Programare avansată

JDBC

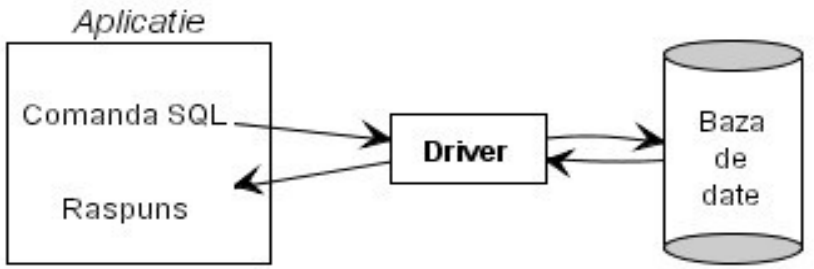
Baze de date

- **BD** - Modalitate de stocare a unor informații (date) pe un suport extern, cu posibilitatea regăsirii acestora.
- **SGBD** (DBMS) – Sistem de gestiune a bazelor de date: creare BD, interogare, administrare, etc.
- *Eficiență* (indecși, etc.)
- *Consistență* (FK, PK, triggers, etc.)
- *Securitate* (utilizatori, permisiuni, etc.)

- Modele: **relațional**, orientat-obiect, graf, XML, NoSQL, NewSQL, etc.
- Producători: Oracle, Microsoft, Sybase, etc.

Aplicații care folosesc BD

- Crearea bazei de date: script SQL
- Conectarea la o bază de date: driver
- Comunicarea cu baza de date
  - Executarea unor comenzi SQL
    - DDL, DML, DCL
  - Procesarea rezultatelor

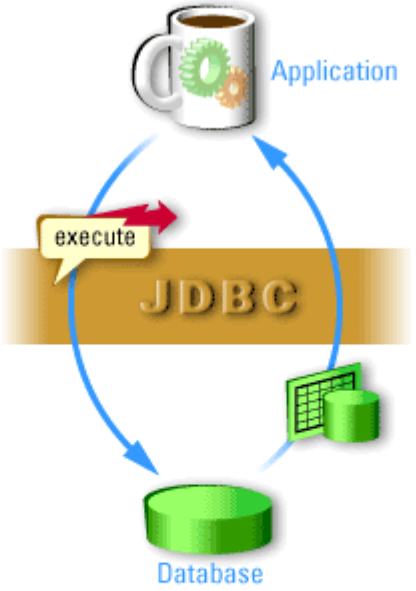


JDBC

- JDBC (Java Database Connectivity) este tehnologia Java de acces la baze de date relaționale
- Este independentă de tipul bazei de date
- Orientată pe utilizarea de adaptori (drivere) între client și SGBD
- **java.sql** – nucleul tehnologiei JDBC
- javax.sql – extensie dedicată platformei JavaEE

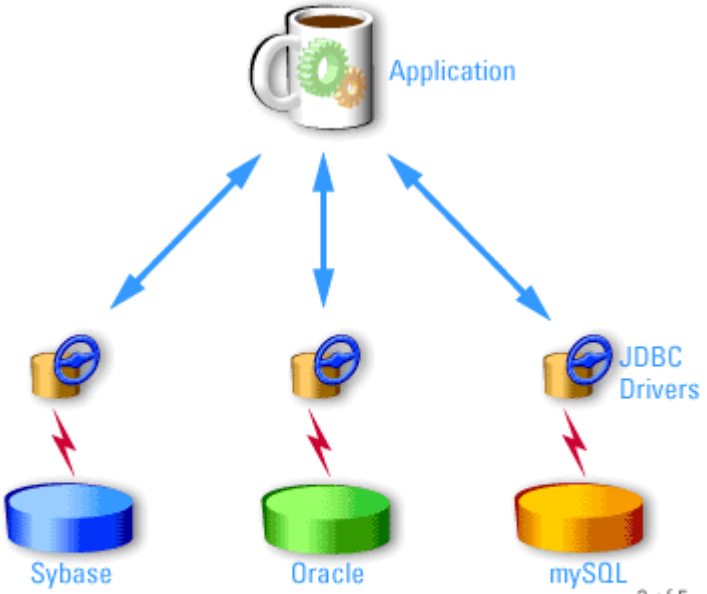
Driver

Interfața dintre aplicație și baza de date



1 of 5

JDBC allows an application to send SQL statements to a database and receive the results.



