



J.D.B.C.

Java Database Connectivity

Mohammad Ghoniem

Université de Bretagne-Sud



Plan

- Se connecter à une base de données
 - Charger un pilote de BDD
 - Ouvrir un connexion à une BDD
- Les interfaces de JDBC
 - L'interface Connection
 - L'interface Statement
 - L'interface DatabaseMetaData



Quatre types de pilotes JDBC

- Les pilotes de pont JDBC/ODBC
 - convertissent les appels JDBC en ODBC
 - livrés avec l'API JDBC
- Les pilotes de type 2
 - écrits partiellement en java
 - dépendent de code natif
- Les pilotes de type 3
 - passent par un pilote intermédiaire de type 2 ou 3.
- Les pilotes de type 4
 - écrits entièrement en java
 - code portable



Chargement du pilote JDBC

- Usage de la méthode `Class.forName("nom.classe")`

```
try {  
    Class.forName("oracle.jdbc.driver.OracleDriver") ;  
}catch(ClassNotFoundException e) {  
    System.err.println("Erreur de chargement du driver :" + e) ;  
}
```

- Variante :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

Établissement d'une connexion

□ Définition de l'url de connexion

- Une chaîne de caractères du type

```
String url = "jdbc:odbc:Fred"
```

- Le sous-protocole à utiliser dépend de l'éditeur du pilote.

□ Ouverture de connexion

- ```
Connection cxn = DriverManager.getConnection(url, "myLogin", "myPassword");
```

# Les requêtes SQL

---

## □ Création

- `Statement statement = cxn.createStatement() ;`

## □ Exécution

- `String query = "SELECT * FROM Employés";`
- `ResultSet resultset = statement.executeQuery(query);`
  
- `String query = "DELETE FROM Employés  
WHERE Région = 'WA'";`
- `int result = statement.executeUpdate(query) ;`



# Traitement des résultats

---

## □ Parcours des résultats obtenus

```
while(resultset.next()) {
 System.out.println(resultset.getString(1)) ;
}
```



# Fermeture de la connexion

---

- `con.close();`

# Les principales interfaces de JDBC

---



# Toutes les interfaces de JDBC

---

- CallableStatement
  - Gère les procédures stockées avec paramètres de sortie
- Connection
  - Gère les connexions avec la BDD
- DatabaseMetaData
  - Fournit des informations sur la BDD
- Driver
  - Interface spécifique à chaque pilote
- PreparedStatement
  - Interface d'exécution des requêtes et des procédures stockées dynamiques
- ResultSet
  - Gère les résultats d'une requête
- ResultSetMetaData
  - Fournit des informations sur les résultats
- Statement
  - Interface d'exécution des requêtes SQL et des procédures stockées normales



# Les objets de l'API JDBC

---

- Date
  - Encapsule une date au format des BDD
- DriverManager
  - Permet de créer des connexions à la BDD
- DriverPropertyInfo
  - Gère les pilotes
- Time
  - Encapsule une valeur de type Time
- Timestamp
  - Encapsule une valeur de type Timestamp
- Types
  - Fournit une liste d'entiers prédéfinis identifiant différents types de données utilisés dans JDBC



# Les exceptions de JDBC

---

- ❑ DataTrunction
- ❑ SQLException
- ❑ SQLWarning



# L'interface Connection

---

- Elle fournit des méthodes pour gérer l'exécution des requêtes SQL et les procédures stockées.
- Instanciation par le biais de la méthode `Driver.connect()`
  - Création d'une connexion persistente

# Les méthodes de l'interface Connection

---

- clearWarnings
  - Évacue les avertissements liés à cette connexion
- close
  - Ferme la connexion à la BDD
- commit
  - Lorsque le commit automatique est désactivé, permet de faire un commit.
- createStatement
  - Crée un objet pouvant servir à exécuter des requêtes SQL

# Les méthodes de l'interface Connection

---

- `getAutoCommit / setAutoCommit`
- `getCatalog / setCatalog`
- `getMetaData`
  - Retourne une instance de `DatabaseMetada`
- `getWarnings`
  - Retourne une instance de `SQLWarning` liée à cette connexion
- `isClosed`
  - Retourne un booléen indiquant si la connexion est fermée

# Les méthodes de l'interface Connection

---

- `isReadOnly / setReadOnly`
  - Retourne un booléen indiquant si la BDD est accessible en lecture seulement
- `nativeSQL`
- `prepareCall`
  - Retourne une instance de `CallableStatement` contenant une procédure stockée
  - Utile pour les procédures ayant des paramètres de E/S ou de sortie
- `prepareStatement`
  - Retourne une instance de `PreparedStatement` permettant d'exécuter des requêtes SQL dynamiques
- `Rollback`
  - Revient sur les derniers changements effectués par la BDD
  - Doit être employée en conjonction avec `setAutoCommit(false)`



# L'interface Statement

---

- ❑ Elle intervient après l'ouverture de la connexion.
- ❑ Elle fournit des méthodes pour exécuter des requêtes SQL statiques.
- ❑ Elle est instanciée par le biais de la méthode `createStatement()` d'une instance de `Connection`.



