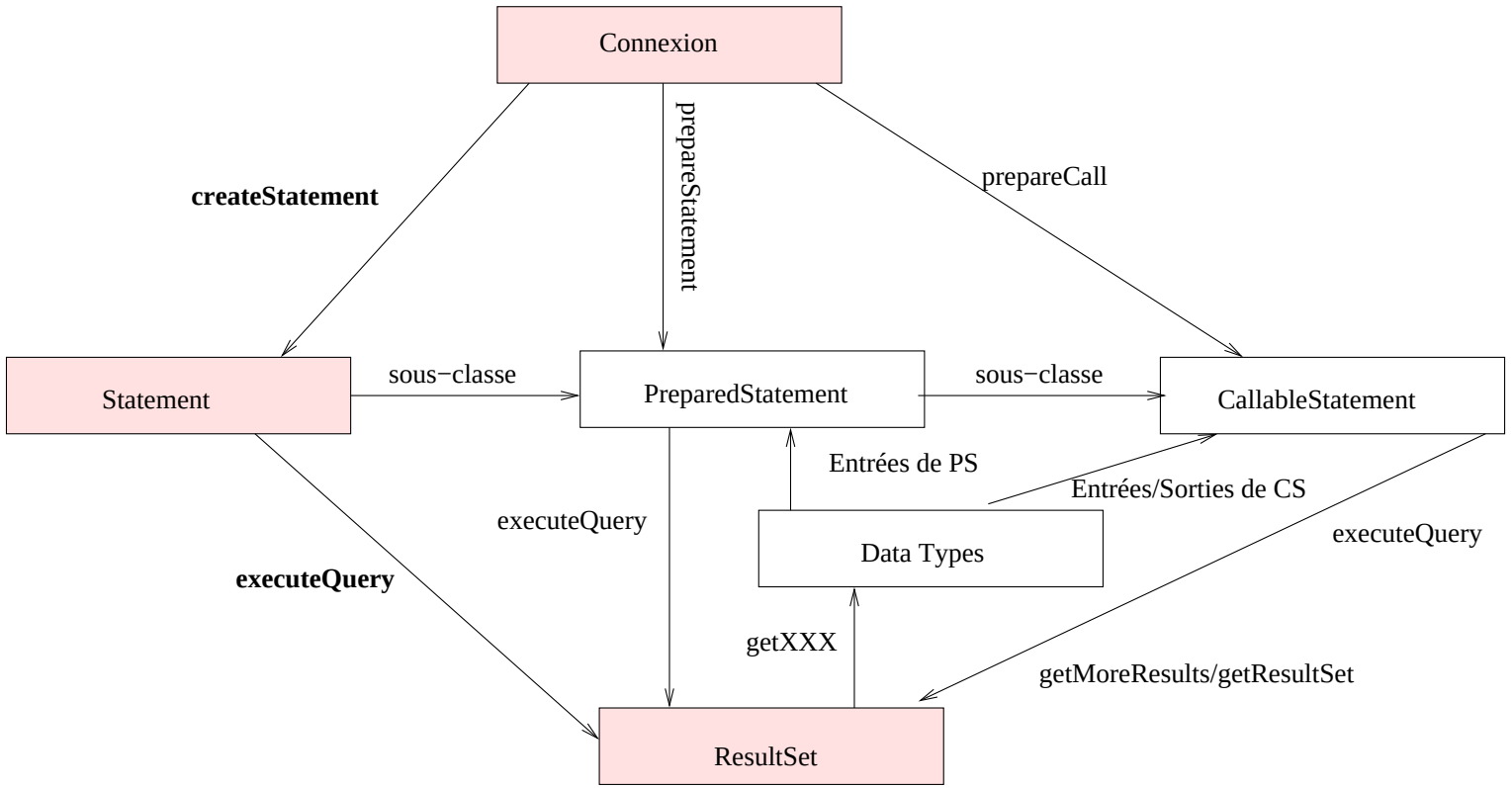
Cours Bases de données 2ème année IUT

Cours 7 : JDBC : ou comment lier ORACLE avec Java 2ème partie

Anne Vilnat
http://www.limsi.fr/Individu/anne/cours

Plan

Les classes et interfaces du package java.sql



Principales interfaces Java de l'API JDBC

Généralités

Interfaces donc méthodes définies comme public abstract.
Susceptibles de lever des exceptions SQLException.

