JDBC

Java Database Connectivity

(Java – Oracle – MySQL)

Préparé par : Larbi HASSOUNI

Cette introduction présente brièvement JDBC.
Les thèmes introduits sont :

- **Qu'est ce que JDBC?**
- **Historique et versions de JDBC**
- **Types de pilotes (Driver) JDBC**
- **Etablissement de connexion aux bases de données à partir de JDBC**
- **DriverManager – Chargement de pilote JDBC**
- **DriverManager – URL de connexion**

JDBC est une API (Application Programming Interface) qui permet d’accéder aux bases de données à partir d’applications écrites en langage Java.

La version actuelle de l’API JDBC est JDBC 4.0 implémentée dans Java SE 6. Elle inclut deux packages:

- **JDBC 4.0 Core API – package java.sql**
- **JDBC 4.0 Standard Extension API – package javax.sql**

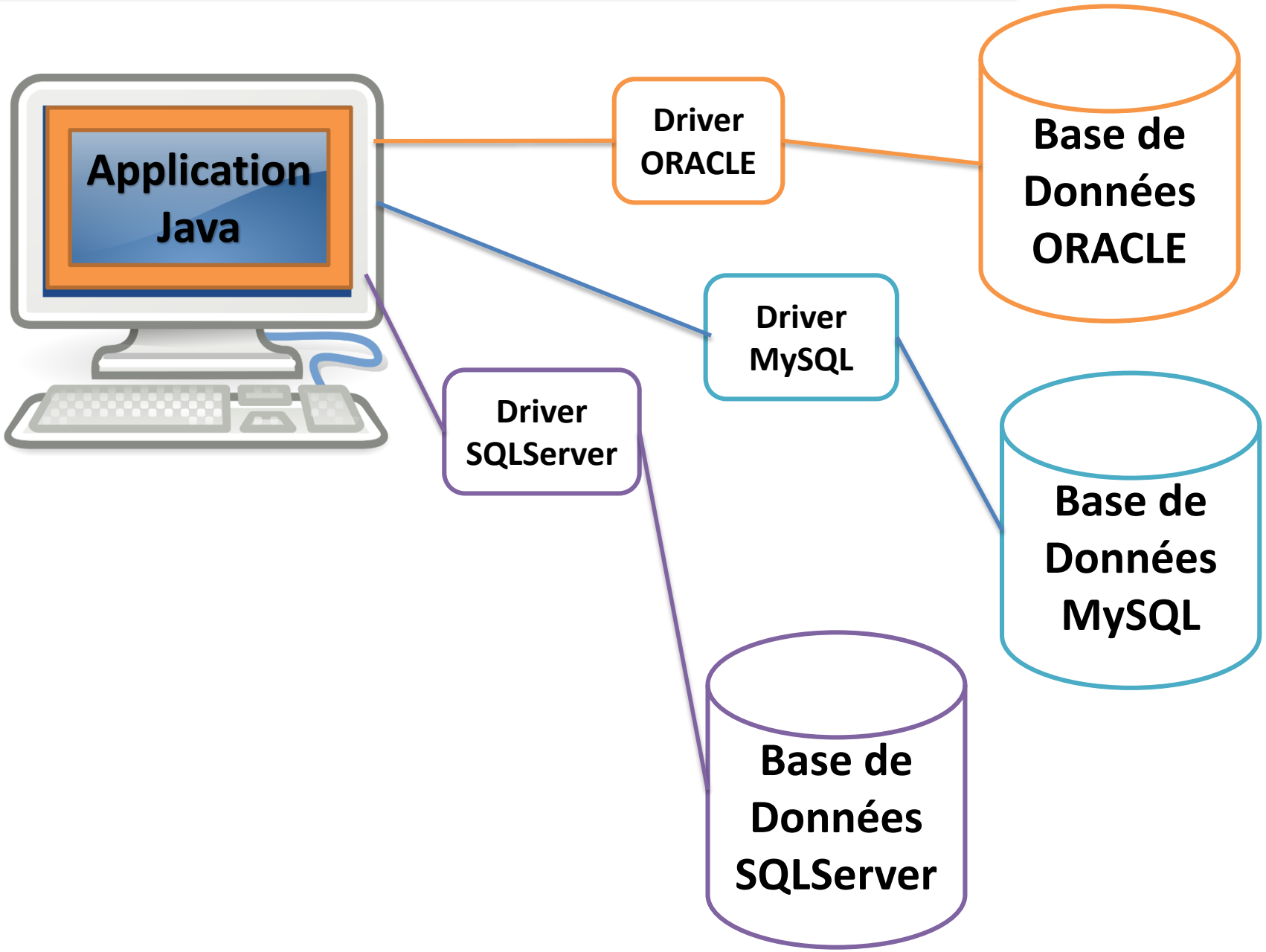
- **JDBC 4.0 Core API – package java.sql** : C’est la partie centrale de l’API JDBC. Elle est suffisante pour les applications dites “normales” sur les bases de données.

- **JDBC 4.0 Standard Extension API – package javax.sql** : C’est une extension de l’API JDBC. elle est recquise par les applications qui utilisent des pools de connections, des transactions distribuées, JNDI (Java Naming Directory Interface), et l’API RowSet.

Pour utiliser JDBC afin de connecter une application Java à une base de données, vous devez disposer d’un **pilote (Driver)** qui fait supporter l’API JDBC par le serveur de ce type de base de données.

Par exemple le pilote **OracleJDBCDriver** vous permet d’accéder à une base de données Oracle par l’intermédiare de l’API JDBC

Le Driver JDBC change d'un type de BD à un autre



La table ci-dessous fournit la liste des versiosn précédentes de JDBC

Year	JDBC Version	JSR Specification	JDK Implementation
2006	JDBC 4.0	JSR 221	Java SE 6
2001	JDBC 3.0	JSR 54	JDK 1.4
1999	JDBC 2.1		JDK 1.2
1997	JDBC 1.2		JDK 1.1

Les principales nouvelles extensions apportées par l'API JDBC 4.0 sont:

- Chargement automatique de java.sql.Driver.
- Support du type de donnée ROWID.
- Support de SQL/XML et XML.

Remarque :

L'API JDBC 4.0 a été implémenté dans Java SE 6.0.

Les pilotes spécifiques aux serveurs de base de données pour cette version d'API peuvent ne pas être encore disponibles.

Mais puisqu'il ya compatibilité ascendante, il n'ya pas de problèmes d'utiliser les drivers pour l'API JDBC 3.0 avec Java SE 6, tant que vous n'utilisez pas les nouvelles classes ou méthodes introduites par l'API JDBC 4.0.

Types de pilotes JDBC

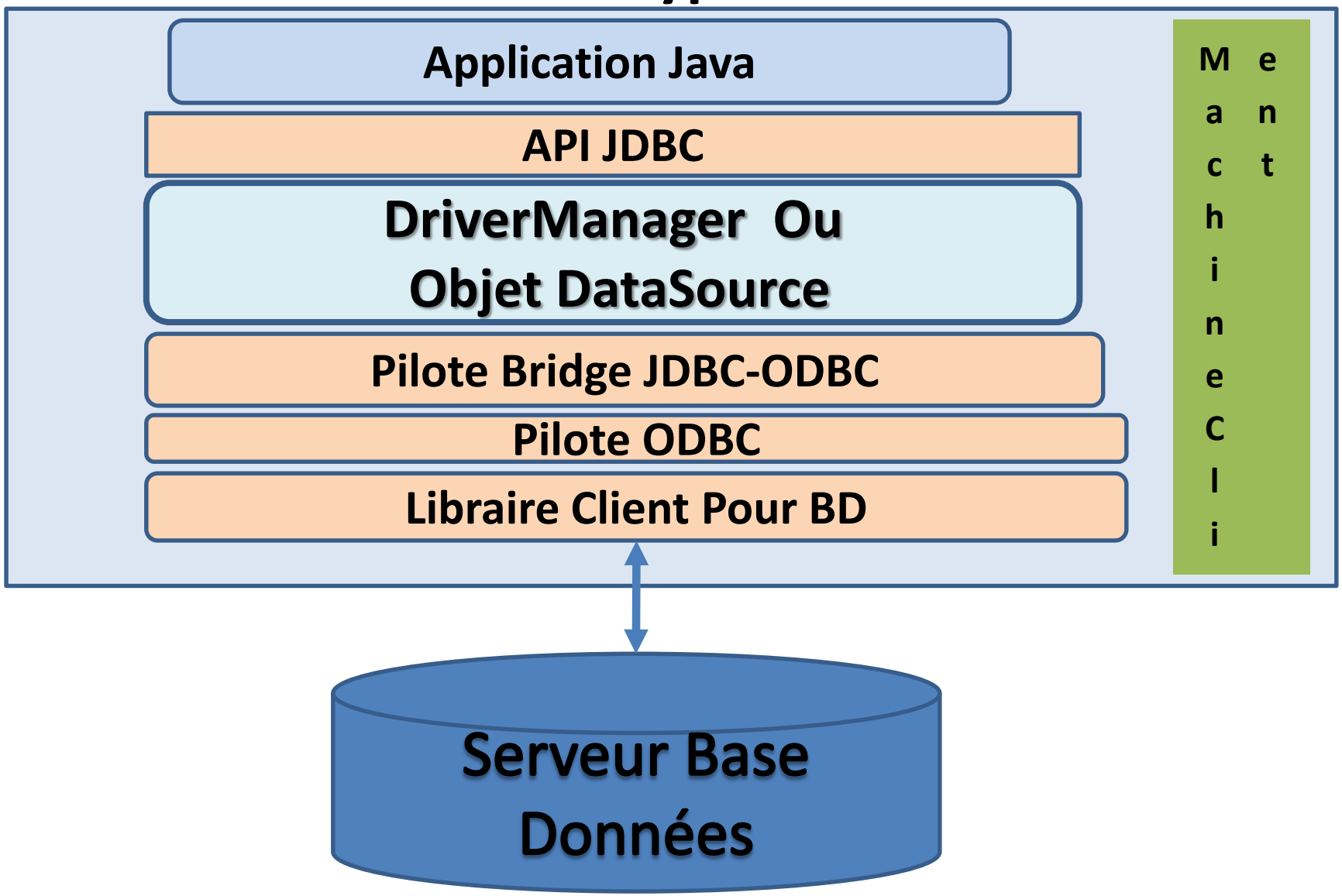
Les pilotes JDBC se répartissent en quatre types:

Type 1:

Ce type de pilote JDBC utilise un pilote ODBC pour se connecter aux bases de données. Le code binaire du pilote ODBC, et dans plusieurs cas, le code client d'une base de données, doivent être chargés dans chaque machine client qui utilise le bridge JDBC-ODBC.

Sun fournit un pilote bridge JDBC-ODBC, qui est approprié pour des utilisations expérimentales, et pour des situations où aucun autre pilote n'est disponible.