# Les méthodes de l'interface Statement

---

- ❑ `cancel`
  - Permet d'annuler une requête en cours d'exécution dans un environnement en multi-threading
- ❑ `close`
  - Permet de libérer toutes les ressources allouées à la requête
- ❑ `Execute`
  - Utilisée avec les procédures stockées lorsque le retour comprend plusieurs ResultSet
- ❑ `executeQuery`
- ❑ `executeUpdate`
  - Utilisée lorsque la requête modifie les données (delete, insert, update)



# Les méthodes de l'interface Statement

---

- `getMoreResults`, `getResultSet`, `getUpdateCount`
  - S'utilisent lorsque le retour comprend plusieurs `ResultSet`s
- `setEscapeProcessing`
  - Permet d'activer ou de désactiver les séquences d'échappement (!)
- `setMaxRows` / `getMaxRows`
  - Détermine le nombre maximal d'enregistrements qu'une requête peut retourner
- `setQueryTimeout` / `getQueryTimeout`



# L'interface DatabaseMetaData

---

- Permet d'obtenir de nombreuses informations sur la BDD et les objets qu'elle contient, comme
  - les clés primaires d'une table,
  - le support des jointures externes,
  - la liste des tables etc.
- S'instancie à l'aide de la méthode `getMetaData` de la classe `Connection`



# Les méthodes de l'interface DatabaseMetada

---

- ❑ getSchemas
- ❑ getColumns
- ❑ getExportedKeys / getImportedKeys
- ❑ getIndexInfo
- ❑ getMaxColumnsInSelect,  
getMaxColumnsInTable,  
getMaxTablesInSelect



# Les méthodes de l'interface DatabaseMetada

---

- ❑ `getPrimaryKeys`
- ❑ `getProcedures`, `getTables`
- ❑ `getURL`, `getUserName`
- ❑ `supportsGroupBy`, `supportsOuterJoins`,  
`supportsTransactions`



J.D.B.C.

# Java Database Connectivity

---

Partie II

11 avril 2005



# Plan

---

- Les interfaces de JDBC (suite)
  - L'interface PreparedStatement
  - L'interface CallableStatement
- Les résultats de requêtes
  - L'objet ResultSet
  - L'accès aux méta-données de ResultSet



# L'interface PreparedStatement

---

- ❑ Instruction préparée/précompilée
- ❑ Permet l'exécution répétée de requêtes dynamiques et des procédures stockées sans paramètres de sortie
- ❑ Veiller à fermer explicitement les instructions préparées lorsqu'on n'en a plus besoin pour libérer les ressources



# Les requêtes dynamiques

---

```
Select FirstName
From tEmployee
Where LastName = ?
```

```
Update tEmployee
Set FirstName = ?,
LastName = ?,
MiddleInitial =?
```



# Les procédures stockées

---

- Gain de performance
- Possibilité de composer plusieurs instructions SQL, procédures stockées

```
CREATE PROCEDURE getEmployees(in LastName char)
result(FirstName varchar(20)) begin
 select tEmployee.FirstName
 from tEmployee
 where tEmployee.LastName=LastName
end
```

# Les méthodes de l'interface PreparedStatement

---

- ❑ `clearParameters()`
- ❑ `execute()`
- ❑ `executeQuery()`
- ❑ `executeUpdate()`
- ❑ `setInt()`
- ❑ `setString()`

# Exemple

---

```
public DynamicSQL () {
 super();
 try {
 String driverName = "symantec.itools.db.jdbc.Driver";
 Driver driver = (Driver)Class.forName(driverName).newInstance();
 Properties p = new Properties();
 p.put("user", "dba");
 p.put("password", "sql");
 String dbURL = "jdbc:dbaw://localhost:8889/WATCOM/JDBC/JDBC";
 connection = driver.connect(dbURL, p);
 // Create the SQL statement that we will use
 String sql = "select FirstName from tEmployee where LastName = ?";
 prepare = connection.prepareCall(sql);
 }
 catch (SQLException e) {}
 catch (Exception e) {}

 // Add the text components
 lastname = new TextField(25);
 text = new TextArea(20,60);
 this.add("North", lastname);
 this.add("Center", text);

 // Resize window and display
 this.resize(300, 300);
 this.show();
}
```

# Exemple (suite)

---

```
public boolean handleEvent (Event evt) {
 // If the textfield caused the event, display items
 if (evt.target == lastname) {
 try {
 // Clear the parameters for the current statement
 prepare.clearParameters();
 // Specify the lastname parameter
 prepare.setString(1, lastname.getText());
 // Execute the statement and get the result set
 ResultSet results = prepare.executeQuery();

 String stext = "";
 // Loop through the returned items and display
 while (results.next()) {
 stext += results.getString(1) + "\n";
 }
 // Set the text displayed for the textarea component
 text.setText(stext);
 }
 catch (SQLException e) {}
 }

 return super.handleEvent(evt);
}
```



# L'interface CallableStatement

---

- Exécution de procédures stockées avec paramètres de sortie
- À comparer avec le passage de variables par référence