Les interfaces

Principales interfaces :

- Connection
- Statement
- ResultSet
- ResultSetMetaData

Connexion

Les transactions

- Par défaut, autocommit on.
- Pour gérer les transactions :
  void setAutoCommit (boolean autoCommit)
  throws SQLException;
  void commit ( ) throws SQLException;
  void rollback ( ) throws SQLException;
  boolean getAutoCommit ( ) throws SQLException;
- Pour clore :
  void close ( ) throws SQLException;
  boolean isClosed ( ) throws SQLException;
- Pour transcoder les chaînes de caractères :
  String nativeSQL (String sql) throws SQLException;

Définition du Statement

Statement

- Définit les objets qui permettent d'exécuter les requêtes statiques SQL et de retourner le résultat produit.
- Donne le type et les propriétés du ResultSet qui lui sera associé
- Un seul ResultSet est actif à la fois. Si plusieurs, il faut plusieurs Statement

Créer un Statement

Les instructions

Pour créer différents Statement qui donneront des ResultSet ayant des propriétés différentes :

- Statement createStatement ( ) throws SQLException;
- Statement createStatement (int rsType, int rsConcurrency) throws SQLException;
- Statement createStatement (int rsType, int rsConcurrency, int rsHoldability) throws SQLException;

permet de définir

- le type du ResultSet (rsType), sa “navigabilité”
- le fait qu’il permette ou non des mises à jour (rsConcurrency)
- son comportement lors d’un commit (rsHoldability)

Prinsipales méthodes sur un Statement

Méthodes

Principales méthodes :

- Pour créer un ResultSet contenant les résultats d'une requête : ResultSet executeQuery (String sql) throws SQLException;
- void close ( ) throws SQLException;
- *int executeUpdate (String sql) throws SQLException*;
- *boolean execute (String sql) throws SQLException*;

Les ResultSet : quelles possibilités?

Différents types de ResultSet

- Navigabilité :
  - TYPE_FORWARD_ONLY (défaut) :
    navigation “en avant” uniquement;
  - TYPE_SCROLL_INSENSITIVE :
    dans tous les sens, et où on veut, mais pas d’accès aux modifications sur la source de données depuis l’ouverture du ResultSet;
  - TYPE_SCROLL_SENSITIVE :
    navigation + accès aux modifications;

Les ResultSet : quelles possibilités?

Différents types de ResultSet (suite)

- Mise à jour :
  - CONCUR_READ_ONLY (défaut) :
    pas de modification
  - CONCUR_UPDATABLE :
    modifications possibles
- Comportement à la validation :
  - HOLD_CURSORS_OVER_COMMIT :
    reste ouvert lors d'une validation (COMMIT ou ROLLBACK)
  - CLOSE_CURSORS_AT_COMMIT (défaut) :
    fermé après chaque validation.

Exemple de ResultSet

Pour pouvoir :

- parcourir le résultat de la requête dans n'importe quel sens, sans avoir accès à d'éventuelles modifications dans les données sur la base,
- sans pouvoir modifier les éléments au travers du ResultSet,
- et fermer le ResultSet si un commit a lieu :

Exemple

```
Statement stm =
  co.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                     ResultSet.CONCUR_READ_ONLY,
                     ResultSet.CLOSE_CURSOR_AT_COMMIT);
ResultSet rs=stm.executeQuery("SELECT * FROM Film");
```

Mouvements dans un ResultSet (1)