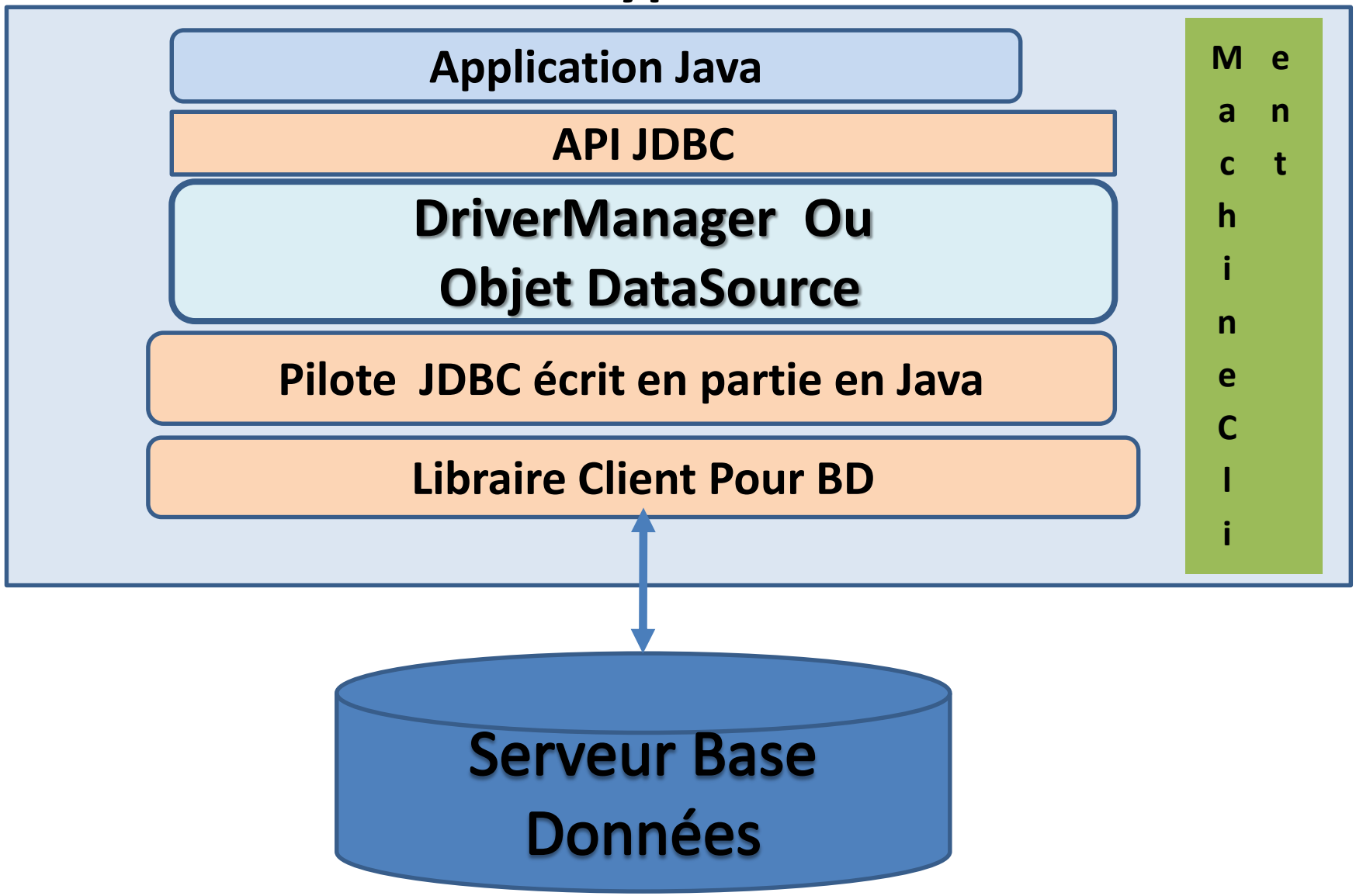
Pilotes Type 1



Type 2 :

Ce type de pilotes est écrit en partie en le langage Java et en partie en code natif. Il utilse une librairie en code natif spécifique à la base de données à laquelle on veut se connecter.

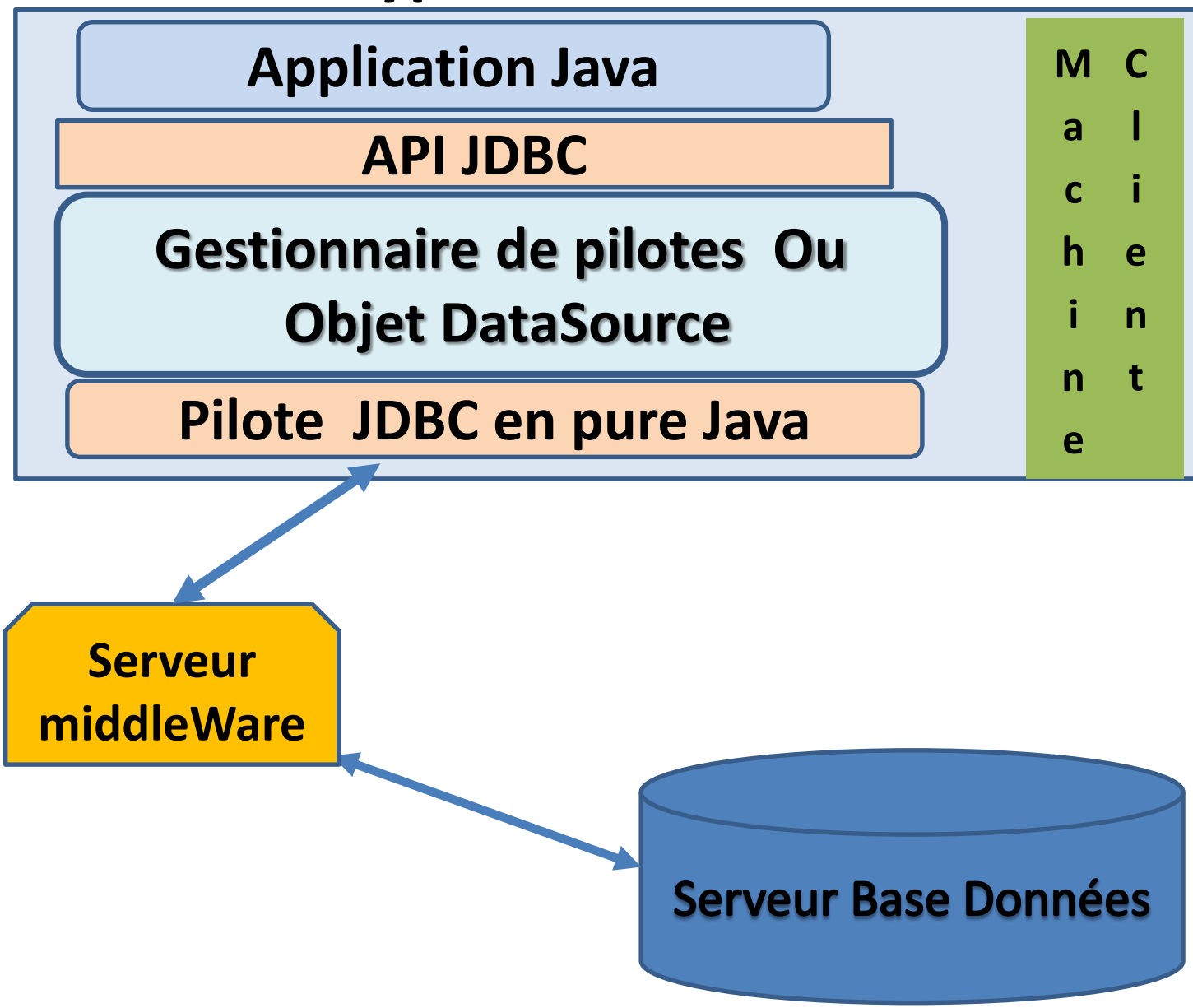
Pilotes Type 2



Type 3 :

Ce type de pilotes est purement écrit en langage Java, et communique avec un serveur middleware qui utilise un protocole réseau indépendant de la base de données. Le serveur middleware communique par la suite les requêtes du client au serveur de base de données.

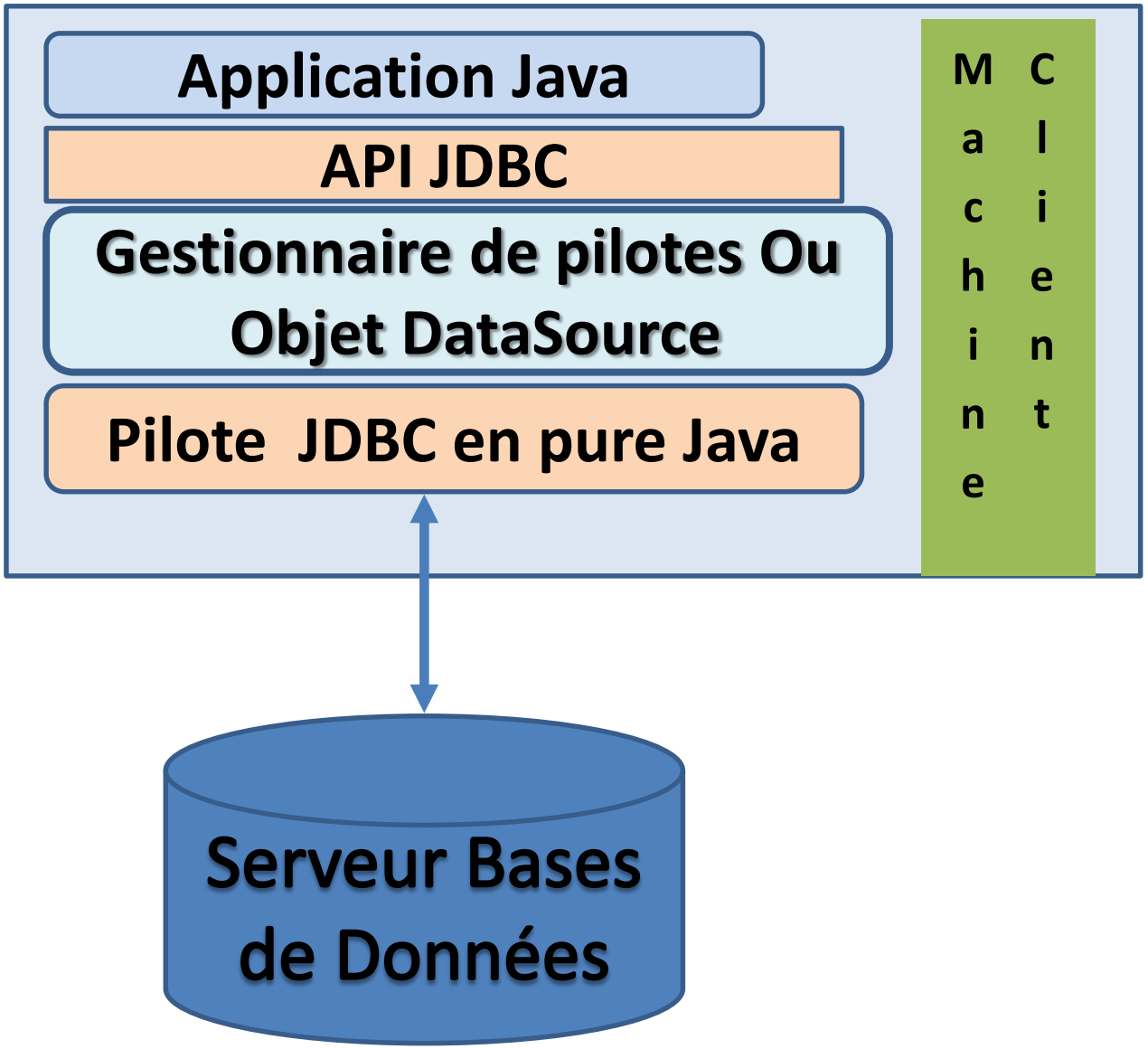
Pilotes Type 3



Type 4 :

Ce type de pilotes sont écrit à 100% en lanage Java et implémentent le protocole pour communiquer avec une base de données spécifique. Le client communique directement avec la base de données.

