ELEMENTE AVANSATE DE PROGRAMARE

Conf.univ.dr. Ana Cristina DĂSCĂLESCU

Java Database Connectivity

- **JDBC (Java Database Connectivity)** reprezintă API-ul dezvoltat de către Sun Microsystems pentru a oferi aplicațiilor Java acces la baze de date.

- O facilitate importantă oferită de către JDBC este posibilitatea de a utiliza instrucțiuni **SQL** (Structured Query Language), precum și de a procesa rezultatele obținute în urma unei interogări.

- Independența față de softul pentru baze de date folosit se realizează prin intermediul unor drivere specifice sistemelor de gestiune a bazelor de date (SGBD sau DBMS).

- Începând cu versiunea JBDC 2.0, API-ul este împărțit în două părți:
  1. JDBC 2.1 – nucleul API-ului, cuprins în pachetul java.sql
  2. JDBC 2.0 Optional Package - oferă servicii pe partea de server, cuprins în pachetul javax.sql

Java Database Connectivity

- O aplicație JDBC presupune efectuarea următoarelor etape:

- Înregistrarea driverului corespunzător, pentru ca JDBC să poată interacționa cu SGBD-ul particular folosit de aplicație

- Deschiderea conexiunii cu baza de date, ce presupune, în general, furnizarea unei combinații user/parola

- Formularea unor interogări de diferite tipuri

- Prelucrarea rezultatelor obținute, care diferă în funcție de tipul interogării

- Închiderea conexiunii cu baza de date

Înregistrarea unui driver

- Înregistrarea unui driver presupune încărcarea în memorie a clasei care implementează driver-ul.
- Modalități utilizate pentru încărcarea unui driver:
  - Prin folosirea clasei DriverManager

```
DriverManager.registerDriver(new TipDriver());
```

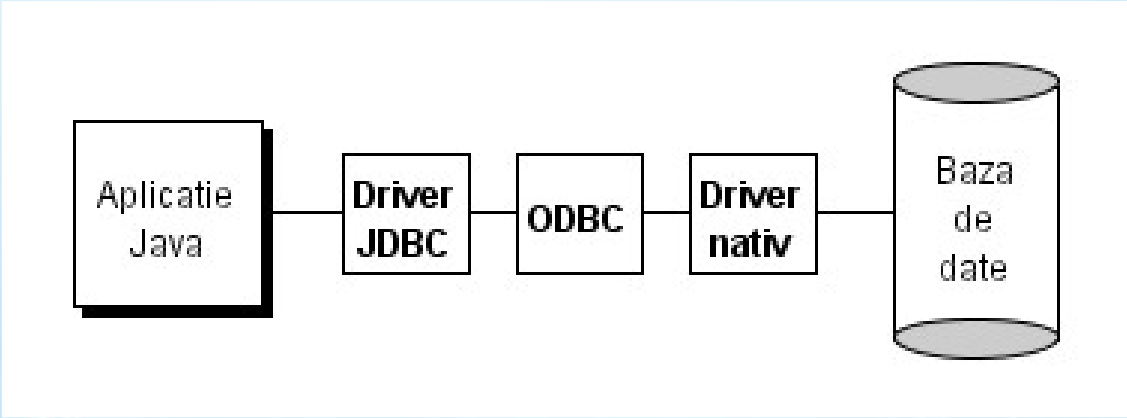
  - Prin folosirea metodei Class.forName() care permite mașinii virtuale să aloce dinamic, să încarce și să realizeze o legătură la o clasă specificată ca argument print-un șir de caractere.

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

Clasificarea driverelor JDBC

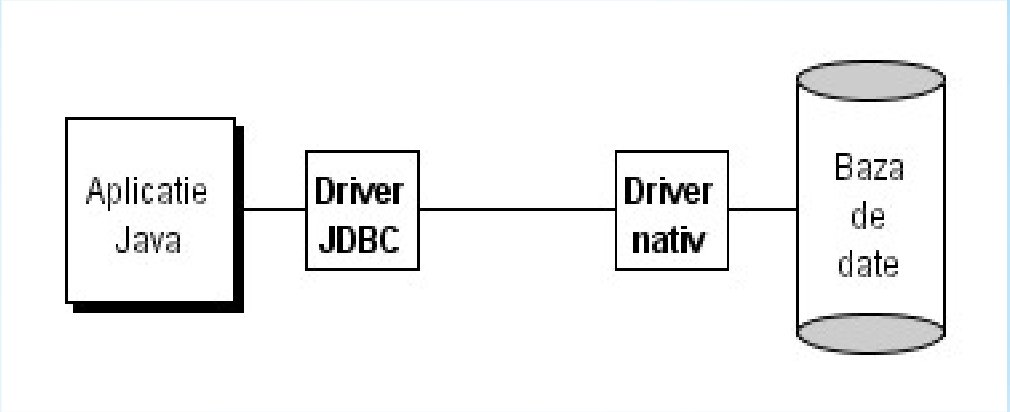
➢ **Tipuri de drivere:**

- **JDBC-ODBC Bridge**
  - Acest tip de driver permite conectarea la o bază de date care a fost înregistrată în prealabil în **ODBC** (Open Database Conectivity).
  - **ODBC** reprezintă o modalitate de a uniformiza accesul la baze de date, asociind acestora un identificator **DSN** (Data Source Name).
  - Conectarea efectivă la baza de date se va face prin intermediul DSN-ului, driver-ul ODBC efectuând comunicarea cu driverul nativ al bazei de date.