Les méthodes

Lors de la création : pointe “avant” la première ligne.
Si de type TYPE_FORWARD_ONLY, que :

- next() :passe à la ligne suivante. retourne true si elle existe, false sinon (après la dernière ligne).

sinon :

- previous() : ligne précédente
- first() : sur la première ligne, retourne true si elle existe, false sinon (resultSet vide)
- last() : sur la dernière;
- beforeFirst() : avant la première (comme à l'ouverture)
- afterLast () : après la dernière

Mouvements dans un ResultSet (2)

Les méthodes (suite)

- relative(int rows) : rows lignes aoprès la position courante, revient en arrière si rows est négatif;
- absolute(int row) : se place à la row-ième ligne. Si row est égal à 1 : sur la première, si row est négatif, sur la dernière. (si 0, sur la première aussi)

Exemple

```
Statement stm =
  co.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE);
ResultSet rs=stm.executeQuery("SELECT * FROM Film");
rs.absolute(5); // positionne sur la 5ème ligne;
rs.relative(10); // positionne sur la 15ème ligne
```

Modifications d'un ResultSet : mise à jour d'une ligne

Définition

Si ouvert en mode CONCUR_UPDATABLE, permet la mise à jour de la base par le biais du ResultSet (sinon on le fera par execute ou executeUpdate)

Mise à jour d'une ligne

En 2 étapes :

- mise à jour de la nouvelle valeur de la colonne : updateXXX
- changements affectés à la ligne concernée (alors seulement la base sera mise à jour) : updateRow()

Modifications d'un ResultSet : Exemple de mise à jour

Exemple

```
Statement stm =
  co.createStatement(ResultSet.TYPE_FORWARD_ONLY,);
                     ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stm.executeQuery(
              "SELECT Titre FROM Film
                 WHERE NumFilm=123");
rs.next(); // positionne sur la 1ère ligne;
rs.updateString("Titre", "Charlie et la chocolaterie");
//modifie l'attribut Titre
rs.updateRow(); // effectue la modification de la ligne
```

Modifications d'un ResultSet : suppression d'une ligne

Suppression d'une ligne

Si ouvert en mode CONCUR_UPDATABLE, permet la suppression de la ligne dans la base : deleteRow(). Se place ensuite avant la première ligne valide suivant celle qui vient d'être supprimée.
Son indice devient invalide.
→ test de l'existence d'une ligne par rowDeleted

Exemple

```
Statement stm =
   co.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,);
                      ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stm.executeQuery("SELECT * FROM Film");
rs.absolute(4); // positionne sur la 4ème ligne;
rs.deleteRow(); //supprime la ligne
bool =rs.rowDeleted(); // bool vaut vrai
```

Modifications d'un ResultSet : insertion d'une ligne

Définition

Si ouvert en mode CONCUR_UPDATABLE, permet l'insertion de nouvelles lignes

Insertion d'une ligne

En 3 étapes :

- se positionner sur la ligne d'insertion : moveToInsertRow():
- initialiser les valeurs des champs de la ligne à insérer : updateXXX,
- insérer la ligne : insertRow()

Modifications d'un ResultSet : Exemple d'insertion

Exemple

```
Statement stm =
  co.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,);
                     ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stm.executeQuery(
              "SELECT Prenom, Nom, NumIndividu FROM
Individu");
rs.moveToInsertRow(); // positionne sur une ligne d'insertion;
rs.updateString(1, "Pedro");//crée le prénom
rs.updateString(2, "Almodovar");//crée le nom
rs.updateLong(3, 278866500);//crée le num...
rs.insertRow(); // insère la ligne
rs.moveToCurrentRow(); // revient à l'ancienne position
```

Meta données sur le ResultSet : ResultSetMetadata

Interface pour obtenir des information supplémentaires sur la structure d'un ResultSet. Pour répondre à :