# Les méthodes de l'interface CallableStatement

---

- ❑ `getBoolean()`
- ❑ `getByte()`
- ❑ `getBytes()`
- ❑ `getDate()`
- ❑ `getDouble()`
- ❑ `etc.`
- ❑ `registerOutParameter()`
- ❑ `wasNull()`



# Exemple

---

```
public CallableSQL () {
 super();

 // create a connection to the database
 try {
 String driverName = "symantec.itools.db.jdbc.Driver";
 Driver driver = (Driver)Class.forName(driverName).newInstance();
 Properties p = new Properties();
 p.put("user", "dba");
 p.put("password", "sql");
 String dbURL = "jdbc:dbaw://localhost:8889/WATCOM/JDBC/JDBC";
 c = driver.connect(dbURL, p);
 }
 catch (Exception e) {}

 tarea = new TextArea(20,60);
 this.add("Center", tarea);

 this.resize(300,300);
 this.show();
}
```

# Exemple (suite)

---

```
// Create the sql that will be executed
String sql = "execute getEmployees ?";

try {
 CallableStatement call = c.prepareCall(sql);

 // declare an integer value and set it for the
 // first parameter
 int i = 0;
 call.setInt(1, i);

 // register the first parameter as an integer
 call.registerOutParameter(1, Types.INTEGER);

 // execute the stored procedure
 call.execute();

 // get the returned count
 i = call.getInt(1);

 // display the returned result
 tarea.setText("There are " + Integer.toString(i) +
 " Employees");
}
catch (SQLException e) {}
}
```



# L'objet ResultSet

---

```
Connection c = driver.connect(url, p);
String sql = "select FirstName, LastName from tEmployee";
Statement sqlStatement = connection.createStatement();
ResultSet employees = sqlStatement.executeQuery(sql);
```

# Le parcours d'un ResultSet

---

```
public ResultSetExample () {
 super();
 this.resize(300, 300);
 this.show();
 TextArea text = new TextArea(25, 80);
 this.add("Center", text);

 try {
 String driverName = "symantec.itools.db.jdbc.Driver";
 Driver driver = (Driver)Class.forName(driverName).newInstance();
 Properties p = new Properties();
 p.put("user", "dba");
 p.put("password", "sql");
 String dbURL = "jdbc:dbaw://localhost:8889/WATCOM/JDBC/JDBC";
 Connection connection = driver.connect(dbURL, p);
 String sql = "select FirstName, LastName from tEmployee";
 Statement sqlStatement = connection.createStatement();
 ResultSet results = sqlStatement.executeQuery(sql);

 String displayText = "";
 while (results.next()) {
 displayText += results.getString("FirstName") + " " + results.getString("LastName");
 }

 text.setText(displayText);
 }
 catch (SQLException e) {}
 catch (Exception e) {}
}
```



# Les méthodes de la classe ResultSet

---

- ❑ clearWarnings()
- ❑ close()
- ❑ findColumn()
- ❑ getBoolean(), getDouble() etc.
- ❑ getMetaData()
- ❑ getWarnings()
- ❑ next()
- ❑ wasNull()



# L'objet ResultSetMetaData

---

- Fournit diverses informations sur un ResultSet telles que le nom et le type des colonnes, la table à laquelle appartient une colonne et si une colonne admet des valeurs à NULL.



# Les méthodes de la classe ResultSetMetaData (extrait)

---

- ❑ `getCatalogName()`
- ❑ `getColumnCount()`
- ❑ `getColumnType()`
- ❑ `getPrecision()`
- ❑ `getTableName()`
- ❑ `isAutoIncrement()`
- ❑ `isNullable()`
- ❑ `isReadOnly()`
- ❑ `isWritable()`

---

```
public TableNameExample () {
 super();
 this.resize(300, 300);
 this.show();
 // Create the textarea to display the table name
 TextArea text = new TextArea(25, 80);
 this.add("Center", text);

 try {
 String driverName = "symantec.itools.db.jdbc.Driver";
 Driver driver = (Driver)Class.forName(driverName).newInstance();
 Properties p = new Properties();
 p.put("user", "dba");
 p.put("password", "sql");
 String dbURL = "jdbc:dbaw://localhost:8889/WATCOM/JDBC/JDBC";
 Connection connection = driver.connect(dbURL, p);
 // Select all items from tEmployee
 String sql = "select * from tEmployee";
 Statement sqlStatement = connection.createStatement();
 ResultSet records = sqlStatement.executeQuery(sql);
 ResultSetMetaData metaData = records.getMetaData();
 // Put the table name of the first column
 // into the text area object
 text.setText(metaData.getTableName(1));
 }
 catch (SQLException e) {}
 catch (Exception e) {}
}
```



# Ressources complémentaires

---

- <http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html>
- <http://www.infini-fr.com/Sciences/Informatique/Langages/Imperatifs/Java/Jdbc/>
- <http://www.commentcamarche.net/jdbc/jdbcintro.php3>
- <http://java.developpez.com/IntroJDBC.pdf>
- <http://www.unixodbc.org/>