
           rst2.close();
          }
        catch(SQLException sql)
        { System.out.println(sql); }
    }
```

```
    public EmployesClg() {
// Fonction du bouton de connexion
      premier.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
            // votre code ici
            // Afficher un message dans la zone de texte
                textBonjour.setText("coucou c'est nous");
                try {
                   OracleDataSource ods = new OracleDataSource();
                    ods.setURL(url);
                    ods.setUser(user);
                    ods.setPassword(mpasse);
                    conn= ods.getConnection();
                    System.out.println("vous etes connectés ");
                       // appel de la fonction listeemploi
                 listeemploi();
                    }
                catch(SQLException exconn)
                {
                    System.out.println(exconn);
                }
            }
        });

// Fonction qui liste les employés en fonction d'une entrée du
//ComboBox.
        comboBox1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

             try {
         PreparedStatement stm = conn.prepareStatement(sql1);
         stm.setString(1,comboBox1.getSelectedItem().toString());
         rst = stm.executeQuery();
          DefaultListModel listModel = new DefaultListModel();
                    while (rst.next())
                    {
                   listModel.addElement(rst.getString(1) + "  " +
                   rst.getString(2));
                    }
                    list1.setModel(listModel);
                    rst.close();
                 }

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

```
            }
        });

// code du bouton quitter
        quitter.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                try
                {
                    conn.close();
                    System.out.println("connexion fermée");
                }

                catch(SQLException se)
                {
                    conn = null;
                }
                System.exit(0);
            }
        });

//  fin de public EmployesClg()
    }
// fin du public class EmployesClg
}
```

420-KEH-LG, Travail No3 — pondération 10% (JDBC)

L'entreprise InfoClg possède une vaste bibliothèque qu'elle met à la disposition de ses employés. Les ouvrages de cette bibliothèque sont classés par genre (Informatique, sciences, divertissement, droit commercial, histoire, littérature ...). Notez que l'on peut ajouter un genre.

Nous souhaitons développer une application graphique avec **intellij Idea** permettant d'informatiser une partie de la bibliothèque pour faciliter les tâches suivantes :

1- ajouter, modifier et supprimer un adhérent. (Un employé doit s'inscrire à la bibliothèque pour pouvoir emprunter)
2- consulter la liste des livres par genre. La consultation doit se faire un livre à la fois et on doit se déplacer d'un enregistrement à un autre.
3- saisir un prêt.(ajouter un prêt).
4- consulter la liste des livres en cours de prêts (titre du livre, le genre, la date du prêt, la date prévue pour le retour, le nom et le prénom de l'adhérent..)
5- la recherche d'un livre par auteur ou par titre (début d'un titre).
6- la liste des livres les plus empruntés.

Indications :

1- Un livre a plusieurs exemplaires et un exemplaire appartient à un seul livre.
2- un livre a un numéro, un titre, un auteur, un genre, une date de parution et la maison d'édition.
3- un adhérent est un employé de la compagnie. Il a un numéro, un nom, un prénom, adresse, numéro de téléphone etc. Le numéro doit être géré par un Trigger.
4- un prêt a une date début et une date prévue pour le retour.

La remise et la correction du travail a lieu le jeudi 23 avril. Aucun délai supplémentaire ne sera accordé. Le travail peut être fait en équipe de deux, mais pas plus.

L'utilisation du PreparedStatement ou du CallableStatement est fortement recommandé.

Annexe3 : Différents SGBD, lequel choisir durant le projet de session 5?

Oracle DataBase.

Disponible sous :Linux, Windows, Unix, MacOSX

Éditeur : Oracle Corporation

Type : serveur.

Licence : Propriétaire, gratuite dans sa version Express

Bonne intégration avec Java, C#, et PHP.

Au collège, facilité de déploiement.

C'est un SGBD recommandé pour les grandes applications avec un volume important d'information (> 200 Go) et beaucoup d'utilisateurs (transactions) au moins 300.

Dans un contexte où le volume d'information est >200 go et le nombre d'utilisateurs est >300 alors Oracle demeure le SGBD de choix.

Quelques Avantages

- Richesse fonctionnelle
- Pérennité de l'éditeur : avec plus de 40% de part de marché, ce n'est pas demain qu'Oracle disparaîtra
- Procédures stockés en PL-Sql (langage propriétaire Oracle, orienté ADA) ou ... en JAVA (depuis la 8.1.7) ce qui peut s'avérer utile pour les équipes de développement.
- Accès aux données système via des vues, bien plus aisément manipulable que des procédures stockées
- Services Web, support XML

Quelques inconvénients :

- Prix élevé, tant au point de vue des licences que des composants matériels (RAM, CPU) à fournir pour de bonnes performances
- Administration complexe... liée à la richesse fonctionnelle
- Fort demandeur de ressources, ce qui n'arrange rien au point précité, Oracle est bien plus gourmand en ressource mémoire que ses concurrents, ce qui implique un investissement matériel non négligeable
- Il manque la couche «base de données»
- Pas de type auto-incrément déclaratif:
- les séquences ne sont pas dédiées.

MYSQL :

Disponible sous :Linux, Windows, MacOSX, Unix, BSD (unix), OS2

Éditeur : Oracle Corporation et MYSQL AB

Type : serveur

License : GPL (Général Public Licence) :

Fait partie de LAMP : Linux, Apache, MySQL (MariaDB) et PHP

Au collège, facile à déployer. Il existe déjà un serveur LAMP.

Quelques Avantages

- Solution très courante en hébergement public
- Très bonne intégration dans l'environnement Apache/PHP
- OpenSource
- Facilité de déploiement et de prise en main.

