Cours 11 : JDBC

Java DataBase Connectivity

Réaliser par :
Gouiferda Soufiane

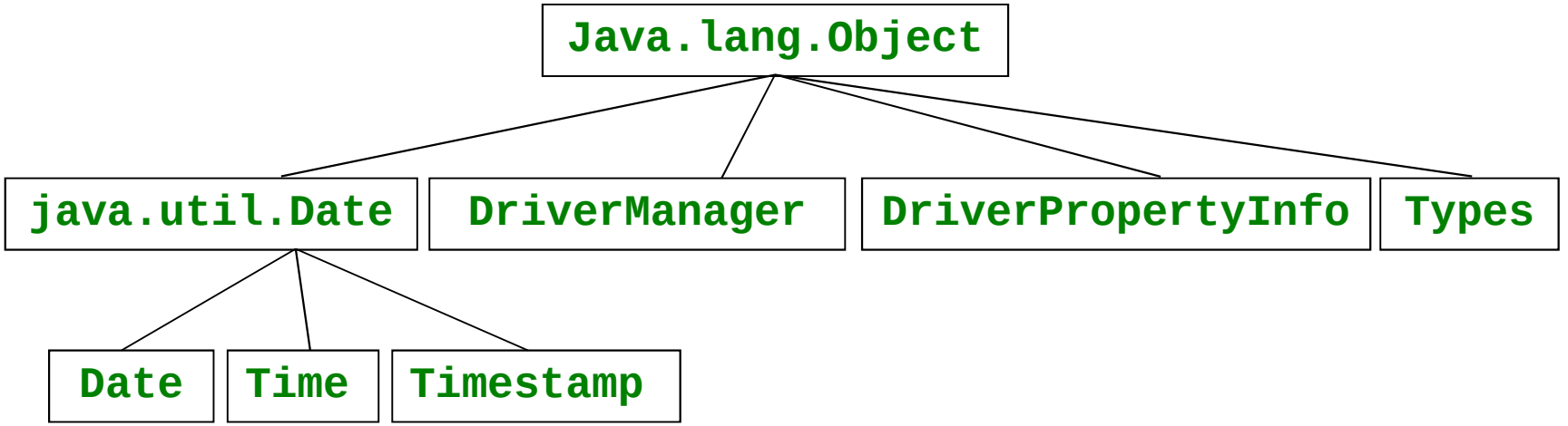
Encadré par :
M. Mhamdi Ahmed

JDBC

- Une API (Application Programming Interface) qui permet d'exécuter des instructions SQL
- JDBC fait partie du JDK (Java Development Kit)
- Toutes les classes et interfaces sont dans le package *java.sql* :