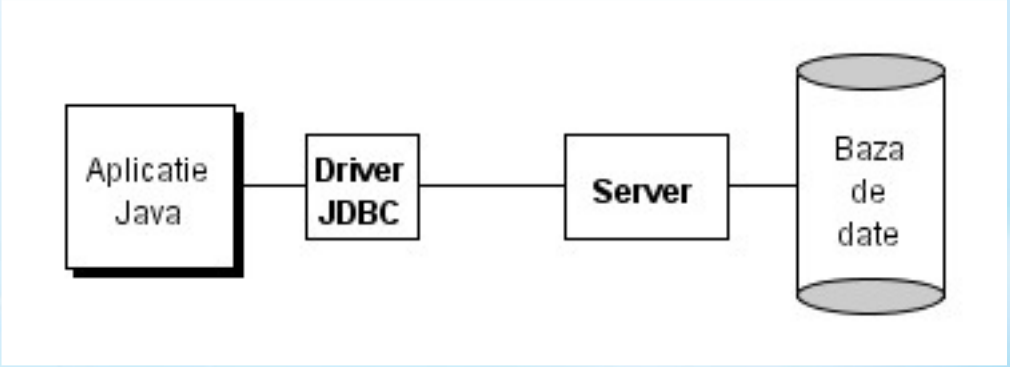


Clasificarea driverelor JDBC

- **Driver JDBC - Driver nativ**
  - Acest tip de driver transformă cererile JDBC direct în apeluri către driverul nativ al bazei de date, care trebuie instalat în prealabil.



- **Driver JDBC – Server**
  - Acest tip de driver transformă cererile JDBC folosind un protocol de rețea independent.



Deschiderea unei conexiuni cu baza de date

- **O conexiune (sesiune) la o bază de date** reprezintă un context prin care sunt trimise secvențe SQL și primite rezultate.

- Într-o aplicație pot exista simultan mai multe conexiuni la baze de date diferite sau la aceeași bază de date.

- Deschiderea unei conexiuni se realizează folosind metoda:

```
DriverManager.getConnection(String url)
```

- Se specifică, pe lângă un identificator al bazei de date, și driverul ce trebuie folosit prin intermediul unei adrese specifice, numită JDBC URL, ce are următorul format:

```
jdbc:sub-protocol:identificator
```

Deschiderea unei conexiuni cu baza de date

- Câmpul sub-protocol specifică tipul de driver ce trebuie folosit pentru realizarea conexiunii: `odbc, oracle, sybase, db2` etc.
- Identificatorul bazei de date este un indicator specific fiecărui driver corespunzător bazei de date.
- În funcție de tipul driver-ului, acest identificator poate include numele unei mașini gazdă, un număr de port, numele unui fișier sau al unui director etc.

Exemple:

```
//pentru MySql
Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/numebazadate");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost/bd","user","pass");
//pentru Apache Derby
Connection con = DriverManager.getConnection("jdbc:derby:numebazadedate");
```

Executarea secvențelor SQL

- Cele mai uzuale comenzi SQL sunt cele folosite pentru:
  - Interogarea bazei de date: **SELECT**
  - Actualizarea datelor: **INSERT, UPDATE, DELETE**
  - Actualizarea structurii: **CREATE, ALTER, DROP** - acestea mai sunt numite și instrucțiuni **DDL** (Data Definition Language)
  - Apelarea unei proceduri stocate: **CALL**

Interfața Statement

- Interfața **Statement** oferă metodele de bază pentru trimiterea de secvențe SQL către baza de date și obținerea rezultatelor.

- JDBC suporta trei tipuri de statement-uri, sub forma a 3 interfețe:

  - **Statement** – este folosită pentru transmiterea de instrucțiuni SQL simple, fără parametri.

  - **PreparedStatement** – permite folosirea instrucțiunilor SQL precompilate (interogări parametrizate)

  - **CallableStatement** – permite folosirea procedurilor stocate pe serverul SGBD.

Interfața Statement

➢Crearea unui obiect **Statement** se realizează prin intermediul metodei createStatement apelată pentru un obiect Connection.

```
Connection con = DriverManager.getConnection(url);
Statement stmt = con.createStatement();
```

➢Executarea unei secvențe SQL poate fi realizată prin intermediul metodelor:

1. **Metoda executeQuery**
   - Este folosită pentru realizarea interogărilor de tip **SELECT**
   - Metoda returnează un obiect de tip ResultSet ce va conține sub o formă tabelară rezultatul interogării.

```
String sql = "SELECT * FROM persoane";
ResultSet rs = stmt.executeQuery(sql);
```

Interfața Statement

- Forma generală a unui **ResultSet** este una tabelară.
- Un obiect de tip **ResultSet** conține și meta-datele interogării, cum ar fi: denumirile coloanelor selectate, numărul lor etc.
- Pentru a extrage informațiile din structura tabelară se va parcurge tabelul linie cu linie și din fiecare se vor extrage valorile de pe coloane prin metode de tip **getTD**.
- Sufixul **TD** reprezintă tipul de dată al unei coloane, iar argumentul primit indică fie numărul de ordine din cadrul tabelului, fie numele acestuia.