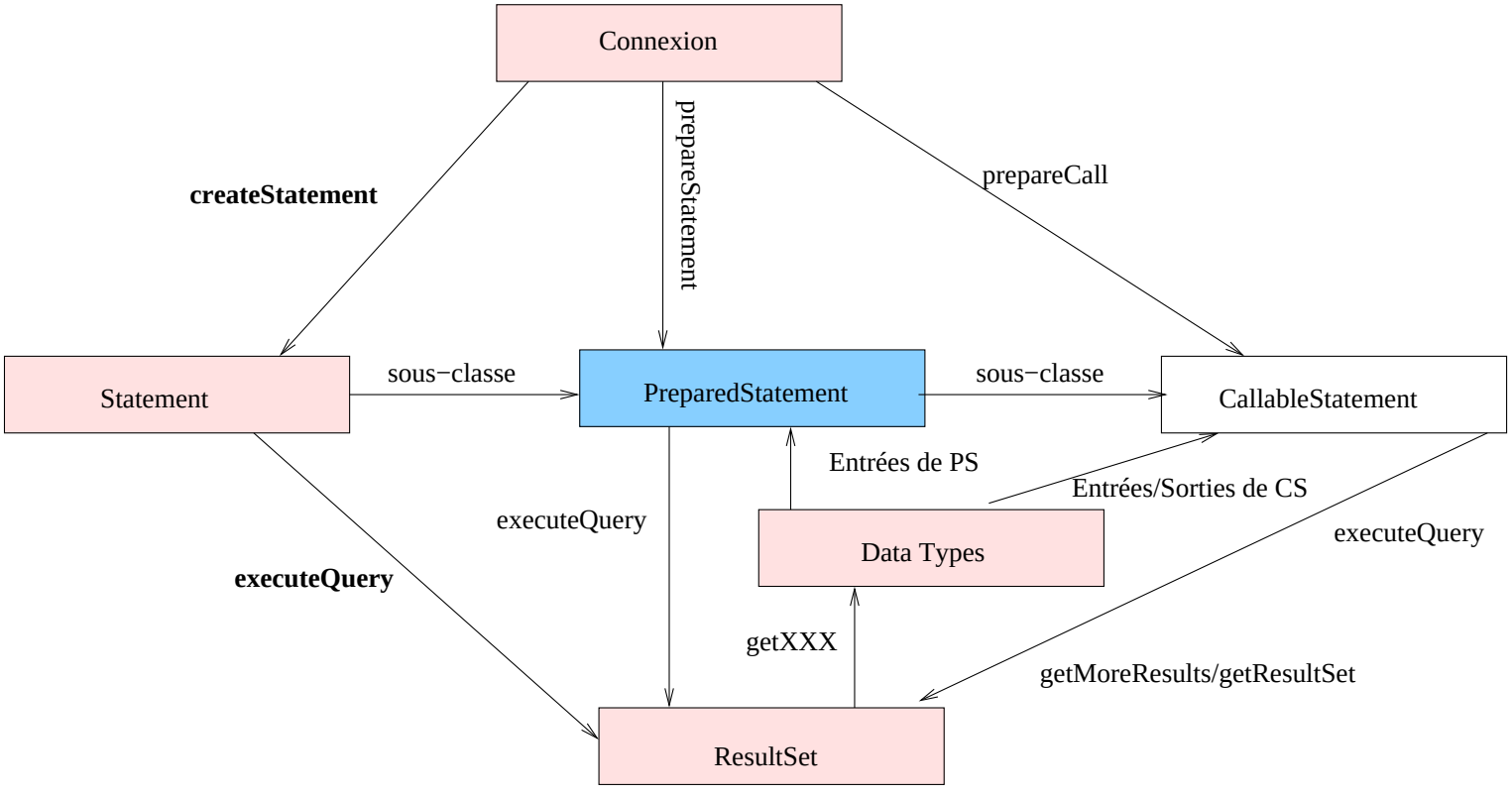
- Combien de colonnes le ResultSet contient-il ?
- Les noms de ces colonnes sont-ils sensibles à la casse ?
- Est-il possible de rechercher des données dans la colonne de son choix ?
- Est-il possible d'affecter NULL à une colonne donnée ?
- Quel est le nombre maximal de caractères affichables pour une colonne donnée ?
- Quel libellé faut-il attribuer à la colonne choisie lors de l'affichage ?
- Quel est le nom de la colonne choisie ?
- De quelle table la colonne choisie provient-elle ?
- Quel type de données la colonne choisie renferme-t-elle ?