2 of 5

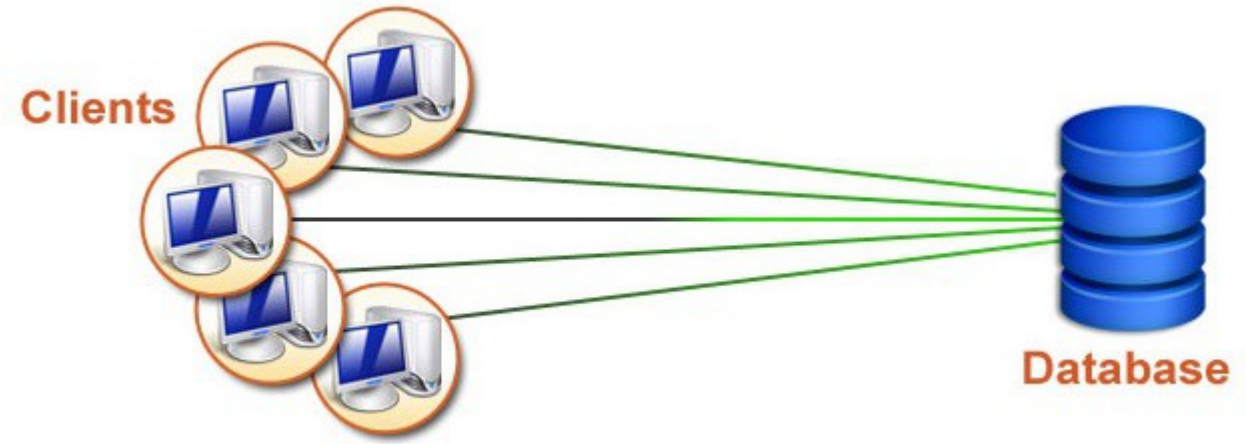
JDBC interfaces for specific database engines are implemented by a set of classes called JDBC drivers. Since the JDBC driver handles the low-level connection and translation issues, you can focus on the database application development without worrying about the specifics of each database.

Utilizarea unui driver

- Identificarea clasei driver specifică bazei de date:
  - de exemplu: *mysql-connector-java.jar*
  - *adăugarea arhivei jar în CLASSPATH*
  - identificarea clasei: *com.mysql.jdbc.Driver*
- Incărcarea în memorie a clasei driver
  - `DriverManager.registerDriver(new com.mysql.jdbc.Driver());`
  - `Class.forName("com.mysql.jdbc.Driver").newInstance();`
  - `System.setProperty(`
    `"jdbc.drivers", "com.mysql.jdbc.Driver");`
  - `java -Djdbc.drivers=com.mysql.jdbc.Driver Aplicatie`

Conexiune

- **Conexiune (sesiune)** = context prin care se realizează comunicarea cu o bază de date:
  - trimiterea de comenzi
  - primirea rezultatelor
- Intr-o aplicație pot exista simultan mai multe conexiuni la baze de date diferite sau la aceeași bază.



Identificarea unei baze de date

JDBC URL

jdbc:sub-protocol:identificator

Sub-protocolul identifică tipul de driver, de exemplu:

odbc, mysql, oracle, sybase, postgres, etc.