Observație: Coloanele sunt numerotate de la stânga la dreapta, începând cu 1!!!

Interfața Statement

- Un obiect **ResultSet** folosește un **cursor** pentru a parcurge articolele rezultate în urma unei interogări.

- Inițial, acest cursor este poziționat înaintea primei linii.

- **Metode uzuale:**
  - first() – poziționarea cursorului pe prima înregistrare
  - last() – poziționarea cursorului pe ultima înregistrare
  - next() – poziționarea pe următoarea înregistrare
  - previous() – poziționarea pe precedenta interogare

- Apelul metodei **next** determină trecerea la următoarea linie.

Interfața Statement

Exemplu:

```
String sql = "SELECT cod,nume FROM persoane";
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
     int cod = r.getInt("cod");
     String nume = r.getString("nume");
       // echivalent:
     int cod = r.getInt(1);
     String nume = r.getString(2);
     System.out.println(cod + ", " + nume);
}
```

Interfața Statement

2. Metoda int executeUpdate(String SQL)

- Este folosită pentru interogări SQL de tipul UPDATE, INSERT, DELETE sau pentru cele care nu produc rezultate (ex: instrucțiunile DDL, cele de manipulare a structurii etc.)

- Metoda returnează un întreg reprezentând numărul de linii modificate în urma efectuării operației respective.

Interfața PreparedStatement

- Interfața PreparedStatement este derivată din Statement
- Instanțele de tip PreparedStatement conțin secvențe SQL care au fost deja compilate.
- O secvență SQL transmisă unui obiect PreparedStatement poate să aibă unul sau mai mulți parametri de intrare, care vor fi specificați prin intermediul unui semn de întrebare (“?”).
- Crearea unui obiect de tip PreparedStatement se realizează prin intermediul metodei **prepareStatement** a clasei **Connection.**

```
Connection con = DriverManager.getConnection(url);
String sql = "UPDATE persoane SET nume=? WHERE cod=?";
Statement pstmt = con.prepareStatement(sql);
```

Interfața PreparedStatement

- Înainte ca secvența SQL să poată fi executată, trebuie ca fiecărui parametru de intrare să i se atribuie o valoare.
- Transmiterea parametrilor se realizează prin metode de tip **setTD**, unde **TD** este tipul corespunzător parametrului, iar argumentele metodei sunt numărul de ordine al parametrului de intrare (al semnului de întrebare) și valoarea pe care dorim să o atribuim.

```
pstmt.setString(1, "Ionescu");
pstmt.setInt(2, 100);
```

- Executarea unei secvențe SQL folosind un obiect **PreparedStatement** se realizează printr-una din metodele **executeQuery**, **executeUpdate** sau **execute**.

```
pstmt.executeUpdate();
```

Lucrul cu metadate

- Meta-informația definește caracteristicile structurilor informaționale care conțin informația însăși.
- În Java programatorul poate obține două tipuri de meta-informații:
  - **Meta-informația atașată unei surse de date** (ex: un server de baze de date):
    - aceasta presupune determinarea listei de baze de date, a tabelelor cuprinse în acestea, lista de coloane din fiecare tabelă etc.
    - informația de acest fel se determină sub forma unui obiect de tip DatabaseMetaData, returnat de metoda getMetaData() a clasei Connection
  - **Meta-informația atașată unui obiect de tip ResultSet**
    - se poate determina numărul de coloane, numele acestora, tipurile lor de date etc.
    - informația de acest fel este memorată în obiecte de tip ResultSetMetaData, returnate de metoda getMetaData() a clasei ResultSet

Lucrul cu metadate

➢ **DatabaseMetaData** și **ResultSetDataMeta**

- **ResultSet getCatalogs()** - returnează lista de baze de date prezente pe server sub forma unui ResultSet cu o singură coloană, denumita **TABLECAT**
- **ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)** - întoarce lista de tabele dintr-o bază de date
- **ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)** - extrage lista de coloane a unei tabele
- **String getColumnName(int)** - returnează numele coloanei în funcție de poziția acesteia în ResultSet
- **int getColumnType(int)** - returnează tipul de date al coloanei, valoarea întreagă fiind una dintre constantele din *clasa java.sql.Types:* INTEGER, DOUBLE, DATE etc.

Lucrul cu metadate

```
ResultSet rs = stmt.executeQuery("SELECT * FROM
 tabel");
ResultSetMetaData rsmd = rs.getMetaData();
// Aflăm numarul de coloane
int n = rsmd.getColumnCount();
// Aflăm numele coloanelor
Sring nume[] = new String[n+1];
for(int i=1; i<=n; i++)
    nume[i] = rsmd.getColumnName(i);
```