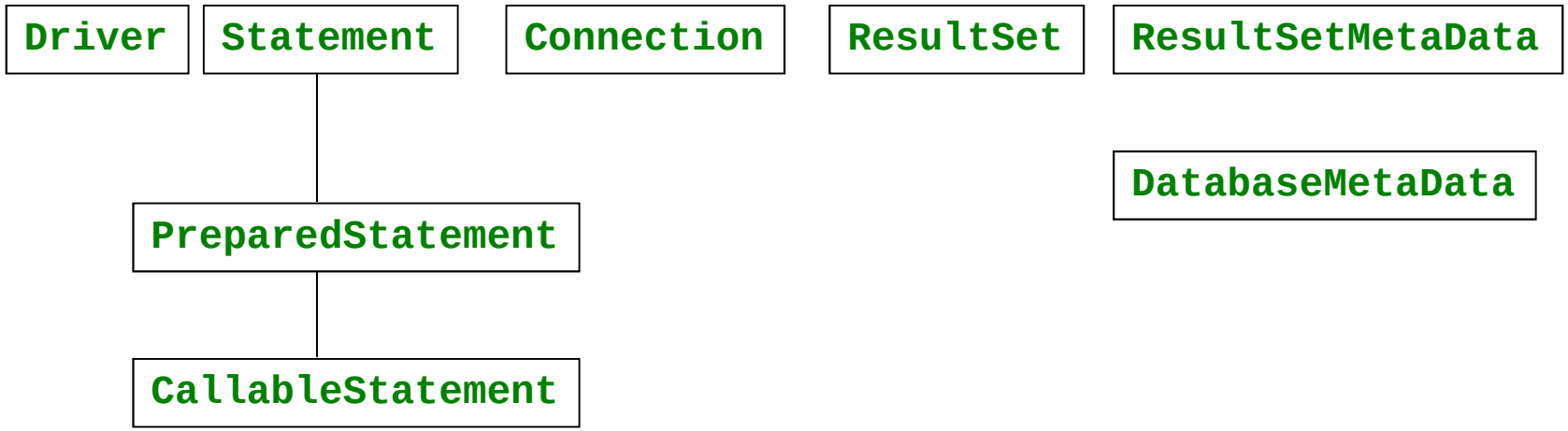
```
import java.sql.*
```

JDBC



Les classes du package *java.sql*

JDBC



Les interfaces du package *java.sql*

JDBC : structure d'un programme

1. On établit une connexion avec une source de données
2. On effectue des requêtes
3. On utilise les données obtenues pour des affichages, des traitements statistiques etc.
4. On met à jour les informations de la source de données
5. On termine la connexion
6. Eventuellement, on recommence en 1

1. Connexion

- Pour établir une connexion à une base de données, il faut :
  1. Connaître son nom
  2. Lui associer un pilote (ou driver)

- Pour charger l'adresse des pilotes :

```
EXPORT CLASSPATH =
    nomFichier:.:$CLASSPATH
```

Déclaration du driver

- Pour charger un driver, on peut utiliser la méthode

```
Class.forName(String)
```

- Pilote utilisé à l'IUT :

```
oracle.jdbc.driver.OracleDriver
```

Exemple (jusqu'ici)

```
import java.sql.*;
public class premiereConnexion {
 public static void main(String[] args){
  try{
   Class.forName("oracle.jdbc.driver.OracleDriver");
  }
  catch (ClassNotFoundException e){
   System.out.println(
    "Impossible de charger le pilote");
   System.exit(1);
  }
 System.out.println("Pilote chargé");
 }
}
```

Nommage des bases de données : les URL

- ***<protocole>:<sous-protocole>:<compléments>***
  - *Protocole* : **`jdbc`**
  - *Sous-protocole* : le driver. Tous les drivers distribués par Oracle commencent par **`oracle:`** (**`oracle:thin`**, **`oracle:oci7`**, **`oracle:oci8`**, …).
    A l'IUT, on utilise **`oracle:thin`**
  - *Compléments* identifie la base de données :
    login/motDePasse@ordinateur:port:base
  - Exemple :

```
String url = "jdbc:oracle:thin:ffioren/mdp@
    r2d2.etu.iut-orsay.fr:1521:etudom"
```

Etablir la connexion

- On utilise la méthode **`getConnection()`** de la classe **`DriverManager`** avec comme argument l'URL :

```
Connection maConnexion =
        DriverManager.getConnection(url);
```

ou bien

```
Connection maConnexion =
   DriverManager.getConnection(url,"ff
   ioren","mdp");
```

Exemple (suite)

```
  ...
  catch (ClassNotFoundException e){ ... }
  String url = "jdbc:oracle:thin:ffioren/mdp";
  url+="@r2d2.etu.iut-orsay.fr:1521:etudom";
  Connection maConnexion = null;
  try{
   maConnexion = DriverManager.getConnection(url);
  }
  catch (SQLException e){
   System.out.println(
    "Impossible de se connecter à l'url : "+url);
   System.exit(1);
  }
 ...}
}
```

2. Déclaration et exécution de la requête

- Il faut tout d'abord demander la création du statement

```
Statement monInstruction =
                    maConnexion.createStatement();
```

- Ensuite il faut déclarer le code SQL de la requête

```
ResultSet monResultat =
monInstruction.executeQuery(maRequête);
```

Exemple (suite)

```
public static void main(String[] args)
                                        throws SQLException{
 ...
 Connection maConnexion =
                    DriverManager.getConnection(url);
 Statement monInstruction =
                         maConnexion.createStatement();
 ResultSet monResultat =
                           monInstruction.executeQuery(
 "select numfilm, titre from ens2004.film where
  titre like 'A%'");
 ...}
}
```

3. Exploitation des résultats

- On peut parcourir les lignes de l'objet **`ResultSet`** avec la méthode **`next()`**. Cette méthode renvoie **`VRAI`** s'il reste des lignes à lire et **`FAUX`** sinon :

```
while(monResultat.next()){
    traitement des données récupérées
    }
```

Exemple (suite)

```
...
ResultSet monResultat =
                        monInstruction.executeQuery(
"select numfilm, titre from ens2004.film where …");
while(monResultat.next()){
  String titre = monResultat.getString("titre");
  int numero = monResultat.getInt("numFilm");
  System.out.print(numero + "\t" + titre);
  System.out.println(" ");
}
...}}
```