Identificatorul bazei de date:

jdbc:odbc:test

jdbc:mysql://localhost/test

jdbc:oracle:thin@persistentjava.com:1521:test

jdbc:sybase:test

Conectarea la o bază de date

O conexiune este reprezentată printrun obiect de tip *java.sql.Connection*

```
Connection conn = DriverManager.getConnection(url);
Connection conn = DriverManager.getConnection(
                  url, username, password);
Connection conn = DriverManager.getConnection(
                  url, dbproperties);
```

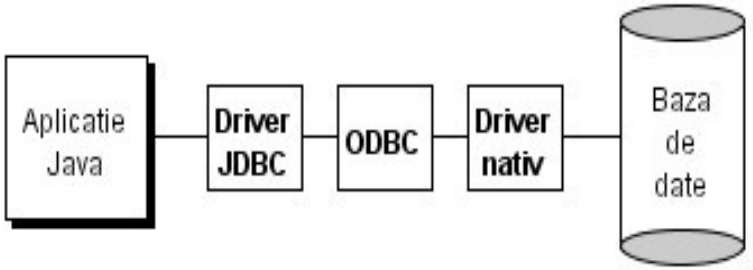
Inchiderea unei conexiuni: `conn.close()`

Exemplu

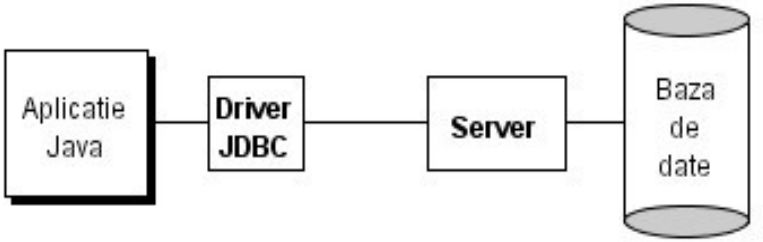
```
String url = "jdbc:mysql://localhost/test" ;

Connection con = null;
try {
  Class.forName("com.mysql.jdbc.Driver");

  Connection con = DriverManager.getConnection(
    url, "myUserName", "mySecretPassword");

} catch(ClassNotFoundException e) {
  System.err.print("ClassNotFoundException: " + e) ;

} catch(SQLException e) {
  System.err.println("SQLException: " + e);

} finally {
  con.close ;
}
```

Tipuri de drivere

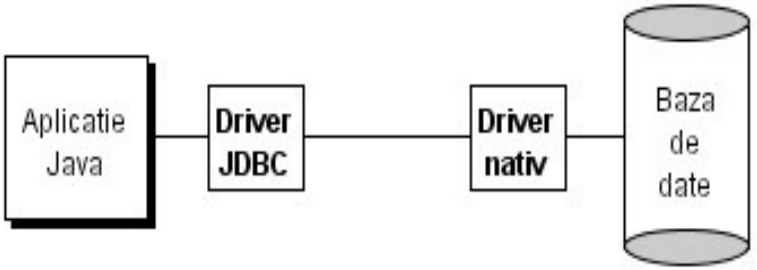
Tip 1



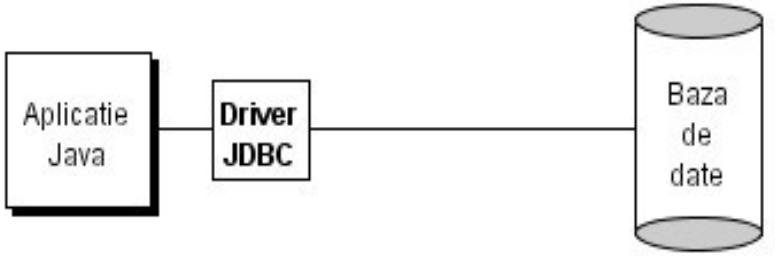
Tip 3



Tip 2



Tip 4



JDBC-ODBC Bridge

- **ODBC**: Open Database Conectivity
- Driver: *sun.jdbc.odbc.JdbcOdbcDriver*
- URL: *jdbc:odbc:identificator*
  - identificator DSN (Data Source Name)
- soluție “universală” de conectare la o BD, simplu de utilizat
- nu este portabilă, viteză de execuție slabă

“The JDBC-ODBC Bridge should be considered a transitional solution. It is not supported by Oracle. Consider using this only if your DBMS does not offer a Java-only JDBC driver.”