ResultSetMetadata : Exemple

Exemple

```
Connection myconnexion = DriverManager.getConnection(url);
Statement stmt = myconnexion.createStatement ( );
ResultSet rs = stmt.executeQuery (
                                "SELECT a,b,c FROM Table1");
ResultSetMetaData rsmd = rs.getMetaData ( );
int numberOfColumns = rsmd.getColumnCount ( );
for (int i = 1; i <= numberOfColumns; i++) {
  int jdbcType = rsmd.getColumnType (i);
  String name = rsmd.getColumnTypeName (i);
  System.out.print ("L'attribut " + i +
                                "est du type JDBC " + jdbcType);
  System.out.println ("dont le nom de type JDBC est " + name);
}
```

Les requêtes pré-compilées



Requêtes pré-compilées

Motivation

Si on doit répéter plusieurs fois la même requête en utilisant un objet Statement, à chaque fois le serveur devra interpréter la requête SQL et en particulier créer un plan de requête :

- $\longrightarrow$ particulièrement coûteux.
- $\longrightarrow$ utilisation de l'interface PreparedStatement.

Le code SQL est transmis à la base une seule fois, dès que la méthode prepareStatement(), issue de la classe Connection délivre l'objet PreparedStatement correspondant.

Requêtes pré-compilées : PreparedStatement

Exemple

```
PreparedStatement psm = myconnexion.prepareStatement (
                        "SELECT nom_emp FROM employe
                              WHERE prime > 999
                              AND service = 'Comptabilite'
                              AND salaire > 2499 ");
for (int i = 0; i < 10; i++) {
   ResultSet myresultat = psm.executeQuery ( );
   while (myresultat.next ( )) {
      // traitement des donnees recupérées
. . .    }
}
```

Pas forcément très intéressant!...

Requêtes pré-compilées : PreparedStatement (2)