Exemple (suite)

```
...
ResultSet monResultat =
                        monInstruction.executeQuery(
"select numfilm, titre from ens2004.film where …");
while(monResultat.next()){
  String titre = monresultat.getString(2);
  int numero = monresultat.getInt(1);
  System.out.print(numero + "\t" + nom);
  System.out.println(" ");
}
...}}
```

Exemple (suite)

```
...
ResultSet monResultat =
                        monInstruction.executeQuery(
"select numfilm, titre from ens2004.film where …");
while(monResultat.next()){
  System.out.print(monresultat.getInt(1) + "\t"
  + monresultat.getString(2));
  System.out.println(" ");
}
...}}
```

Cas des valeurs nuls

- Un **`null`** Java peut être renvoyé par les méthodes qui retournent un objet : **`getString()`**, **`getObject()`**, **`getDate()`**, …
- Un **`0`** peut être renvoyé par les méthodes qui retournent une valeur numérique : **`getInt()`**, **`getByte()`**, **`getShort()`**, **`...`**
- Une valeur **`FAUX`** peut être renvoyée par la méthode **`getBoolean()`**.
- **`wasNull()`** : méthode de **`ResultSet`**
- **`isNullable()`** : méthode de **`ResultSetMetaData`**

4. Accès en mise à jour

- En outre de l'exécution d'un **`select`** avec **`executeQuery,`** on peut aussi exécuter un **`update`**, un **`insert`** ou un **`delete`** :

```
Statement monInstruction =
                    maConnexion.createStatement();

int monResultat =
monInstruction.executeUpdate(maRequête);
```

Exemple (suite)

```
...
while(monResultat.next()){ ... }
...
int num = 1112;
String client = "charlie";
...
int nbLignes = monInstruction.executeUpdate(
"INSERT INTO location(numExemplaire, login,
dateLocation) VALUES("+num+",'"+client
+"',SYSDATE)");
System.out.println(nbLignes + " ligne(s)
inserée(s).");
...}}
```

Gestion des transactions

- Par défaut, **`autocommit on`**
- Pour modifier ce mode on appel la méthode **`setAutoCommit()`** de la classe **`Connection`** (paramètres **`true`** et **`false`**)
- Pour valider ou annuler on appel les méthodes **`commit()`** et **`rollback()`** de la classe **`Connection`**

5. Déconnexion

- Libérer les objets **`ResultSet`** et **`Statement`** :

```
monObjet.close();
```

- Fermer la connexion :

```
maConnexion.close();
```

Exemple (suite et fin)

```
... ResultSet monResultat = ...
... while(monResultat.next()){ ... } ...
monResultat.close();
...
int num = 1112;
String client = "charlie";
...
System.out.println(nbLignes+" ligne(s)inserée(s).");
monInstruction.close();
maConnexion.close();
}//main
}//premiereConnexion
```

Interface *ResultSetMetaData*

- Elle fournit des informations concernant les types et les propriétés des colonnes d'une instance de **`ResultSet`**
  - Combien d'attributs contient le **`ResultSet`** ?
  - Les noms des attributs sont-ils sensibles à la casse ?
  - Est-il possible de rechercher des données dans la colonne de son choix ?
  - Est-il possible d'affecter la valeur **`NULL`** à un attribut ?
  - Quel est le nombre maximal de caractères affichables pour un attribut donné ?
  - Quel est le nom d'un attribut ?
  - A quelle table appartient un attribut ?
  - De quel type est un attribut ?

Contenu de l'interface *ResultSetMetaData*

- `getCatalogName()`
- `getColumnClassName()`
- `getColumnCount()`
- `getColumnDisplaySize()`
- `getColumnLabel()`
- `getColumnName()`
- `getColumnType()`
- `getColumnTypeName()`
- `getPrecision()`
- `getScale()`
- `getSchemaName()`
- `getTableName()`
- `isAutoIncrement()`
- `isCaseSensitive()`
- `isCurrency()`
- `isDefinitelyWritable()`
- `isNullable()`
- `isReadOnly()`
- `isSearchable()`
- `isSigned()`
- `isWritable()`