Pilotes Type 4



Méthodes de connexion à une base de données

L'API JDBC 4.0 offre deux méthodes pour établir une connection avec un serveur de base de données:

1. Utilisation de la classe DrirverManager :

DriverManager.getConnection(<URL_De_Connection>)

Le gestionnaire des pilotes passe l'<URL_De_Connection> à tous les pilotes qui ont été chargés dans l'espoir de trouver un qui reconnaît l'URL et créer une connection à la base de données

Méthodes de connexion à une base de données

Exemple a: Oracle

// Chargement du pilote d'oracle:

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

//Etablir une connection à une base de données Oracle:

```
String url = "jdbc:oracle:thin:@localhost:1521:xe“;
Connection conn = DriverManager.getConnection(url,“utilisateur",”password");
```

Exemple b: MySQL

// Chargement du pilote MySQL:

```
Class.forName(" com.mysql.jdbc.Driver ");
```

//Etablir une connection à une base de données MySQL:

```
String dbURL = "jdbc:mysql://localhost:3306/<nomBD>";
Connection conn = DriverManager.getConnection(url,“utilisateur",”password");
```

Méthodes de connexion à une base de données

2. Utilisation d'un objet DataSource : ds.getConnection()

Nous présenterons cette méthode ultérieurement

JDBC DriverManager-Chargement du pilote JDBC

Pour créer une connection à une base de données à l'aide de la classe DriverManager, il faut charger le pilote qui sait créer une connection au serveur de la base de données en question.

Le pilote chargé sera automatiquement enregistré auprès du DriverManager

Il ya deux méthodes pour charger un pilote:

- Utiliser la méthode Class.forName()
- Utiliser la propriété **jdbc.drivers** du **java.lang.System**

Le programme de la diapositive suivante permet de tester la méthode qui utilise Class.forName() et qui est la plus utilisée.

Pour tester ce programme vous devez disposer des pilotes pour les bases de données ORACLE et MySQL.

```
import java.sql.*;
import java.util.*;
public class LoadJdbcDriver {
import java.sql.*;
import java.util.Enumeration;
public class LoadJdbcDriver {
  public static void main(String [] args) {
    try {
      System.out.println("Before loading OracleDriver:");
      listDrivers();
// Charger le pilote d'oracle
      Class.forName("oracle.jdbc.OracleDriver");
       System.out.println("After loading OracleDriver:");
       listDrivers();
// Charger le pilote de MySQL
      Class.forName("com.mysql.jdbc.Driver");
       System.out.println("After loading MySQLDriver:");
      listDrivers();
    }
      catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());}
  }
```

```
private static void listDrivers() {
    Enumeration<Driver> driverList = DriverManager.getDrivers();
    while (driverList.hasMoreElements()) {
      Driver driverClass = (Driver) driverList.nextElement();
      System.out.println("   "+driverClass.getClass().getName());
    } }
}
 }
```

Test: Charger le pilote d'oracle et de MySQL avec Class.forName():

C:\..>javac LoadJdbcDriver.java

>**java -cp .;ojdbc6.jar; mysql-connector-java-5.1.38-bin.jar LoadJdbcDriver**

L'URL de connection utilisé par la classe DriverManager pour établir une connection à une base de donnée a le format:

jdbc:<subprotocol>:<subname>

<subprotocol> est utilisé pour identifier la classe du pilote JDBC qui va créer un objet connection basé sur les informations fournies par **<subname>.**

Exemples d'URL :

Exemple 1 :

"jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl"

"oracle:thin" identifie le pilote qui va créer un objet connection à la base de données **orcl** sur le poste **"PC-de-Hassouni"** en utilisant le port **"1521"**

Exemple 2:

"jdbc:mysql://localhost:3306/elibrairie";

"mysql" identifie le pilote qui va créer un objet connection à la base de données **"elibrairie"** sur le poste **local** en utilisant le port **"3306"**

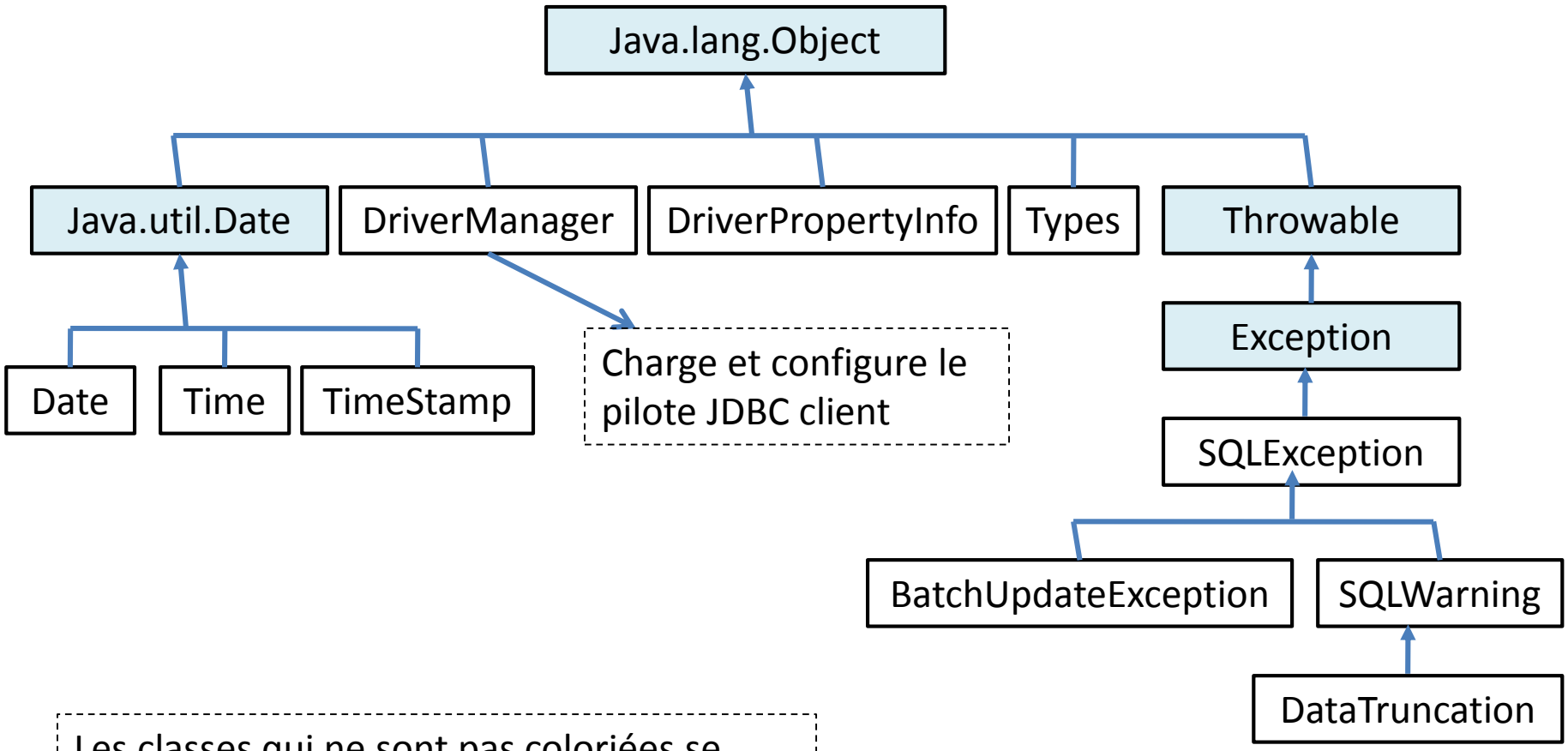
JDBC DriverManager-URL de connection

La classe DriverManager offre 3 méthodes pour créer un objet connection:

```
Connection con = DriverManager.getConnection(String url);

Connection con = DriverManager.getConnection(String url,   Properties info)

Connection con = DriverManager.getConnection(String url,   String user, String password)
```

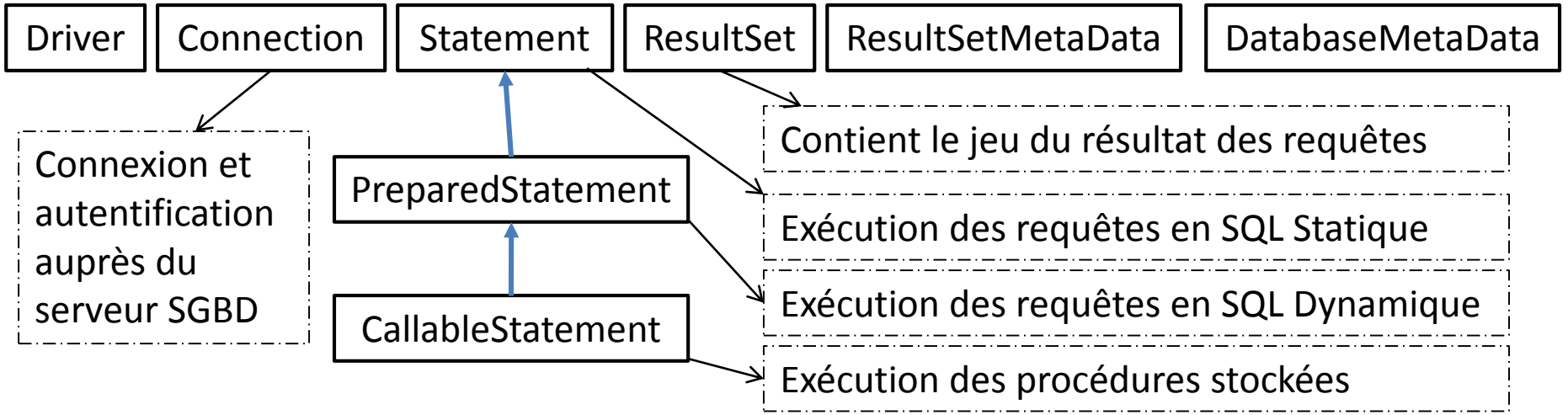
Nous utiliserons la 3ème méthode qui est la plus utilisée.

Classes de JDBC



Les classes qui ne sont pas coloriées se trouvent dans le package **java.sql**

Interfaces de JDBC (java.sql)



Ces interfaces définissent une **abstraction du pilote (driver) de la** base de données. Chaque fournisseur propose **sa propre implémentation de ces** interfaces.