Utilizarea conexiunilor

- Crearea de secvențe SQL utilizate pentru interogarea sau actualizarea bazei
  - *Statement, PreparedStatement,*
  - *CallableStatement*
- Aflarea meta-datelor: informații legate de baza de date sau de rezultatele unor interogări
  - *DatabaseMetaData, ResultSetMetaData*
- Controlul tranzacțiilor
  - *commit, rollback*
  - *setAutoCommit*

Secvențe SQL: *Statement*

- Crearea unui *Statement*

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

- Execuția unei interogări

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

- Execuția unei actualizări a bazei de date

```
String sql = "DELETE FROM persoane WHERE cod > 100";
int linii = stmt.executeUpdate(sql);
// Nr de articole care au fost afectate (sterse)
sql = "DROP TABLE temp";
stmt.executeUpdate(sql); // Returneaza 0
```

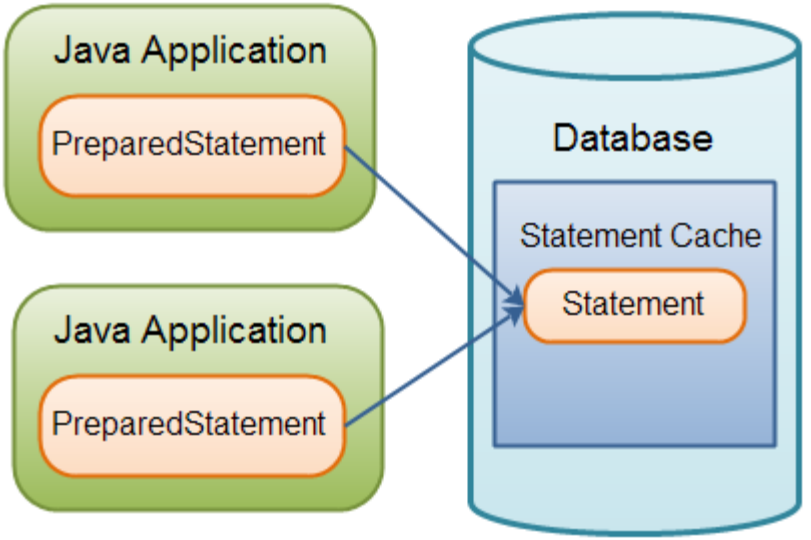
- Execuție unei comenzi generice

```
stmt.execute("secventa SQL oarecare");
```

PreparedStatement

Instanțele de tip *PreparedStatement* conțin secvențe SQL care au fost deja compilate, scopul fiind creșterea vitezei de execuție, în cazul comenzilor *batch*.

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

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

pstmt.setString(1, "Popescu");
pstmt.setInt(2, 200);
pstmt.executeUpdate();
```



Tipuri de date JDBC

Sunt definite în clasa *java.sql.Types*

tipuri de date Java – tipuri de date SQL

Metoda setObject permite specificarea explicită a tipului

```
pstmt.setObject(1, "Ionescu", Types.CHAR);
pstmt.setObject(2, 100, Types.INTEGER); // sau doar
pstmt.setObject(2, 100);
```

Metoda setNull

```
pstmt.setNull(1, Types.CHAR);
pstmt.setInt(2, null);
```

Date de dimensiuni mari

setBinaryStream, setAsciiStream, setUnicodeStream

```
//Exemplu
File file = new File("date.txt");
InputStream fin = new FileInputStream(file);
java.sql.PreparedStatement pstmt =
  con.prepareStatement(
    "UPDATE fisiere SET continut = ? " +
    "WHERE nume = 'date.txt'");
pstmt.setUnicodeStream (1, fin, fileLength);
pstmt.executeUpdate();
```

CallableStatement

Apelarea procedurilor stocate

```
//Crearea
Connection con = DriverManager.getConnection(url);
CallableStatement cstmt = con.prepareCall(
  "{call proceduraStocata(?, ?)}");