Quelques inconvénients :

- Ne supporte qu'une faible partie des standards SQL-92
- Support incomplet des triggers et procédures stockées
- Manque de robustesse avec de fortes volumétries

MariaDB

Semblable à MYSQL. Branche de MySQL

SQL Server

Disponible sous : Windows

Éditeur : Microsoft

Type : serveur.

Licence : Propriétaire, gratuit sous sa version Express avec obligation de s'enregistrer

Bonne intégration avec C#, ASP.net (Serveur web : IIS)

Au CEGEP, il n'y a pas de serveur IIS /SQL Serveur de monté si vous voulez faire du WEB. SQL serveur et IIS, il faudra s'y prendre d'avance pour que les technicien le monte.

Quelques avantages :

- Niveau de SQL très près de la norme SQL et implémente presque toutes les possibilités de SQL.
- Services Web
- Support XML
- Langage T-SQL très convivial--- Équivalent du PL/SQL

Quelques inconvénients :

- Distributions fortement liées au système d'exploitation
- Mono-plateforme (MS Windows)

MSAccess

Disponible sous : Windows

Éditeur : Microsoft

Type : L4G.

Licence : Propriétaire. (Commerciale)

En tant que programmeurs ayant fait 3 cours de BD à moins que ce soit une exigence de l'entreprise ou du département, il est fortement déconseillé d'aller avec ACCESS.

MS-Access reste un bon choix si vous souhaitez avoir une base de donnée de petite taille mais facilement gérable et que vous avez peu connaissances en Bases de données

Quelques avantages :

- Il contient une grande série d'outils de conversion de données, pour récupérer ou exporter depuis presque n'importe quel format vers pratiquement n'importe quel format.
- Les macros permettent à des néophytes de se lancer dans une forme 'allégée' de l'automatisation.
- Quantité d'assistants dirigeant l'utilisateur vers une première solution.
- Forte intégration à la Suite Microsoft Office/VBA, déjà fortement répandue en entreprise

Quelques inconvénients

- Mono-plateforme (MS Windows)
- N'implémente pas complètement les normes SQL

- Le moteur JET étant un moteur "Fichier", il est gourmand en ressources réseau et ne convient pas pour les applications distantes.

SQLite

Disponible sous : Linux, MacOSX, Windows, Unix, BSD

Éditeur : *D. Richard Hipp*

Licence Libre (domaine public)

Type : composante logiciel

Directement intégré au programme.

Idéale pour les applications sur téléphone intelligents

- OpenSource et gratuit
- Le plus petit moteur SGBDR du marché (une simple librairie C)
- Porté sur C# (sous le nom de SQLite-C#)
- Simple d'utilisation et d'administration
- Aisément installable
- Recommandé pour micro-base (Téléphone intelligent)

Inconvénients

- Fonctionnalités minimales
- Pas d'intégrité référentielle, DDL très limité
- Ne supporte pas les jointures externes, Pas de vue matérialisée
- Volume d'information très faible

La société Solid IT vient de sortir le classement des SGBD (Système de Gestion de Base de données) selon leur popularité et sans surprise Oracle occupe le podium, suivis respectivement de MySQL et de SQL Server, en 6ieme place MangoDB, 8ieme place SQLite et 10 ieme place Casandra.

Tous les SGBD classés de de 1à 10 sont des SGBDs relationnels sauf MangoDB et Cassandra

SGBD non relationnels (noSQL)

MangoDB :

Windows, Linux, OS x

Système de gestion de bases de données orienté document. Le format des documents est BSON (Binary, JSON –JavaScript Object Notation.). Ce type de SGBD est **noSQL.**

La notion de normalisation telle que nous la connaissons n'existe plus dans MangoDB. Pour représenter les liens entre les données, il existe deux options : La référence ou les documents incorporés.

Pour les références, on stocke les relations entre les données en incluant des liens ou des références d'un document à l'autre.--> C'est le principe de normalisation dans MangoDB.

Pour les documents incorporés, toutes les informations sont contenues dans un même document appelé MangoDB Document.--> Aucune normalisation .

Plus de détails sur http://docs.mongodb.org/manual/

Avantages:

- Les données sont représentées sont proches du format natif
- Pas besoins de jointures, ce qui réduit la lenteur des accès à la BD.(documents et tableaux intégrés)

Vous pouvez consulter la structure d'un document JASON à http://www.w3schools.com/js/js_json.asp proche du format MangoDB.

Apache Cassandra :

Développée par la fondation Apache. C'est un SGBD open source

C'est un SGBD orienté colonne. On insère colonne par colonne et non ligne par ligne.

Cassandra utilisée par Facebook, twitter.

Utilise CQL Cassandra Query Language pour la gestion des données.

Veuillez consulter http://cassandra.apache.org/ pour plus de détails.

Sources :

http://en.wikipedia.org/wiki/JDBC_driver#Type_4_Driver_-_Native-Protocol_Driver

http://download.oracle.com/javase/tutorial/jdbc/overview/architecture.html

http://jguillard.developpez.com/JDBC/11.html

http://java.developpez.com/

http://docs.oracle.com/javase/7/docs/api/java/sql/ResultSetMetaData.html

http://download.oracle.com/javase/1.4.2/docs/api/java/sql/package-summary.html
pour le package Java.qsl

http://docs.oracle.com/cd/B14099_19/web.1012/b14017/jdbcejb.htm#i1001657

https://www.jetbrains.com/idea/help/gui-designer-basics.html

Introduction à JDBC, de Denis Brunet