Les classes d'implémentation du **driver jdbc** sont dans une archive (fichier jar ou zip) qu'il faut intégrer (sans le décompresser) au niveau du **classpath** de l'application au moment de l'exécution

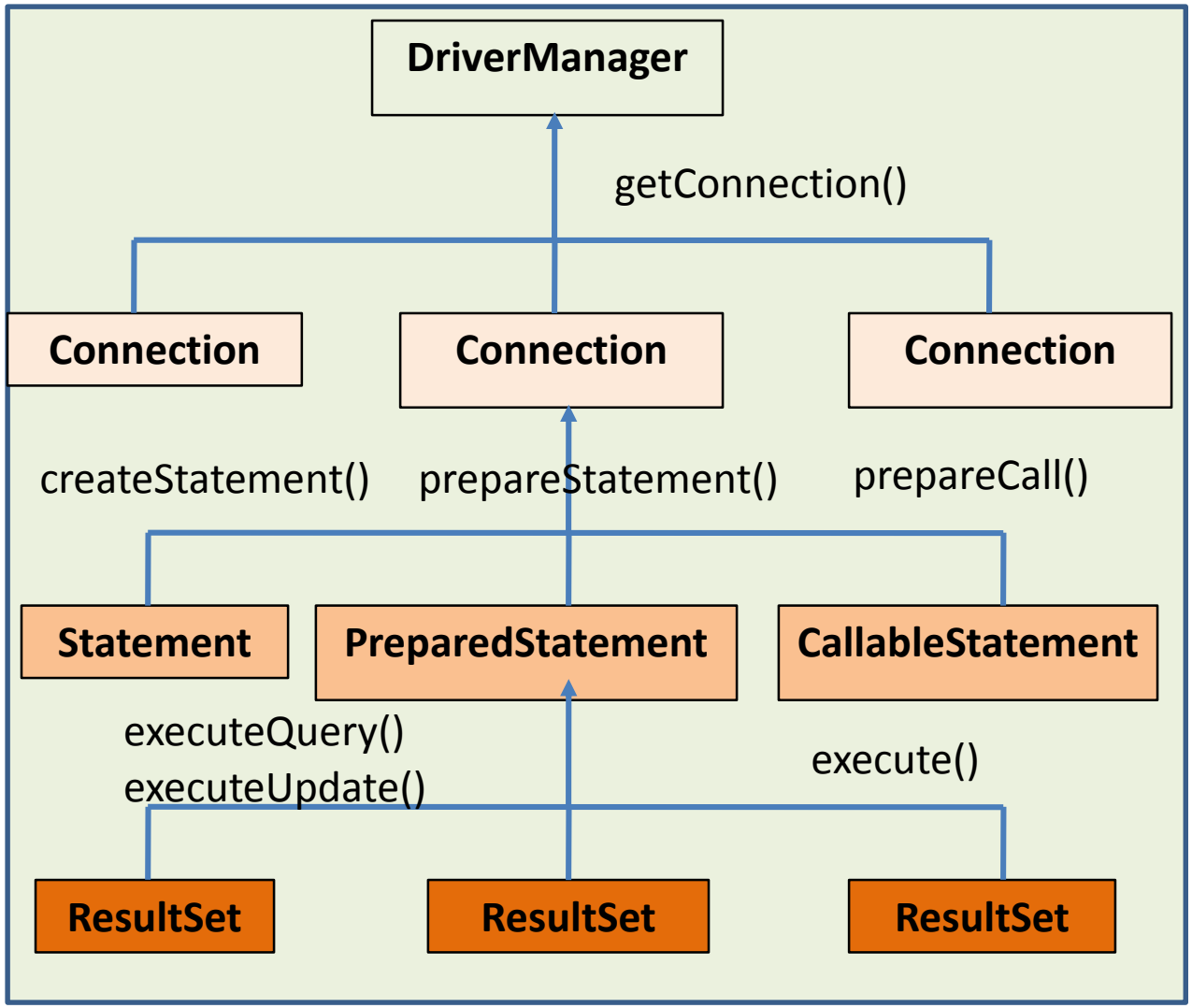
Au niveau du programme d'application **on ne travaille qu'avec les abstractions (interfaces) sans se** soucier des classes effectives d'implémentation

Objets instanciés a partir des classes et interfaces de JDBC

DriverManager permet de créer des objets Connection

Un objet Connection permet de créer des objets encapsulant des requêtes SQL

Les objets encapsulant les requêtesSQL permettent de créer des objets ResultSet encapsulant le résultat d'une requête



Résumé sur les formats et exemples d'URL de connection

*Ce résumé fournit les formats et exemples d'URL de connection utilisée par les pilotes JDBC avec la classe **DriverManager**.*

La documentation sur JDBC recommande d'utiliser l'interface **DataSource** pour créer des objets connection aux bases de données.

Cependant la plupart des applications Java continuent à utiliser la méthode **DriverManager.getConnection()** pour créer des objets connection.

La méthode **DriverManager.getConnection()** nécessite qu'on lui fournisse en **argument une URL** de connection (String) qui est reconnue par le pilote JDBC.

Chaque pilote JDBC requiert un format particulier pour son URL de connection.

URL pour Access

Nom du Driver : Bridge JDBC-ODBC

Fichier JAR du Driver : aucun (inclus dans Java SE 1.6)

Connection URL Formats:

jdbc:odbc:DSN[;user=xxx][;password=xxx]

Exemples d'URL de Connection :

jdbc:odbc:Fichier_Texte

jdbc:odbc:MaBase_ACCESS

jdbc:odbc:MaBase_SQL_SERVER;user=Moha;password=TopSecret

Pilote pour MySQL

Nom du Driver : MySQL Connector/J

Fichier JAR du Driver: mysql-connector-java-5.0.38-bin.jar

Formats de l'URL de Connection :

jdbc:mysql://[host][:port]/[database][?p1=v1]...

Exemples d'URL de Connection:

jdbc:mysql://localhost:3306/MaBD?user=Moha&password=TopSecret

Pilote pour Oracle

Nom du Driver : Oracle JDBC Thin client-side driver

Fichier JAR du Driver : ojdbc6.jar

Formats d'URL de Connection :

jdbc:oracle:thin:[user/password]@[host][:port]:SID

jdbc:oracle:thin:[user/password]@//[host][:port]/SID

Exemples d'URL de Connection :

jdbc:oracle:thin:Moha/TopSecret@localhost:1521:orcl

jdbc:oracle:thin:Moha/TopSecret@//localhost:1521:orcl

Pilote pour SQLServer

Nom du Driver : Miscrosoft JDBC Driver

Fichier JAR du Driver : sqljdbc.jar

Formats d'URL de Connection :

jdbc:sqlserver://host[:port];user=xxx;password=xxx[;p=v]

Exemples d'URL de Connection :

jdbc:sqlserver://localhost;user=Moha;password=Topsecret
jdbc:sqlserver://localhost:1269;user=Moha;password=TopSecret
jdbc:sqlserver://localhost;user=Moha;password=Secret;database=myDB

ORACLE - JDBC

Types de pilotes JDBC pour Oracle

Cette partie du cours présente comment accéder à une base de données Oracle.

Elle traite les thèmes suivants :

- **Vue générale sur les pilotes JDBC pour Oracle**
- **Installation du pilote JDBC Thin Client-Side (coté client)**
- **Chargement de la classe JDBC Driver - ojdbc14.jar**
- **URL de connection du pilote JDBC**
- **Informations sur le server SGBD et sur le pilote JDBC**
- **Crétaion de nouvelles tables : "CREATE TABLE“**
- **Insertion de nouvelles lignes dans une table "INSERT INTO“**
- **Extraction des lignes à partir d’une table “SELECT”**
- **Modification des valeurs des colonnes d’une table “UPDATE”**
- **Supression des lignes d’une table “DELETE”**

Vue générale sur les pilotes JDBC pour Oracle

Oracle fournit 4 types differents de pilotes JDBC, pour être utilisés dans différents scénarios de déploiement.

1. JDBC OCI client-side driver: C'est un pilote de type 2 qui utilise des méthodes natives Java pour appeler des programmes d'une libraire C.
   Cette librairie C, appelée OCI (Oracle Call Interface), intéragit avec la base de données Oracle.

   Ce pilote JDBC OCI recquiert l'installation d'Oracle client de la même version que le pilote.

   L'utilisation des méthodes natives rend le pilote JDBC OCI dépendant de la plateforme.Oracle supporte les plateformes Solaris, Windows, et plusieurs autres platformes.

   Le fait que le pilote Oracle JDBC OCI dépende d'une librairie C, le rend inaproprié pour développer des applets java.

2. JDBC Thin client-side driver: C’est un pilote JDBC de type 4 qui utilise Java pour établir une connection directe avec la base de doonnées.

Il implémente SQL*Net8 d’Oracle en utilisant ses propres sockets Java basées sur TCP/IP.

Ce pilote ne requiert pas l’installation d’Oracle client, mais nécessite que le serveur SGBD soit configuré avec un listener (écouteur)TCP/IP.

Puisqu’il est écrit entièrement en Java, ce pilote est indépendant de la palteforme.

3. JDBC Thin server-side driver: C’est un autre pilote JDBC de type 4 qui utilise Java pour se connecter directement avec un serveur Oracle.

Ce pilote est utilisé à l’intérieur du serveur Oracle.

Ce pilote offre les mêmes fonctionnalités que le pilote JDBC Thin Client-Side vu précédemment, mais s’exécute à l’intérieur du serveur SGBD d’Oracle et est utilisé pour accéder à des bases de données à distance.

4. JDBC Server-Side Internal driver: C’est un autre pilote JDBC de type 2 qui utilise des méthodes natives Java pour appeler des programmes d’une libraire C.

   Cette libraire C fait partie d’un processus du serveur Oracle et communique directement avec le moteur SQL interne d’Oracle.

   Ce pilote accède au moteur SQL en utilisant des appels de fonctions internes évitant ainsi tout trafic réseau.

Ce cours utilise "JDBC Thin client-side driver" pour éviter d'installer toute autre librairie native.

Bien sûr , il est aussi possible de connecter vos programmes Java à une base de données Oracle en utilisant le bridge JDBC-ODBC et le pilote ODBC pour un serveur SGBD Oracle.

Installation du pilote JDBC Thin Client-Side

Pour installer le pilote JDBC Thin Client-Side il suffit d'obtenir le fichier "ojdbc6.jar".

Si vous avez installé Oracle 11g Express Edition sur votre poste vous devez le trouver dans le répertoire:

C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar

Installation du pilote JDBC Thin Client-Side

Si vous ne l'avez pas, vous pouvez le télécharger à l'adresse:

http://www.oracle.com/technetwork/apps-tech/jdbc-112010-090769.html

Sauvegardez le fichier "ojdbc6.jar" dans un répertoire de votre choix.

Pour éviter de spécifier un long chemin dans classpath, j'ai placé une copie du fichier ojdbc6.jar dans mon répértoire courant.

Chargement de la classe JDBC Driver - ojdbc6.jar

Oracle Thin client-side driver ojdbc6.jar est un pilote JDBC 4.0 .

La classe du pilote, oracle.jdbc.OracleDriver, ne nécessite d'être chargée si vous souhaitez utiliser la classe DriverManager pour créer les objets Connection.

N'oubliez pas de spécifier le fichier ojdbc6.jar dans le classpath lorsque vous exécutez le programme.

URL de connection du pilote JDBC d'Oracle

Pour créer des objets Connection à l'aide de la classe DriverManager, vous devez savoir construire l'URL de connection qui fournit des informations d'accès au serveur Oracle.

La méthode getConnection() de la classe DriverManager a trois formats différents:

1. **static Connection getConnection(String url)**
2. **static Connection getConnection(String url, String user, String password)**
3. **static Connection getConnection(String url, Properties info)**

Le format de l'URL de connection Oracle pour le pilote **<thin client-side ojdbc6.jar>** dépend du format de la méthode getConnection() utilisé.

URL de connection du pilote JDBC d'Oracle

Format de l'URL pour **:**

getConnection(String url)

jdbc:oracle:thin:user/password@host:port:SID

jdbc:oracle:thin:user/password@//host:port/SID

Format de l'URL pour :

getConnection(String url, String user, String password)

jdbc:oracle:thin:@host:port:SID

jdbc:oracle:thin:@//host:port/SID

URL de connection du pilote JDBC d'Oracle

user : Nom d'un compte utilisateur créé sur le serveur Oracle.

password : Mot de passe du compte.

host : Nom hote du poste où s'exécute le serveur Oracle.
par default c'est 127.0.0.1 – l'adresse IP du localhost.

port : Numéro du port utilisé par Oracle listener. La valeur par défaut est 1521.

SID : L'ID Système de l'instance Base de données créée sur le serveur Oracle.
Oracle 10g Release 2 crée par défaut une instance de base de donnée appelée orcl. Pour Oracle 10g Express Edition le SID est XE.