//Trimiterea parametrilor și execuția
cstmt.setString(1, "Ionescu");
cstmt.setInt(2, 100);
cstmt.executeQuery();

CallableStatement cstmt = con.prepareCall(
  "{call calculMedie(?)}");
cstmt.registerOutParameter(1, java.sql.Types.FLOAT);
cstmt.executeQuery();
float medie = cstmt.getDouble(1);
```

Obținerea rezultatelor

Interfața **ResultSet**

```
Statement stmt = con.createStatement();
String sql = "SELECT cod, nume FROM persoane";

ResultSet rs = stmt.executeQuery(sql);
```

cod	nume
100	Ionescu
200	Popescu

```
while (rs.next()) {
  int cod = rs.getInt("cod"); //rs.getInt(1)
  String nume = rs.getString("nume");
  System.out.println(cod + ", " + nume);
}
```

Cursoare modificabile

```
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT cod, nume FROM persoane";
ResultSet rs = stmt.executeQuery(sql);
  // rs will be scrollable,
  // will not show changes made by others
  // and will be updatable
```

Metode suplimentare

- absolute
- updateRow
- moveToInsertRow
- insertRow
- moveToCurrentRow
- deleteRow

supportsPositionedUpdate/Delete

Interfața *RowSet*

- Extinde *ResultSet*
- Conformă cu specificațiile JavaBeans
  - Proprietăți
  - Mecanism de notificare (bazat pe evenimente)
- *JdbcRowSet*
- *CachedRowSet* (disconnected)
- *WebRowSet* (XML)
- *JoinRowSet* (offline join)
- *FilteredRowSet* (offline filtering)

Exemple

```
JoinRowSet jrs = new JoinRowSetImpl();

ResultSet rs1 = stmt.executeQuery("SELECT * FROM EMPLOYEES");
CachedRowSet empl = new CachedRowSetImpl();
empl.populate(rs1);
empl.setMatchColumn(1);
jrs.addRowSet(empl);

ResultSet rs2 = stmt.executeQuery("SELECT * FROM BONUS_PLAN");
CachedRowSet bonus = new CachedRowSetImpl();
bonus.populate(rs2);
bonus.setMatchColumn(1); // EMP_ID is the first column
jrs.addRowSet(bonus);

FilteredRowSet frs = new FilteredRowSetImpl();
frs.populate(rs1);
Range name = new Range("Ionescu", "Popescu", "EMP_NAME");
frs.setFilter(name); //accepta obiecte Predicate
frs.next();
```

Interfața *DatabaseMetaData*

Informații sistem ale BD (metadate - ”date despre date”): *tabele, proceduri stocate, capabilitățile conexiunii, gramatica SQL suportată, etc.*

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

DatabaseMetaData dbmd = con.getMetaData();

// Obținem tabelele din baza de date
ResultSet rs = dbmd.getTables (null, null, null, null);
// catalog, schemaPattern, tableNamePattern, types)

while (rs.next ())
  System.out.println(rs.getString ("TABLE_NAME"));
  con . close ();
}
```

Interfața *ResultSetMetaData*

Informații despre rezultatul conținut într-un *ResultSet,* cum ar fi: numărul coloanelor, tipul și denumirile lor, etc.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM tabel");
ResultSetMetaData rsmd = rs.getMetaData();
// Aflam numarul de coloane
int n = rsmd.getColumnCount();

// Aflam numele coloanelor
Sring nume[] = new String[n];
for(int i=0; i<n; i++) {
  nume[i] = rsmd.getColumnName(i);
}
```

Controlul tranzacțiilor

- Tranzacție = unitate de lucru ACID
- ACID = Atomic, Consistent, Isolated, Durable
- COMMIT, ROLLBACK

```
con.commit();
con.rollback();
```

- Savepoints

```
Savepoint save1 = con.setSavepoint();
…
con.rollback(save1);
```

- Dezactivarea modului *AutoCommit*