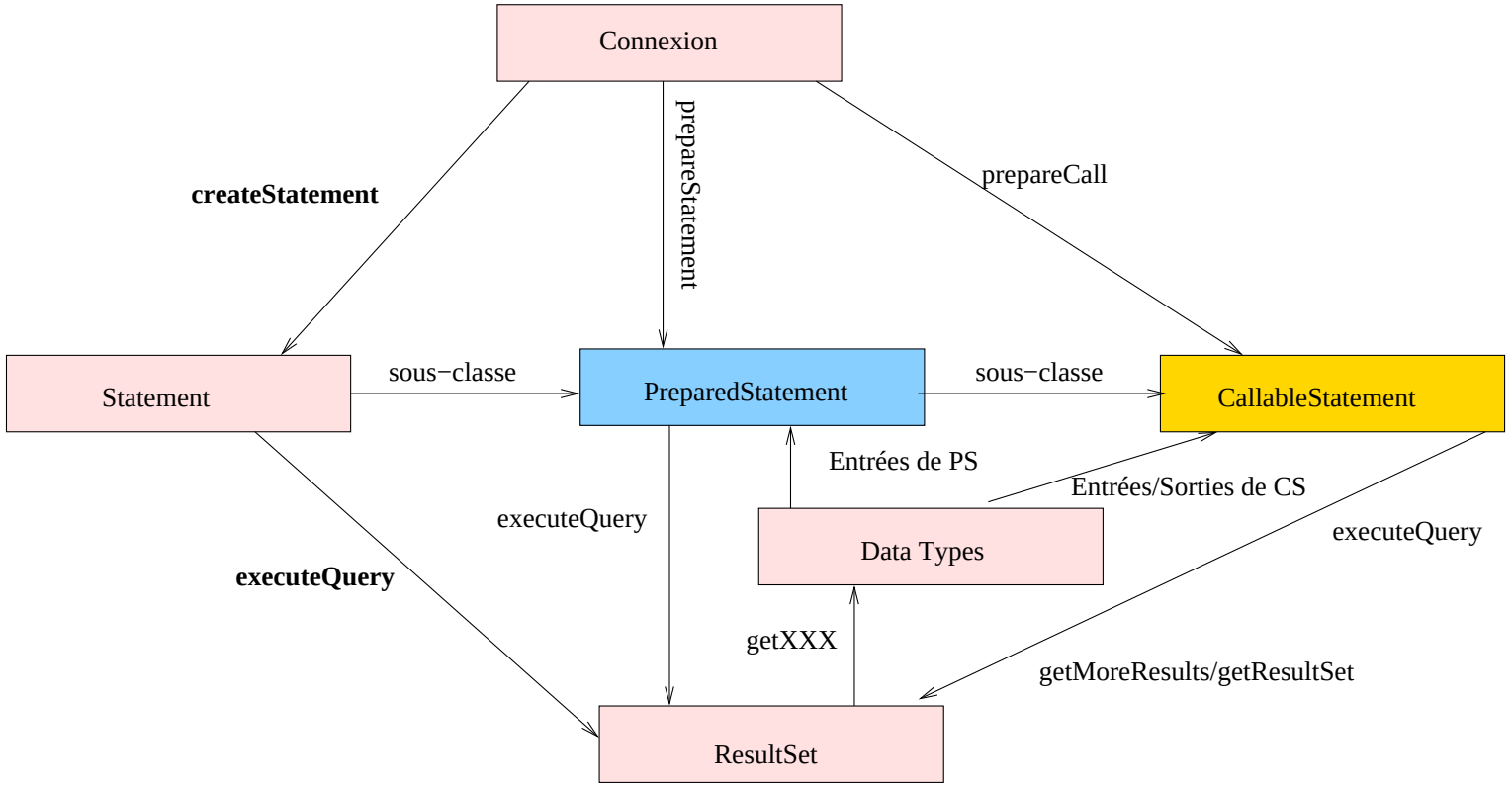
Avec des paramètres symbolisés par ? et setXXX

Exemple

```
PreparedStatement psm = myconnexion.prepareStatement (
                        "UPDATE employe SET prime = prime + ?
                             WHERE num_employe = ?
                             AND service = ?");
while (suite ( )) {
    psm.setInt (2, saisie_numero_employe ( ));
    psm.setInt (1, saisie_prime ( ));
    psm.setString (3, saisie_service ( ));
    int myresultat = psm.executeUpdate ( );
}
```

Avec : saisie_numero_employe, saisie_prime, saisie_service sont des méthodes, et suite est un booléen.

Procédures et fonctions stockées



Appel aux procédures et aux fonctions stockées

Motivation

- L'interface CallableStatement permet d'appeler des procédures ou des fonctions stockées.
- On indique le nom de la procédure ou de la fonction requise lors de l'initialisation de l'objet CallableStatement grâce à la méthode prepareCall() de l'interface Connection.
- Deux formes possibles, selon que la procédure ou la fonction stockée comporte des paramètres lors de l'appel ou n'en comporte pas

Appel aux procédures et aux fonctions stockées

Comment?