Remarque:
Pour les besoins de test des programmes, j'ai créé un compte dont le nom est **MOHA** et le password est **topsecret**.
Le nom hote de mon poste est **hassouni-PC**
Sur mon poste j'ai installé Oracle 11g Express Edition, j'ai gardé le nom par défaut de l'instance de la base de données créée qui est **xe**
Le numéro du port du listener est : **1521**

URL de connection du pilote JDBC d'Oracle

En tenant compte des paramètres du poste sur lequel je vais exécuter les programmes de test, le format de l'URL de connexion que nous rencontrerons sont:

Format 1 :

String url = « jdbc:oracle:thin:MOHA/TopSecret@hassouni-PC:1521:xe »;

ou

String url = « jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521:xe»;

Format 2 :

String url = « jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl »;

ou

String url = « jdbc:oracle:thin:@//localhost:1521:orcl »;

String user = « MOHA »;

String password = « TopSecret »;

```
import java.sql.*;
public class OracleConnectionUrlFormat1 {
  public static void main(String [] args) {
    Connection con = null;
    try {
// Chargement du Driver Oracle
     Class.forName("oracle.jdbc.OracleDriver") ;
     System.out.println("Le Driver JDBC Oracle est charge avec succes.");
     con = DriverManager.getConnection(
"jdbc:oracle:thin:MOHA/TopSecret@localhost:1521:xe");
     System.out.println("Connexion reussie avec  url =
jdbc:oracle:thin:MOHA/TopSecret@localhost:1521:orcl");
     con.close();

      con = DriverManager.getConnection(
 "jdbc:oracle:thin:MOHA/TopSecret@hassouni-PC:1521:xe");

  System.out.println("Connexion reussie avec  url=jdbc:oracle:thin:MOHA/TopSecret@PC-de-
Hassouni:1521:orcl");
     con.close();
```

```
 con = DriverManager.getConnection(
"jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521/xe");
  System.out.println("Connexion reussie avec  url =
jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521/orcl");
      con.close();

 con = DriverManager.getConnection(
 "jdbc:oracle:thin:MOHA/TopSecret@//hassouni-PC:1521/xe");
  System.out.println("Connexion reussie avec
url=jdbc:oracle:thin:MOHA/TopSecret@//PC-de-Hassouni:1521/orcl");
      con.close();

   } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
   }  }}
```

```
import java.sql.*;
public class OracleConnectionUrlFormat1 {
  public static void main(String [] args) {
    Connection con = null;
    try {
// Chargement du Driver Oracle
     Class.forName("oracle.jdbc.OracleDriver") ;
     System.out.println("Le Driver JDBC Oracle est charge avec succes.");
     con = DriverManager.getConnection( "jdbc:oracle:thin:MOHA/TopSecret@localhost:1521:xe");
     System.out.println("Connexion reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@localhost:1521:xe");
     con.close();

     con = DriverManager.getConnection( "jdbc:oracle:thin:MOHA/TopSecret@hassouni-PC:1521:orcl");
     System.out.println("Connexion reussie avec  url=jdbc:oracle:thin:MOHA/TopSecret@PC-de-Hassouni:1521:xe");
     con.close();

  con = DriverManager.getConnection(     "jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521/xe");
  System.out.println("Connexion reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521/xe");
     con.close();

   con = DriverManager.getConnection( "jdbc:oracle:thin:MOHA/TopSecret@//hassouni-PC:1521/xe");
  System.out.println("Connexion reussie avec url=jdbc:oracle:thin:MOHA/TopSecret@//hassouni-PC:1521/xe");
     con.close();

    } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
    } }}
```

```
import java.sql.*;
public class OracleConnectionUrlFormat2 {
  public static void main(String [] args) {
   Connection con = null;
    try {
      Class.forName("oracle.jdbc.OracleDriver") ;
     System.out.println("Le Driver JDBC Oracle est charge avec succes.");
     con = DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl", "MOHA", "TopSecret");
     System.out.print("Connexion reussie avec  url =
jdbc:oracle:thin:@localhost:1521:orcl");
     System.out.println("\t\tuser = Moha\tPassword = TopSecret");
      con.close();
     con = DriverManager.getConnection(
"jdbc:oracle:thin:@hassouni-PC:1521:orcl", "MOHA",
"TopSecret");
     System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@PC-de-
Hassouni:1521:orcl");
     System.out.println("\tuser = Moha\tPassword = TopSecret");
       con.close();
```

```
     con = DriverManager.getConnection(
"jdbc:oracle:thin:@//localhost:1521/orcl", "MOHA", "TopSecret");
     System.out.print("Connexion reussie avec  url =
jdbc:oracle:thin:@//localhost:1521/orcl");
     System.out.println("\t\tuser = Moha\tPassword = TopSecret");
     con.close();

    con = DriverManager.getConnection(
"jdbc:oracle:thin:@//PC-de-Hassouni:1521/orcl", "MOHA",
"TopSecret");
    System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@PC-de-
Hassouni:1521:orcl");
    System.out.println("\tuser = Moha\tPassword = TopSecret");
     con.close();
    } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
    }  }}
```

```
import java.sql.*;
public class OracleConnectionUrlFormat2 {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.OracleDriver") ;
      System.out.println("Le Driver JDBC Oracle est charge avec succes.");

     con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl", "MOHA", "TopSecret");
      System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@localhost:1521:orcl");
      System.out.println("\t\tuser = Moha\tPassword = TopSecret");
      con.close();

      con = DriverManager.getConnection("jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl", "MOHA", "TopSecret");
      System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl");
      System.out.println("\tuser = Moha\tPassword = TopSecret");
      con.close();

      con = DriverManager.getConnection("jdbc:oracle:thin:@//localhost:1521/orcl", "MOHA", "TopSecret");
      System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@//localhost:1521/orcl");
      System.out.println("\t\tuser = Moha\tPassword = TopSecret");
      con.close();
    con = DriverManager.getConnection("jdbc:oracle:thin:@//PC-de-Hassouni:1521/orcl", "MOHA", "TopSecret");
    System.out.print("Connexion reussie avec  url = jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl");
    System.out.println("\tuser = Moha\tPassword = TopSecret");
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }  }}
```

URL de connection du pilote JDBC d'Oracle

Résultat de l'exécution du programme utilisant le premier format de l'URL:

```
C:\Users\hassouni\JavaProgs\JDBC\Herong\Oracle>javac OracleConnectionUrlFormat1.java

C:\Users\hassouni\JavaProgs\JDBC\Herong\Oracle>java -cp .;ojdbc14.jar OracleConnectionUrlFormat1
Le Driver JDBC Oracle est charge avec succes.
Connection reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@localhost:1521:orcl
Connection reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@PC-de-Hassouni:1521:orcl
Connection reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@//localhost:1521/orcl
Connection reussie avec  url = jdbc:oracle:thin:MOHA/TopSecret@//PC-de-Hassouni:1521/orcl
```

Résultat de l'exécution du programme utilisant le deuxième format de l'URL:

```
C:\..>javac OracleConnectionUrlFormat2.java

C:\..>java -cp .;ojdbc14.jar OracleConnectionUrlFormat2
Le Driver JDBC Oracle est charge avec succes.
Connexion reussie avec url= jdbc:oracle:thin:@localhost:1521:orcl            user = Moha    Password = TopSecret
Connexion reussie avec url= jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl    user = Moha    Password = TopSecret
Connexion reussie avec url= jdbc:oracle:thin:@//localhost:1521/orcl        user = Moha    Password = TopSecret
Connexion reussie avec url= jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl   user = Moha    Password = TopSecret
```

Informations sur le serveur SGBD et sur le pilote JDBC

Il est possible d'obtenir des informations sur le serveur SGBD et sur le pilote JDBC à l'aide d'un objet **DatabaseMetaData** obtenu par la méthode **getMetaData()** d'un objet connection.

Le programme exemple qui suit obtient le nom du serveur SGBD, et sa version, respectivement par les méthodes :

- getDatabaseProductName();
- getDatabaseProductVersion();

Il obtient aussi le nom du Driver, sa version, et les numéros majeur et mineur de la version , respectivement par les méthodes :

- getDriverName();
- getDriverVersion();
- getJDBCMajorVersion();
- getJDBCMinorVersion();

Pour obtenir d'autres informations, consulter la doc de L'API
(Package java.sql, interface DatabaseMetaData)

Informations sur le serveur SGBD et sur le pilote JDBC

```
import java.sql.*;
public class OracleDatabaseInfo {
  public static void main(String [] args) {
    Connection con = null;
    try {
//Chargement du Driver JDBC
     Class.forName("oracle.jdbc.OracleDriver");

//Obtention d'un objet connection
     con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","Moha","topsecret");

// Information sur la base de données et le pilote JDBC
     DatabaseMetaData meta = con.getMetaData();

     System.out.println("Server name: "  + meta.getDatabaseProductName());
     System.out.println("Server version: " + meta.getDatabaseProductVersion());
     System.out.println("Driver name: "  + meta.getDriverName());
     System.out.println("Driver version: "  + meta.getDriverVersion());
     System.out.println("JDBC major version: "  + meta.getJDBCMajorVersion());
     System.out.println("JDBC minor version: "   + meta.getJDBCMinorVersion());

//Fermeture de la connection
     con.close();
    } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
    } }}
```

Informations sur le serveur SGBD et sur le pilote JDBC

Résultat de l'exécution du programme:

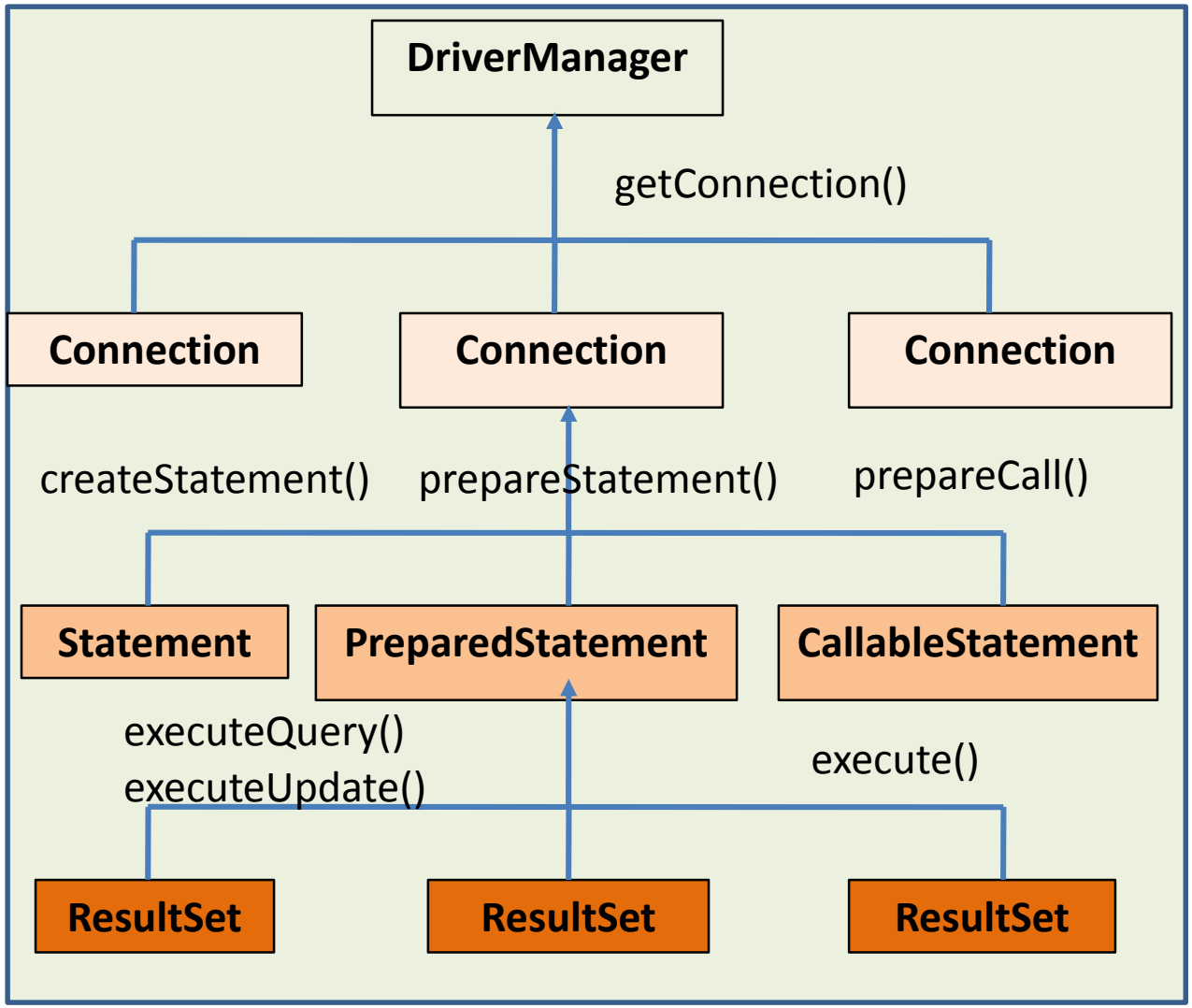
```
C:\..>javac OracleDatabaseInfo.java

C:\..>java -cp .;ojdbc14.jar  OracleDatabaseInfo
Server name: Oracle
Server version: Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production
With the Partitioning, OLAP and Data Mining options
Driver name: Oracle JDBC driver
Driver version: 10.2.0.3.0
JDBC major version: 10
JDBC minor version: 2
```

Objets instanciés a partir des classes et interfaces de JDBC

DriverManager permet de créer des objets Connection

Un objet Connection permet de créer des objets encapsulant des requêtes SQL

Les objets encapsulant les requêtesSQL permettent de créer des objets ResultSet encapsulant le résultat d'une requête



Crétaion de nouvelles tables : "CREATE TABLE“

L’exécution d’une commande LDD ou LMD à partir d’un programme java est très simple. Il suffit de suivre la procédure suivante:

1. Chargement du pilote :

Class.forName(<nom du piloteJDBC >);

2.Création d’un objet Connection:

(Connection con = DriverManager.getConnection(<url>)

3.Création d’un objet Statement par la méthode createStatement de l’objet Connection:

(Statement sta = con.createStatement())

4.Création d’une String qui contient la commande SQL LDD/LMD

(String SQLCmdLddLmd = <Commande LDD / LMD>)

5.Exécution de la commande par la méthode executeUpdate() de l’objet Statement

sta.executeUpdate(<commande LDD/LMD>)

Le programme exemple suivant permet de créer une table.

```
import java.sql.*;
public class OracleCreateTable {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:xe";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
      Statement sta = con.createStatement();
      String SQLCreateTable = "CREATE TABLE Profile ("
        + " ID NUMBER(5) CONSTRAINT PKProfile PRIMARY KEY,"
        + " FirstName VARCHAR2(20) NOT NULL,"
        + " LastName VARCHAR2(20),"
        + " Point NUMBER DEFAULT 0.0)";
      int count = sta.executeUpdate(SQLCreateTable);
      System.out.println("Creation de la table reussie");
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }  }}
```

Programme avec SGBD MySQL

```
import java.sql.*;
public class MySQLCreateTable {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("com.mysql.jdbc.Driver");
      String url = "jdbc:mysql://localhost:3306/testjdbc";
      con = DriverManager.getConnection(url,"moha","topsecret");
      Statement sta = con.createStatement();
      String SQLCreateTable = "CREATE TABLE Profile ("
        + " ID INT PRIMARY KEY,"
        + " FirstName VARCHAR(20) NOT NULL,"
        + " LastName VARCHAR(20),"
        + " Point DECIMAL(9,2))";
      int count = sta.executeUpdate(SQLCreateTable);
      System.out.println("Creation de la table reussie count = " +
count);
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }
}
}
```

Insertion de nouvelles lignes dans une table : “INSERT INTO“

La commande INSERT INTO est une commande LMD, pour l’exécuter, il faut suivre les mêmes étapes que nous avons présentées pour la création d’une table.

Le programme exemple qui suit insère 12 lignes dans la table Profile créée par le programme précédent.

Les 10 dernières lignes insérées reçoivent des valeurs aléatoires générées par un objet Random.

```
import java.util.*;
import java.sql.*;
public class OracleMultipleInserts {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:orcl";
      con = DriverManager.getConnection(
            url,"Moha","TopSecret");
      Statement sta = con.createStatement();
      int count = 0;
// insérer une ligne en utilisant les valeurs par défaut
    String SQLInsert = "INSERT INTO Profile(ID, FirstName,
LastName) VALUES (1, 'Moha', 'John')";
      count += sta.executeUpdate(SQLInsert);
```

```
// insérer une seule ligne dont on fournit les valeurs
    SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)
VALUES (2, 'Manolo', 'Brahim', 999.99)";
     count += sta.executeUpdate(SQLInsert);
// insérer des lignes avec une boucle et des valeurs aléatoires
    Random r = new Random();
    for (int i=0; i<10; i++) {
            Float points = 1000*r.nextFloat();
            String firstName = Integer.toHexString(r.nextInt(9999));
            String lastName = Integer.toHexString(r.nextInt(999999));
     SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)"   +
" VALUES (" + (i+10) + ", '" + firstName + "', '" + lastName + "', " + points +
")";
            count += sta.executeUpdate( SQLInsert);
    }
    System.out.println("Nombre de lignes inserees: "+count);
     sta.close();
     con.close();
   } catch (Exception e) {
    System.err.println("Exception: "+e.getMessage());
   }  }}
```

```
import java.util.*;
import java.sql.*;
public class OracleMultipleInserts {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:xe";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
      Statement sta = con.createStatement();
      int count = 0;
// insérer une ligne en utilisant les valeurs par défaut
     String SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName) VALUES (1, 'Moha', 'John')";
      count += sta.executeUpdate(SQLInsert);
// insérer une seule ligne dont on fournit les valeurs
      SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)"
                          + " VALUES (2, 'Manolo', 'Brahim', 999.99)";
      count += sta.executeUpdate(SQLInsert);
// insérer des lignes avec une boucle et des valeurs aléatoires
      Random r = new Random();
      for (int i=0; i<10; i++) {
                        Float points = 1000*r.nextFloat();
                        String firstName = Integer.toHexString(r.nextInt(9999));
                        String lastName = Integer.toHexString(r.nextInt(999999));
                        SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)"
                                + " VALUES (" + (i+10) + ", '" + firstName + "', '" + lastName + "', " + points + ")";
                        count += sta.executeUpdate( SQLInsert);
      }
      System.out.println("Nombre de lignes inserees: "+count);
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    } }}
```

Programme avec SGBD MySQL

```
import java.util.*;
import java.sql.*;
public class MySQLMultipleInserts {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("com.mysql.jdbc.Driver");
      String url = "jdbc:mysql://localhost:3306/testjdbc";
      con = DriverManager.getConnection(url,"moha","topsecret");
      Statement sta = con.createStatement();
      int count = 0;
// insérer une ligne en utilisant les valeurs par défaut
     String SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName) VALUES (1, 'Moha', 'John')";
      count += sta.executeUpdate(SQLInsert);
// insérer une seule ligne dont on fournit les valeurs
      SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)"
                              + " VALUES (2, 'Manolo', 'Brahim', 999.99)";
      count += sta.executeUpdate(SQLInsert);
// insérer des lignes avec une boucle et des valeurs aléatoires
      Random r = new Random();
      for (int i=0; i<10; i++) {
      Float points = 1000*r.nextFloat();
      String firstName = Integer.toHexString(r.nextInt(9999));
      String lastName = Integer.toHexString(r.nextInt(999999));
      SQLInsert = "INSERT INTO Profile(ID, FirstName, LastName, Point)"
           + " VALUES (" + (i+10) + ", '" + firstName + "', '"
                     + lastName + "', " + points + ")";
           count += sta.executeUpdate( SQLInsert);
      }
      System.out.println("Nombre de lignes inserees: "+count);
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }  }}
```

Extraction des lignes à partir d'une table "SELECT"

Pour exécuter une commande SELECT à partir d'un programme java, on suit à peu près les mêmes étapes que pour exécuter une commande LDD/LMD:

1. Effectuez les étapes 1. 2. 3. présentées pour exécuter une commande LDD/LMD
2. Création d'une String qui contient la commande SQL SELECT

   String SQLQuery = <Commande SELECT>
3. Exécution de la commande par la méthode **executeQuery()** de l'objet Statement, le résultat retourné doit être affecté à un objet **ResultSet**

   ResultSet res = sta.executeQuery(<commande SELECT>)
4. **Exploiter les lignes du résultat représenté par l'objet ResultSet**

```
import java.util.*;
import java.sql.*;
public class OracleExecuteQuery {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:xe";
      con = DriverManager.getConnection(url,"Moha","topsecret");
      Statement sta = con.createStatement();
// Extraction de toutes les lignes de la table Profile
      String SQLQuery = "SELECT * FROM Profile";
      ResultSet res = sta.executeQuery(SQLQuery);

      System.out.println("Liste des Profils: ");
      while (res.next()) {
        System.out.println(
          "  "+res.getInt("ID")
          + ", "+res.getString("FirstName")
          + ", "+res.getString("LastName")
          + ", "+res.getDouble("Point"));
      }
      res.close();
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    } }}
```

Résultat de l'exécution

```
java -cp .;ojdbc6.jar OracleExecuteQuery
iste des Profils:
 1, Moha, John, 0.0
 2, Manolo, Brahim, 999.99
 10, 3f6, 7e08e, 885.5092
 11, 200f, e82f8, 42.3851
 12, 1d93, 14bd7, 359.63333
 13, 182d, ae9ca, 881.9009
 14, 1287, 1061c, 218.53745
 15, bd, 2d2be, 805.2661
 16, 261c, 7ca6d, 523.52576
 17, 193, 533f0, 547.2503
 18, 249b, 6bbc6, 442.02423
 19, 109b, 6789f, 381.88998
```

Programme avec SGBD MySQL

```
import java.util.*;
import java.sql.*;
public class MySQLExecuteQuery {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("com.mysql.jdbc.Driver");
      String url = "jdbc:mysql://localhost:3306/testjdbc";
      con = DriverManager.getConnection(url,"moha","topsecret");
      Statement sta = con.createStatement();
// Extraction de toutes les lignes de la table Profile
     String SQLQuery = "SELECT * FROM Profile";
      ResultSet res = sta.executeQuery(SQLQuery);
      System.out.println("Liste des Profils: ");
      while (res.next()) {
        System.out.println("  "+res.getInt("ID") + ", "+res.getString("FirstName")
          + ", "+res.getString("LastName")  + ", "+res.getDouble("Point"));
      }
      res.close();
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
    }  }}
```

Modification des valeurs des colonnes d'une table "UPDATE"

Composez un programme Java qui permet de modifier le prenom et les points de Moha.

Le programme doit afficher les informations concernant Moha avant et après modification.

Supression des lignes d'une table "DELETE"

Composez un programme Java qui permet de supprimer tous les enregistrements dont l'ID est supérieur à 15.

Le programme doit afficher tous les enregistrements de la table avant et après suppression.

ORACLE - PreparedStatement

Cette partie du cours a pour objectif de mettre en oeuvre les **PreparedStatement (instructions préparées)** *de l'API JDBC avec le pilote JDBC d'Oracle.*
Nous traiterons dans l'ordre les sujets cidessous:

- **Vue générale sur PreparedStatement**
- **PreparedStatement avec Parametres**
- **PreparedStatement en Mode Batch**
- **Performance d'Insertion des lignes avec PreparedStatement**
- **Performance d'Insertion des lignes avec Statement**
- **Performance d'Insertion des lignes avec ResultSet**

Vue générale sur PreparedStatement

Lorsqu'une instruction SQL doit être exécutée plusieurs fois, il est plus efficace d'utiliser un objet PreparedStatement pour l'exécuter plutôt que d'utiliser un objet Statement.

La classe JDBC PreparedStatement possède les carastétiques suivantes:

- Les instructions SQL des objets PreparedStatement sont précompilées sur le serveur de base de données.
- Les paramètres IN sont supportés dans les instructions SQL des objets PreparedStatement.
- Le mode d'exécution en Batch est supporté pour exécuter une instruction SQL plusieurs fois dans une seule transaction.

Un objet PreparedStatement est créé par la méthode prepareStatement() d'un objet Connection et est exécuté comme un objet Statement normal.

Le programme exemple qui suit utilise un objet PreparedStatement pour exécuter une instruction SQL.

```
import java.sql.*;
public class OraclePreparedSelect {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");

// PreparedStatement pour une instruction SELECT
      String SQLQuery = "SELECT * FROM Profile WHERE ID = 2";
      PreparedStatement sta = con.prepareStatement(SQLQuery);

// Executer  PreparedStatement  comme requêté
      ResultSet res = sta.executeQuery();

// Extraire les valeur à partir values du ResultSet
      res.next();
      String firstName = res.getString("FirstName");
      String lastName = res.getString("LastName");
      System.out.println("User ID 2: "+firstName+' '+lastName);
// Fermer le  ResultSet , PreparedStatement et Connection
      res.close();
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
      e.printStackTrace();
    }  }}
```

Résultat de l'exécution:

```
C:\..>javac OraclePreparedSelect.java

C:\..>java -cp .;ojdbc14.jar  OraclePreparedSelect

User ID 2: Manolo Brahim
```

PreparedStatement avec Parametres

Pour rendre un objet PreparedStatement plus flexible, on ajoute des paramètres à l'instruction SQL intégrée en utilisant les points d'interrogation (?).

Les valeurs réelles doivent être attribuées aux paramètres avant d'exécuter l'objet PreparedStatement.

L'attribution des valeurs aux paramètres d'un PreparedStatement, se fait par l'appel des méthodes setXXX() selon le format :

<PreparedStament>.setXXX(<rang du paramètre>, <valeur>);

Le premier point d'interrogation (?) qui représente le 1er paramètre a le rang 1, le 2ème a le rang 2, …etc.
En supposant que ps est un objet preparedStatement.

```
Ps.setXXX(1, v1);
Ps.setXXX(2, v2);
……….
Ps.setXXX(n, vn);
```

PreparedStatement avec Parametres

JDBC supporte plusieurs méthodes setXXX(), une pour chaque type de données Java, par conséquent on peut attribuer des valeurs directement aux paramètres avec le type de données souhaité de Java sans aucune conversion.
Le tableau ci-dessous fournit la liste de ces méthodes setXXX().

Méthode	**Type**		**Méthode**	**Type**
setArray()	Tableau		setDate()	java.sql.Date
setAsciiStream()	Flux de caractères ASCII		setDouble()	double
setBigDecimal()	java.sql.BigDecimal		setFloat()	float
setBinaryStream()	Flux d'octets		setInt()	int
setByte()	byte		setLong()	long
setBlob()	Blob		setNull()	null
setBoolean()	boolean		setObject()	Object
setBytes()	Byte[]		setRef()	Ref
setCharacterStream()	java.io.Reader		setShort()	short
setClob()	Clob		setString()	String
setTimestamp()	java.sql.Timestamp		setTime()	java.sql.Time

```
import java.sql.*;
public class OraclePreparedStatementParameter {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
     String url = "jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
// PreparedStatement pour une instruction SELECT avec un parametere
      String SQLQuery = "SELECT * FROM Profile WHERE ID = ?";
                PreparedStatement sta = con.prepareStatement(SQLQuery);
// Attribution d'une valeure au parametere
      int id = 19;
      sta.setInt(1,id);
// Executer le PreparedStatement en tant que requête
      ResultSet res = sta.executeQuery();
// Obtenir les valeurs à partir du  ResultSet
      res.next();
      String firstName = res.getString("FirstName");
      String lastName = res.getString("LastName");
      System.out.println("User ID "+id+": "+firstName+' '+lastName);

      res.close();
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
      e.printStackTrace();
    } }}
```

Résultat de l'exécution:

```
C:\..>javac OraclePreparedStatementParameter.java
C:\..>java -cp .;ojdbc14.jar OraclePreparedStatementParameter
User ID 19: 1781 8e2a6
```

PreparedStatement en Mode Batch

Pour exécuter un objet PreparedStatement plusieurs fois dans une seule transaction, on peut utiliser le mode batch de PreparedStatement.

Chaque appel de la méthode addBatch(), crée une copie de l'instruction SQL intégrée.

Toutes les copies de l'instruction ne sont exécutées que lors de l'appel de la méthode d'exécution executeBatch().

Une séquence possible des appels de méthodes pour exécuter un objet PreparedStatement en mode batch est:

```
ps.setXXX(...); // attribuer les valeurs aux paramètres de la 1ère copie
…
ps.addBatch();  // Créer la 1ère copie de l'instruction SQL

ps.setXXX(...); // attribuer les valeurs aux paramètres de la 2ème copie
…
ps.addBatch();  // Créer la 2ème copie de l'instruction SQL

ps.setXXX(...); // attribuer les valeurs aux paramètres de la 3ème copie
…
ps.addBatch();  // Créer la 3ème copie de l'instruction SQL

ps.executeBatch(); // Executer toutes les copies ensemble comme un batch
```

Le programme exemple suivant illustre tout cela:

```
import java.sql.*;
public class OraclePreparedStatementBatch {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
     String url = "jdbc:oracle:thin:@PC-de-Hassouni:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");

// Obtention d'un objet PreparedStatement
   String SQLInsert = "INSERT INTO Profile (ID, FirstName, LastName) VALUES (?, ?, ?)";
      PreparedStatement ps = con.prepareStatement(SQLInsert);

// Fournir les valeurs des paramètre de la 1ère copie
      ps.setInt(1,101);
      ps.setString(2, "ABCD");
      ps.setString(3,"First");

// Créer la 1ère copie
      ps.addBatch();

// Fournir les valeurs des paramètre de la 2ème copie
      ps.setInt(1,102);
      ps.setString(2, "EFGH");
      ps.setString(3,"Second");

// Créer la 2ème copie
      ps.addBatch();
```

```
// Fournir les valeurs des paramètre de la 3ème copie
     ps.setInt(1,103);
     ps.setString(2, "IJKL");
     ps.setString(3,"Third");

// Créer la 3ème copie
     ps.addBatch();

// Fournir les valeurs des paramètre de la 4ème copie
     ps.setInt(1,104);
     ps.setString(2, "MNOP");
     ps.setString(3,"Last");

// Créer la 4ème copie
     ps.addBatch();

// Executer toutes les 4  copies
     int[] counts = ps.executeBatch();

     int count = 0;
     for (int i=0; i<counts.length; i++) {
                System.out.println(" " + i + " : " + counts[i]);
                  count += counts[i];   }
     System.out.println("Total des lignes inserees: "+count);
     ps.close();
     con.close();
   } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
     e.printStackTrace();
   }  }}
```

```
C:\...>javac OraclePreparedStatementBatch.java

C:\...>java -cp .;ojdbc14.jar OraclePreparedStatementBatch
 0 : -2
 1 : -2
 2 : -2
 3 : -2
Total des lignes inserees: -8
```

Performance d'Insertion des lignes avec PreparedStatement

L'exécution des instructions SQL en utilisant les objets PreparedStatement est supposé être plus rapide qu'en utilisant les objets Statements. Le programme qui suit calcule le temps nécessaire pour insérer 10000 lignes dans une table vide en utilisant un objet PreparedStatement.

Résultat de l'exécution

```
C:\...>javac OraclePerformancePreparedStatement.java

C:\...>java -cp .;ojdbc14.jar OraclePerformancePreparedStatement
Temps ecoule pour inserer 10000 lignes avec PreparedStatement 11733 milliseconds
```

```
import java.sql.*;
public class OraclePerformancePreparedStatement {
  public static void main(String [] args) {
    Connection con = null;
    try {
       // Chargement du Driver Oracle
      Class.forName("oracle.jdbc.driver.OracleDriver");

// Obtention d'un objet Connection
      String url = "jdbc:oracle:thin:@localhost:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");

// Suppression de tous les enregistrements de la table
      Statement sta = con.createStatement();
      sta.executeUpdate("DELETE FROM Profile");

// Début du test
     int count = 10000;
     long t1 = System.currentTimeMillis();
```

```
// PreparedStatement pour insérer des lignes
     PreparedStatement ps = con.prepareStatement(
               "INSERT INTO Profile (ID, FirstName, LastName)"
               + " VALUES (?, ?, ?)");
     java.util.Random r = new java.util.Random();
     for (int i = 0; i < count; i++) {
               ps.setInt(1,i+1);
               ps.setString(2,Integer.toHexString(r.nextInt(9999)));
               ps.setString(3,Integer.toHexString(r.nextInt(999999)));
               ps.executeUpdate();
     }
     ps.close();

// Fin du test
     long t2 = System.currentTimeMillis();
     System.out.println("Temps ecoule pour inserer "+count
        +" lignes avec PreparedStatement "+(t2 -t1) +" milliseconds");

     con.close();
    } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
     e.printStackTrace();
    }  }}
```

Performance d'Insertion des lignes avec Statement

Le programme qui suit calcule le temps nécessaire pour insérer 1000 lignes dans une table vide en utilisant un objet Statement.

Résultat de l'exécution

```
C:\...>javac OraclePerformanceRegularStatement.java

C:\...>java -cp .;ojdbc14.jar  OraclePerformanceRegularStatement
Temps ecoule pour inserer  10000 lignes avec Statement : 19017 milliseconds
```

```
import java.sql.*;
public class OraclePerformanceRegularStatement {
  public static void main(String [] args) {
    Connection con = null;
    try {
// Chargement du Driver Oracle
     Class.forName("oracle.jdbc.driver.OracleDriver");

// Obtention d'un objet Connection
                String url = "jdbc:oracle:thin:@localhost:1521:orcl";
     con = DriverManager.getConnection(url,"Moha","TopSecret");

// Suppression de toutes les lignes de la table
     Statement sta = con.createStatement();
     sta.executeUpdate("DELETE FROM Profile");

// Début du test
    int count = 10000;
    long t1 = System.currentTimeMillis();
```

```
// Insertion des lignes par un objet  Statement
     Statement rs = con.createStatement();
     java.util.Random r = new java.util.Random();
     for (int i = 0; i < count; i++) {
       rs.executeUpdate(
         "INSERT INTO Profile (ID, FirstName, LastName) VALUES ("
         +(i+1)+", '"+Integer.toHexString(r.nextInt(9999))
         +"', '"+Integer.toHexString(r.nextInt(999999))+"')");
     }
     rs.close();

// Fin du test
     long t2 = System.currentTimeMillis();
     System.out.println("Temps ecoule pour inserer  "+count
        +" lignes avec Statement : "+(t2 -t1) +" milliseconds");

     con.close();
    } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
     e.printStackTrace();
    }  }}
```

```
mport java.sql.*;
import java.text.SimpleDateFormat;
public class OraclePreparedStatementParameterUpdate {
  public static void main(String [] args) {
    Connection con = null;
    try {
      // Chargement du Driver Oracle
      Class.forName("oracle.jdbc.driver.OracleDriver");
// Obtention d'un objet Connection
                    String url = "jdbc:oracle:thin:@localhost:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
// PreparedStatement pour une instruction SELECT avec un parametere
      String SQLUpdate = " UPDATE Profile SET BirthDate = ? WHERE ID = ?";
                    PreparedStatement sta = con.prepareStatement(SQLUpdate);
// Attribution d'une valeure aux parameteres
    SimpleDateFormat sdf = new SimpleDateFormat( "dd/MM/yyyy" ) ;
     Date birthd = new Date(sdf.parse("23/12/2008").getTime());
      sta.setDate(1, birthd);
      int id = 19;
      sta.setInt(2,id);
// Executer le PreparedStatement
      int count = sta.executeUpdate();
// Fermer les objets   PreparedStatement et connection
      sta.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
      e.printStackTrace();
    }  }}
```

ORACLE - JDBC CallableStatement

Il est possible d'exécuter les procédures stockées écrites en PL/SQL et sauvegardées sur le serveur à partir des programmes Java en utilisant l'objet JDBC **CallableStatement**. Dans ce qui suit nous traiterons les sujets ci-dessous:

- **Vue générale sur les objets CallableStatement**
- **"CREATE PROCEDURE" – Creation d'une simple Procédure Stockée.**
- **Creation d'une procédure avec des parametres IN et OUT**
- **Creation des objets CallableStatement avec prepareCall()**
- **Creation des objets CallableStatement avec des Parametres**
- **getProcedures() – Obtention de la liste des Procedures Stockées**

• Vue générale sur les objets CallableStatement

Les ojets JDBC CallableStatement sont utilisés pour faire des appels aux procédures stockées dans le serveur SGBDR.