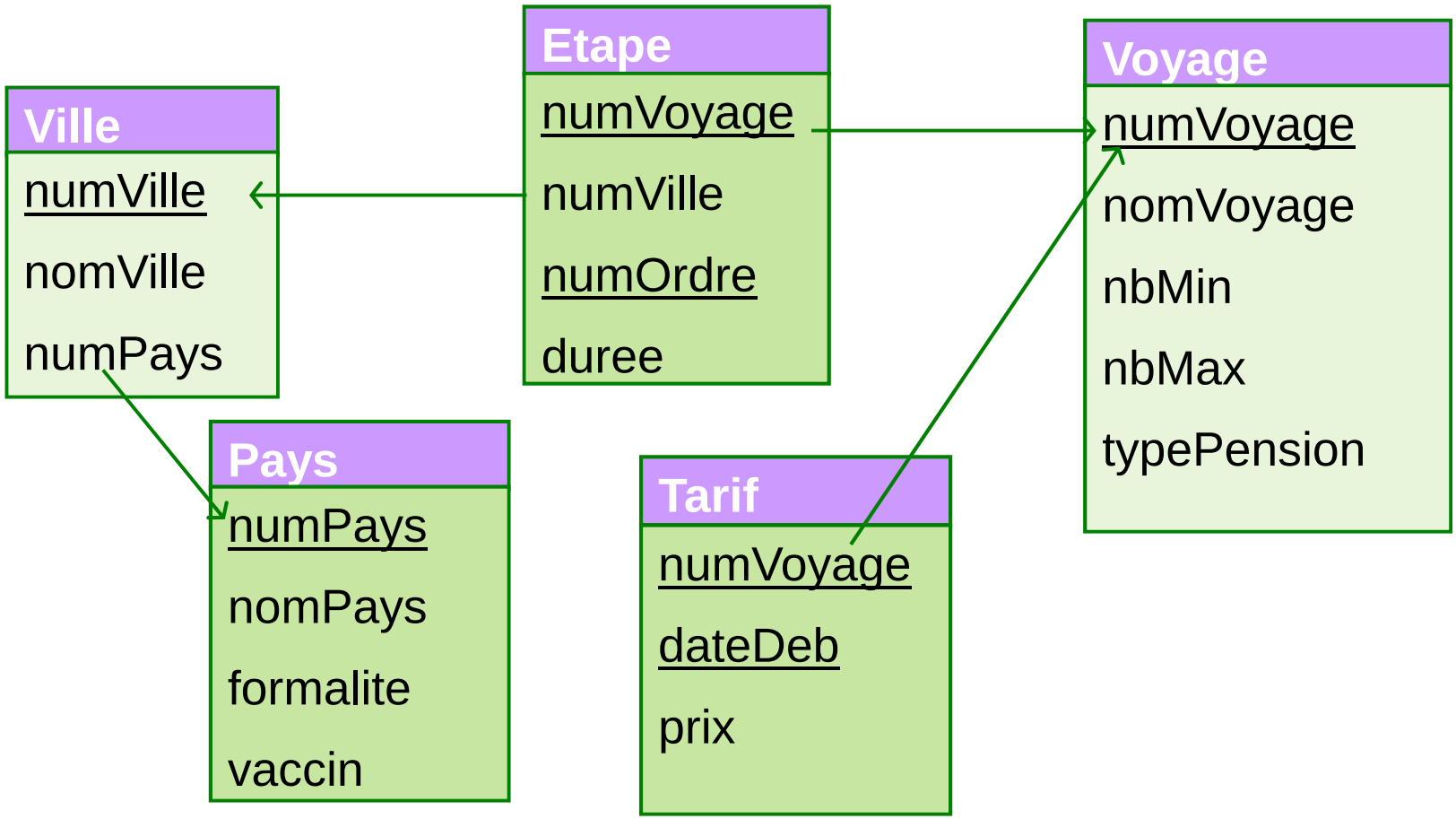
Exemple

```
ResultSet monResultat =
 monInstruction.executeQuery("select numfilm,
 titre from ens2004.film");
ResultSetMetaData rsmd = monResultat.getMetaData();
int nombreDeColonnes = rsmd.getColumnCount();
for(int i=1; i<=nombreDeColonnes; i++){
   String nomColonne = rsmd.getColumnName(i);
   String nomType = rsmd.getColumnTypeName(i);
   System.out.println("La colonne "+i+" est  "
   +nomColonne+" dont le nom de type Oracle est "
   +nomType);
}
```

Interface *PreparedStatement*

- Elle permet de transférer une seule fois le code SQL d'une requête au serveur de façon à en accélérer l'exécution lorsqu'on veut répéter plusieurs fois cette requête (un seule « compilation » et un seul calcul d'un plan de requête)
  - Nom de classe : **`java.sql.PreparedStatement`**
  - Classe mère : **`java.sql.Statement`**
  - Sous-classe directe : **`java.sql.CallableStatement`**
  - Disponible depuis la version 1.1 du JDK