- sans paramètre :

```
CallableStatement cst = myconnexion prepareCall (
                        "call nom_procedure");
```

- avec des paramètres :

```
CallableStatement cst = myconnexion prepareCall (
                        "? = call nom_fonction (? , ? , ... )");
```

ou :

```
CallableStatement cst = myconnexion prepareCall (
                        "call nom_procedure (? , ? , ... )") ;
```

Appel aux procédures et aux fonctions stockées : paramètres

Paramètres en entrée

- Pour positionner les paramètres effectifs d'entrée (IN ou IN OUT):
  setXXX(), avant le : execute()
  où XXX est un nom de type Java.
- Paramètres de gauche à droite dans l'ordre d'apparition dans l'instruction SQL.
- setXXX() a deux paramètres : le rang du paramètre à positionner et la valeur transmise au paramètre de la fonction ou de la procédure.

Appel aux procédures et aux fonctions stockées : paramètres(suite)

Paramètres en sortie

- Paramètres de sortie (OUT ou IN OUT):
  récupérés après le execute(), par getXXX(),
  où XXX est un nom de type Java.
- getXXX() a un paramètre : le rang du paramètre à récupérer.

Typage des paramètres

Lorsqu'une fonction ou une procédure stockée renvoie une valeur (valeur de retour, paramètre OUT ou paramètre IN OUT), JDBC exige d'en spécifier le type:
registerOutParameter() avant exécution de la procédure.

CallableStatement : Exemple

Exemple d'une fonction stockée

On cherche le nombre d'Individus portant le même nom

```
CREATE FONCTION memeNomFonc (nom Individu.nomIndividu/
RETURN NUMBER IS
  nbIndividus NUMBER;
BEGIN
  SELECT COUNT(*) INTO nbIndividus
    FROM individu
        WHERE nomIndividu = nom
        GROUP BY nomIndividu
    RETURN Individus;
END;
```

CallableStatement : Exemple

Exemple d'une fonction stockée : les paramètres

- le nom en entrée : setString
- le nombre en sortie : le type (registerOutParameter) et la récupération (getInt)

Exemple d'une fonction stockée

```
CallableStatement cst = myconnexion prepareCall
                                ("? = call memeNomFonc (? )");
cst.registerOutParameter (1,java.sql.Types.NUMBER);
cst.setString (2,'FONDA');

boolean succes = cst.execute ( );

int rNb = cst.getInt (1);
cst.close ( );
```

CallableStatement : Exemple

Exemple d'une fonction stockée plus complexe

```
CallableStatement cst = myconnexion prepareCall
                                ("? = call fNomEmp (? , ? , ?)");
cst.registerOutParameter (1,java.sql.Types.VARCHAR2);
cst.registerOutParameter (2,java.sql.Types.INTEGER);
cst.registerOutParameter (3,java.sql.Types.VARCHAR2);
cst.setInt (4,247);

boolean succes = cst.execute ( );

String rNom = cst.getString (1);
int rSalaire = cst.getInt (2);
String rService = cst.getString (3);
. . .
cst.close ( );
```

La classe SQLException

Méthodes

java.sql.SQLException hérite de java.sql.SQLException.
Parmi les méthodes définies dans java.lang.Exception,:

- getSQLState() qui renvoie la chaîne de caractères correspondant à SQLSTATE (le code d'erreur de la norme SQL),
- getErrorCode() qui renvoie l'entier correspondant au "code d'erreur vendeur" (le code d'erreur propre à l'éditeur de la base de données, donc non normalisé),
- getNextException() qui donne l'exception qui suit l'exception courante dans le chaînage des exceptions,
- setNextException() qui permet d'ajouter l'exception passée en paramètre au chaînage des exceptions. Cette méthode n'est normalement utilisée que par les développeurs de drivers.

La classe SQLException

Exemple

```
catch (SQLException ex) {
    System.out.println (" Capture une SQLException :);
      while (ex ! = null) {
      System.out.print ("SQLSTATE: " + ex.getSQLState ( ));
      System.out.print (" Message: " + ex.getMessage ( ));
      System.out.println (" Code d'erreur vendeur: "
                              + ex.getErrorCode ( ));
      ex.printStackTrace (System.out)
      ex = ex.getNextException ( );
      System.out.println (" ");
    }
}
```

Les classes et interfaces du package java.sql