Pour utiliser de façon efficace les objets CallableStatement vous devez retenir un ensemble de choses:

1. Un objet CallableStatement est crée en appelant la méthode prepareCall() d'un objet Connection comme suit:

```
// Appel d'une procedure stockée sans paramètres
CallableStatement cs = con.prepareCall("CALL Abc()");

// Appel d'une procédure stockée avec des paramètres
CallableStatement cs = con.prepareCall("CALL Abc(v1, v2, ...)");
```

2. Il est possible d'utiliser les point d'interrogation (?) dans l'instruction d'appel pour marquer la place d'un paramètre de la même manière qu'avec un objet PreparedStatement.
Les paramètres OUT (output), INOUT (input and output), et return doivent être représentés par des points d'interrogation (?):

```
// Appel d'une procédure avec des marqueurs ? de parametres
CallableStatement cs = con.prepareCall(    "CALL Abc(v1, v2, ?, ?, ...)");

// Appel d'une procédure qui a une valeur de retour
CallableStatement cs = con.prepareCall(    "? = CALL Abc(v1, v2, ?, ?, ...)");
```

Ce dernier format "? = CALL Abc(v1, v2, ?, ?, ...)" ne s'applique pas à Oracle, puisque les procédures Oracle ne retournent pas de valeurs.

3. Si un paramètre IN est représenté par un ?, sa valeur doit être fournie par une méthode setXXX() avant l'exécution de l'objet CallableStatement.
Les méthodes setXXX() fonctionnent de la même manière qu'avec les objets PreparedStatement.

4. Les marqueurs de place (?) pour les paramètres OUT, INOUT et return doivent être enregistrés comme des paramètres avec des types de données JDBC avant l'exécution de l'objet CallableStatement:

// Enregistrement du premier marqueur comme un paramètre output INTEGER
cs.registerOutParameter(1, java.sql.Types.INTEGER);

// Enregistrement du 4ème marqueur de place comme un paramètre output DATE
cs.registerOutParameter(4, java.sql.Types.DECIMAL);

5. Si la procédure stockée ne retourne aucun objet ResultSet, l'objet CallableStatement doit être exécuté en appelalnt la méthode executeUpdate().
// Execution de l'objet CallableStatement: cs.executeUpdate();

6. Si la procédure stockée retourne un ou plusieurs objets ResultSet, l'objet CallableStatement doit être exécuté en appellant la méthode executeQuery(). Le résultat retourné est le premier objet ResultSet, qui représente le résultat de la première requête exécutée par la procédure stockée.
// Execution de l'objet CallableStatement ResultSet res = cs1.executeQuery();

7. Un objet CallableStatement peut aussi être exécuté en mode batch d'une manière similaire à l'objet PreparedStatement:

```
// Création d'une première copie de l'instruction Call
cs.setXXX(...);
cs.addBatch();

// Création d'une autre copie de l'instruction Call
cs.setXXX(...);
cs.addBatch();

// Execution de toutes les copies de l'instruction call en mode batch
cs.executeBatch();
```

8. Après l'exécution de l'l'instruction Call, toutes les lignes de tous les objets ResultSet retournés doivent être traitées avant de retrouver n'importe quel paramètre output.

```
// Execution de l'instruction Call
ResultSet res = cs.executeQuery();
while (res.next()) {    ...  }
```

9. Après avoir traités tous les objets ResultSet, les paramètres output enregistrés peuvent être retrouvés en utilisant les méthodes getXXX() :

```
// Retrouver les paramètres output enregistrés
int first = cs.getInt(1);
Date four = cs.getDate(4);
```

"CREATE PROCEDURE"

Creation d'une simple Procédure Stockée.

Pour tester l'objet CallableStatement, nous fournissons ci-dessous un script PL/SQL pour créer une procédure stockée trés simle qui permet de modifier la valeur du champ Point de l'enregistrement qui correspond à "MOHA"de la table Profile:

```
>CREATE OR REPLACE PROCEDURE UpdatePoint IS
  BEGIN
    UPDATE Profile SET Point = Point+1
  WHERE FirstName LIKE 'MOHA';
  END;
/
```

```
--Tester la valeur du champ avant appel de la procédure stockée
select * from profile where firstname like 'MOHA';

--Appel de la procédure stockée
CALL UpdatePoint();

--Tester la valeur du champ après appel de la procédure stockée
select * from profile where firstname like 'MOHA';
```

Creation d'une procédure avec des parametres IN et OUT

Pour faire davantage de test avec l'objet CallableSattement, nous fournissons un autre script PL/SQL qui crée une procédure stockée avec des paramètres IN et OUT :

```
--Création d'une procédure stockée avec des paramètres IN et OUT
CREATE PROCEDURE CountChar(Chaine IN CHAR, Longueur OUT INTEGER) AS
BEGIN
  Longueur := LENGTH(Chaine);
END;
/
```

```
-- Permettre l'affichage avec  PUT_LINE()
 SET SERVEROUTPUT ON;
```

```
-- Tester la procédure avec un bloc anonyme
DECLARE
C INTEGER;
BEGIN
  CountChar('MOHA John',C);
  DBMS_OUTPUT.PUT_LINE('Count: ' || C);
END;
/
```

Creation des objets CallableStatement avec prepareCall()

Le programme qui suit montre crée un objet CallableStatement pour appeler la procédure stockée UpdatePoint créée précédemment

```
import java.sql.*;
public class OracleCallableStatement {
  public static void main(String [] args) {
    Connection con = null;
    try {
       // Chargement du Driver Oracle
     Class.forName("oracle.jdbc.driver.OracleDriver");

// Obtention d'un objet Connection
     String url = "jdbc:oracle:thin:@localhost:1521:orcl";
     con = DriverManager.getConnection(url,"Moha","TopSecret");

// Creation d'un objet CallableStatement
     CallableStatement cs = con.prepareCall("CALL UpdatePoint()");

// Execution de la procédure
     cs.execute();
```

```
// fermer l'objet CallableStatement resource
     cs.close();

// Tester le résultat
     Statement sta = con.createStatement();
     ResultSet res = sta.executeQuery(
       "SELECT * FROM Profile WHERE FirstName LIKE ‘Moha'");
     while(res.next()){
        System.err.println("Point: "+res.getInt("Point"));
                }
//Fermeture des objets ressources
    res.close();
     sta.close();
     con.close();
   } catch (Exception e) {
     System.err.println("Exception: "+e.getMessage());
     e.printStackTrace();
   }  }}
```

```
C:\..>javac OracleCallableStatement.java

C:\..>java -cp .;ojdbc14.jar OracleCallableStatement
Point: 7
Point: 7

C:\..>java -cp .;ojdbc14.jar OracleCallableStatement
Point: 8
Point: 8
```

Creation des objets CallableStatement avec des Parametres

Lors de la création d'un objet CallableStatement, pour appeler une procédure qui a un paramètre IN, il est possible de mettre une valeur statique, ou un marqueur de place (?).

Pour fournir une valeur au marqueur de place, il faut utiliser la méthode setXXX().

Pour un paramètre OUT définie dans la procédure, nous devons mettre un marqueur de place qui doit être enregistré avec la méthode registerOutParameter().

Nous avons définie précédemment la procédure CountChar(), avec un pramètre IN et un paramètre OUT. Le programme qui suit vous montre comment créer un objet CallableStatement pour exécuter une telle procédure.

```
C:\...>javac OracleCallParameter.java

C:\...>java -cp .;ojdbc14.jar OracleCallParameter
Chaine fournie: MOHA John
longueur de la chaine: 9
```

```
import java.sql.*;
public class OracleCallParameter {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
// Create CallableStatement
      CallableStatement cs = con.prepareCall(
        "CALL CountChar(?,?)");
// Attibuer une valeur pour le paramètre IN
      String chaine = "MOHA John";
      cs.setString(1,chaine);
// Enregister  le paramètre OUT
      cs.registerOutParameter(2, java.sql.Types.INTEGER);
// Executer l'instruction  CALL
      cs.execute();
// Retrouver la valeur du paramètre OUT
      int length = cs.getInt(2);
      System.out.println("Chaine fournie: " + chaine);
      System.out.println("longueur de la chaine: "+length);
// Close resource
      cs.close();
      con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
      e.printStackTrace();
    }  }}
```

getProcedures()–Obtention de la liste des Procedures Stockées

Le programme qui suit permet d'obtenir la liste des procédures et des tables stockées dans un schéma, en utilisant la méthode getProcedures() d'un objet DatabaseMetaData.

```
C:\...>javac OracleCallGetProcedures.java

C:\...>java -cp .;ojdbc14.jar  OracleCallGetProcedures
Stored procedures:
   null, MOHA, COUNTCHAR
   null, MOHA, UPDATEPOINT
Stored tables:
   null, MOHA, PROFILE
```

```
import java.sql.*;
public class OracleCallGetProcedures {
  public static void main(String [] args) {
    Connection con = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      String url = "jdbc:oracle:thin:@localhost:1521:orcl";
      con = DriverManager.getConnection(url,"Moha","TopSecret");
DatabaseMetaData meta = con.getMetaData();
// Liste des procedures stockées
      ResultSet res = meta.getProcedures(null, "MOHA", "%");
      System.out.println("Stored procedures:");
      while (res.next()) {
        System.out.println( "   "+res.getString("PROCEDURE_CAT« )  + ", "+res.getString("PROCEDURE_SCHEM")
          + ", "+res.getString("PROCEDURE_NAME"));
      }
      res.close();
// Liste des tables
      res = meta.getTables(null, "MOHA", "%", null);
      System.out.println("Stored tables:");
      while (res.next()) {
        System.out.println( "   "+res.getString("TABLE_CAT« )  + ", "+res.getString("TABLE_SCHEM")
          + ", "+res.getString("TABLE_NAME"));
      }
      res.close();     con.close();
    } catch (Exception e) {
      System.err.println("Exception: "+e.getMessage());
      e.printStackTrace();
    }  }}
```