Exemple


Ville
numVille
nomVille
numPays
Etape
numVoyage
numVille
numOrdre
duree
Voyage
numVoyage
nomVoyage
nbMin
nbMax
typePension
Pays
numPays
nomPays
formalite
vaccin
Tarif
numVoyage
dateDeb
prix

Exemple (2)

```
PreparedStatement pst =
maConnexion.prepareStatement(
"select numvoyage from tarif
 where datedeb > '01-07-2015'
 and prix < 1000");
ResultSet monResultat = pst.executeQuery();
while(monResultat.next()){
    int numero = monresultat.getInt(1);
    System.out.println(numero);
}
...
monResultat = pst.executeQuery();
```

Exemple (3)

```
PreparedStatement pst =
maConnexion.prepareStatement(
"select numvoyage from tarif
 where datedeb > ?
 and prix < ?");
pst.setString(1, "01-07-2015");
pst.setInt(2, 1000);
ResultSet monResultat = pst.executeQuery();
while(monResultat.next()){
    ...
}
```

Exemple (en mise à jour)

```
PreparedStatement pst =
maConnexion.prepareStatement(
 "update tarif set prix = prix + ? where
  numvoyage = ? and datedeb > ?");
for (int i = 0; i<10; i++){//10 fois update
   pst.setInt(2,saisieVoyage());
   pst.setInt(1,saisieMajoration());
   pst.setString(3,saisieDate());
   int compteur = pst.executeUpdate();
   System.out.println(compteur + " ligne(s)
   mise(s) à jour.");
}
pst.close();
```

PreparedStatement – Création

```
PreparedStatement nomstatement =
maconnexion.prepareStatement(requête)
```

PreparedStatement – Gestion des paramètres

- Paramètrage :
  ***nomstatement.*`setXXX`(*rang*, *valeur*)**
  - Exemples : **`setAsciiStream(), setBigDecimal(), setBinaryStream(), setBoolean(), setByte(), setBytes(), setDate(), setDouble() setFloat(), setInt(), setLong(), setNull(), setObject(), setShort(), setString(), setTime(), setTimestamp(), setUnicodeStream()`**

PreparedStatement – Exécution

- **`executeQuery()`** pour une requête
- **`executeUpdate()`** pour tous les autres ordres SQL

Interface *CallableStatement*

- Elle permet de faire appel à une procédure ou à une fonction stockée dans la base de donnée

Exemple

- Soit la fonction définie par

```
CREATE OR REPLACE FUNCTION entitule
(numero IN voyage.numVoyage%TYPE)
RETURN voyage.nomVoyage%TYPE IS
nom voyage.nomVoyage%TYPE
BEGIN select nomvoyage into nom
FROM voyage where numvoyage = numero;
return nom; END;
/
```

Exemple (2)

- Soit la fonction définie par

```
CREATE OR REPLACE FUNCTION entitule
(numero IN voyage.numVoyage%TYPE)
RETURN voyage.nomVoyage%TYPE IS
...
```

- Donc **numero** sera de type **INTEGER** et la fonction renverra du **VARCHAR2(60)**

Exemple (appel à la fonction)

```
CallableStatement cst =
 maConnexion.prepareCall(
 "{? = call entitule(?)}");
cst.setInt(2,6);
cst.registerOutParameter(
 1,java.sql.Types.VARCHAR);
boolean retour = cst.execute();
String nom = cst.getString(1);
...
```

CallableStatement – Création

```
CallableStatement nomstatement =
    maconnexion.prepareCall(appel)
```

- Appel d'une fonction

```
{? = call nomfonction (?, ?, ...)}
```

- Appel d'une procédure

```
{call nomprocédure (?, ?, ...)}
```

CallableStatement – Gestion des paramètres

- Paramètre d'entrée

  ***nomstatement.*`setXXX`(*rang*, *valeur*)**

- Paramètre de sortie

  ***nomstatement.*`registerOutParameter`(*rang*, *valeur*) *nomstatement.*`getXXX`(*rang*)**

  - Exemples : `getBigDecimal()`, `getBoolean()`, `getBytes()`, `getDate()`, `getDouble()`, `getFloat()`, `getInt()`, `getLong()`, `getString()`, `getTime()`, `getTimestamp()`

CallableStatement – Exécution

***nomstatement*.`execute()`**

Correspondances entre les types SQL et les types JAVA

CHAR, VARCHAR, LONGCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp