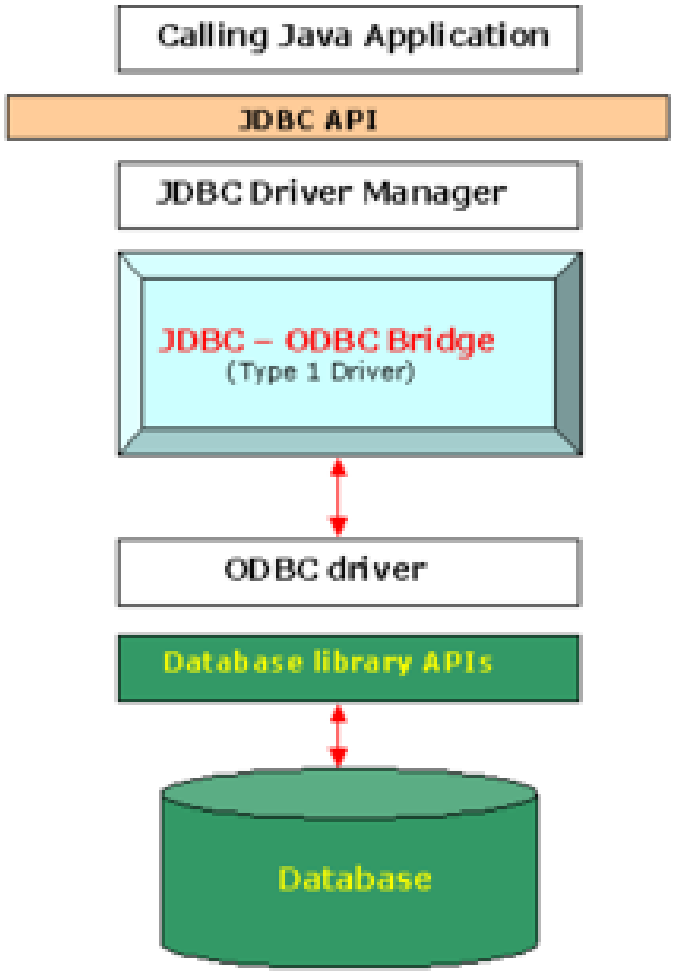
EISTI
Ecole Internationale des Sciences du Traitement de l'Information

Traitement et administration des données

JDBC: Java Database Connectivity

JDBC : Introduction

- JDBC : Java Database Connectivity
- JDBC est un ensemble de classes Java permettant d'interfacer un programme Java avec une base de données SQL
- Cette ensemble de classe permet de coder la partie Java sans connaître le type SGBD : Oracle, MySQL, SQL Server, Informix
- Pour satisfaire le point précédent, il faut récupérer dans votre projet Java, un jar propre au SGBD. Pour Oracle, vous devez récupérer ojdbc14.jar pour les actuelles versions d'Oracle
- Il y a trois autres types de middleware. Il y a des JDBC qui ne s'appuient pas sur un driver ODBC. En particulier celui qu'on utilise ne s'appuie sur ODBC.



JDBC : La connexion à la BDD

- Pour se connecter à la BDD, on utilise un objet de la classe Connection dans le package java.sql.
- Pour récupérer cet objet Connection, on fait appel à la méthode statique getConnection de la classe DriverManager
- Connection con = DriverManager.getConnection(url, login , password);
- Le paramètre *url* est une chaîne de caractère composé de 3 parties séparées par le symbole ":" . Chacune des parties contient :
    1. Le protocole qui vaut toujours JDBC
    2. Le sous protocole qui correspond au *nom du driver* ou au *mécanisme de connexion* à la base de données.
    3. Un mécanisme qui décrit la façon de localiser la BDD
- Les deux dernières parties sont liées entre elles et dépendent du driver

JDBC : La connexion à la BDD

- Pour Oracle, le sous protocole et le mécanisme de localisation sont : :
  1. oracle:thin:@adresseServeur:n° de port

     Exemple : *oracle:thin:@localhost:1521* en local avec Oracle XE
  2. Le nom de l'instance de la BDD sur le serveur (*XE* en local avec Oracle XE)
- Le code pour obtenir une connexion

```
Connection connection = null;
try{
Class.forName("nom de la classe du driver"); // Pour Oracle, c'est oracle.jdbc.driver.OracleDriver.
connection = DriverManager.getConnection(url,login,password);
}
catch(ClassNotFoundException cnfe){  // cas du driver introuvable
….
}
catch(SQLException sqle) { //Cf. Comment gérer les erreurs  SQL?
…
}
```

JDBC : Principe du requêtage

Pour requêter, il faut demander à l'objet de connexion un objet pour requêter

- Un objet Statement pour une requête SQL non paramétrée

```
Statement req = connection.createStatement();
```

- Un objet PreparedStatement pour une requête SQL paramétrée

```
PreparedStatement req = connection.preparedStatement("req. Par. SQL");
```

- Un objet CallableStatement pour l'appel d'une fonction ou d'une procédure stockée.

```
CallableStatement req = connection.prepareCall("{ call nomP(?,..,?) }");
CallableStatement req = connection. prepareCall("{ ? = call nomF(?,..,?) }");
```

JDBC : Exécution d'une requête

- Dans le cas d'un objet de la classe Statement ou de la classe PreparedStatement , les méthodes d'exécution sont :
  - executeQuery dans le cas d'un SELECT
  - executeUpdate dans les cas INSERT, UPDATE ou DELETE
- Dans le cas d'un objet de la classe CallableStatement, la méthode d'exécution est execute.

JDBC : Requête paramétrée

- Quand on veut gérer une requête paramétrée, on utilise un objet PreparedStatement pour la préparer.
- La requête SQL doit contenir autant de fois le symbole "?" Pour chaque paramètre. Exemple "SELECT * FROM Client where age = ?"
- Après on affecte les paramètres avec les méthodes set???(int noParametre). Le n° de paramètre commence à 1.

  Liste non exhaustive des méthodes set??? : setInt, setLong, setDouble, setDate, setString, setClob, …
- Enfin, on exécute la requête.

JDBC : Requête SELECT

- Une requête SELECT peut renvoyer plusieurs lignes, il faut donc utiliser un objet spécial pour gérer cet ensemble. L'objet appartient à la classe ResultSet.
- Si la requête est un objet Statement alors le code est :

```
ResultSet rs = req.executeQuery("requête SQL");
```

- Si la requête est un objet PreparedStatement alors le code est :

```
ResultSet rs = req.executeQuery();
```

- Lecture des lignes avec l'objet ResultSet
  - La méthode next() renvoie un booléen s'il y a ou pas une nouvelle ligne à lire.
  - Les méthodes get???(short noColonne) ou les méthodes get???(String nomColonne) renvoient la valeur d'une colonne de la ligne courante
  - Liste non exhaustive des méthodes get??? : getString, getBoolean, getDouble, getInt, getLong, getNString, …

JDBC : Batch

- Quand on veut grouper une série de requêtes de mises à jour de la base dans un seul lot, il est préférable d'utiliser un batch : traitement par lots.
- Le principe d'un batch :
  - On crée un objet req de la classe Statement
  - On prépare le lot : pour chaque requête on écrit req.addBatch("la requête");
  - On exécute le lot : req.executeBatch();
  - On valide le lot : req.commit();

JDBC : Gestion d'une transaction

- Pour gérer une transaction, on utilise l'objet de connexion
- A la création de l'objet de connexion, une transaction est créée.
- Les différentes actions sont :
  - Pour valider une transaction :

    ```
    connection.commit();
    ```
  - Pour annuler une transaction :

    ```
    connection.rollback();
    ```
  - Pour créer un point de validation :

    ```
    SavePoint sp = connection.setSavePoint();
    ```
  - Pour créer un point de validation nommé :

    ```
    SavePoint sp = connection.setSavePoint("…");
    ```
  - Pour annuler une partie d'une transaction :

    ```
    connection.rollback(sp);
    ```

JDBC : Procédures stockées

- Pour exécuter une procédure stockée, on opère en deux ou trois temps
  - On utilise un objet de la classe CallableStatement.
    CallableStatement cs = req.prepareCall("{ call … }");
  - Si un paramètre de la procédure est inconnu, on le mentionne avec le symbole "?". Avant d'exécuter le call, on donne les valeurs des paramètres inconnus avec les instructions cs.set???(noParamètre, valeurParamètre);.
  - On exécute la requête avec l'instruction cs.execute();.

JDBC : Fonctions stockées

- Pour exécuter une fonction stockée, on opère en deux ou trois temps
  - On utilise un objet de la classe CallableStatement. CallableStatement cs = req.prepareCall("{ ? = call … }");
  - Si un paramètre de la procédure est inconnu, on le mentionne avec le symbole ?. Avant d'exécuter le call, on donne les valeurs des paramètres inconnus avec les instructions. cs.set???(noParamètre, valeurParamètre);.
  - ATTENTION : les numéros des paramètres commencent à partir de 2 car le n° 1 est réservé au retour de la fonction.
  - Il faut déclarer le type du retour :

    cs.registerOutParameter(1, java.sql.Types.???);. La constante ??? Indique le type du retour (voir la documentation pour les diférents types)
  - Pour terminer, on exécute la requête avec l'instruction cs.execute();.

JDBC : Paramètres en entrées/sorties

- Une fonction ou une procédure PL/SQL peut contenir des paramètres en OUT ou IN OUT. Il est donc sûrement utile de récupérer leurs valeurs après l'exécution du code PL/SQL
- Comme pour le retour d'une fonction, il faut utiliser la méthode registerOutParameter comme suit :

```
cs.registerOutParameter(short noParam, java.sql.Types.???);
```

JDBC : Gestion des exceptions

- Pour traiter les exceptions qui sont levées par JDBC suite à une exception dans la base de données, on dispose de la classe SQLException.
- Cette classe contient 3 attributs accessibles par des getters :
  - message : contient une description de l'erreur
  - SQLState : code défini par les normes X/Open et SQL99
  - ErrorCode : le code d'erreur du fournisseur du pilote
- ATTENTION : Une requête JDBC peut générer plusieurs erreurs SQL . En conséquence, la classe SQLException a une méthode next() qui permet de parcourir toutes les exceptions levées par une requête JDBC. Dans le catch, votre code reçoit la première .

  Le parcours s'arrête quand cette méthode renvoie null.

Questions ?