```
con.setAutoCommit(false);
```

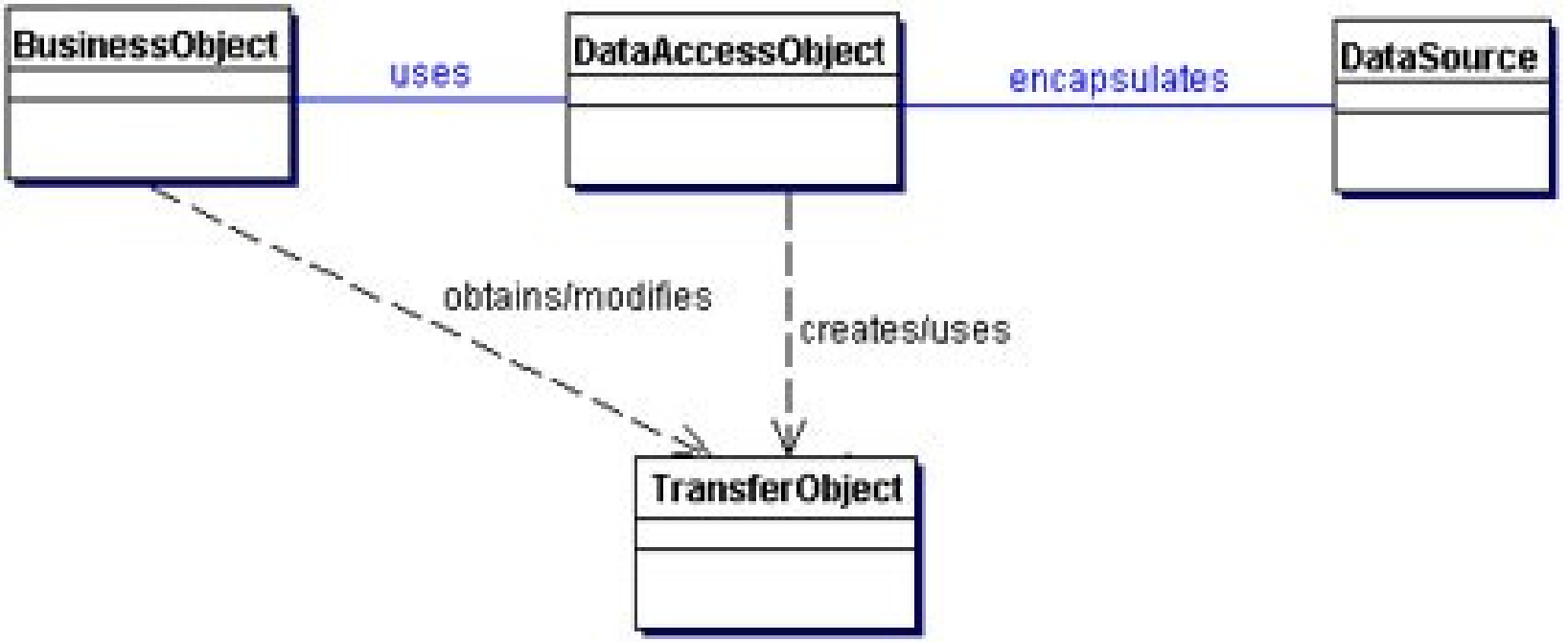
Tratarea excepțiilor

- *SQLException*

```
public static void printSQLException(SQLException ex) {
  for (Throwable e : ex) {  //SQLException implements Iterable<Throwable>
    //chained exceptions
    if (e instanceof SQLException) {
      SQLException sqlEx = (SQLException)e);
      System.err.println("SQLState  : " + sqlEx.getSQLState());
      System.err.println("Error Code: " + sqlEx.getErrorCode());
      System.err.println("Message   : " + sqlEx.getMessage());
      Throwable t = ex.getCause();
      while(t != null) {
        System.out.println("Cause: " + t);
        t = t.getCause();
      }
    }
  }
}
```

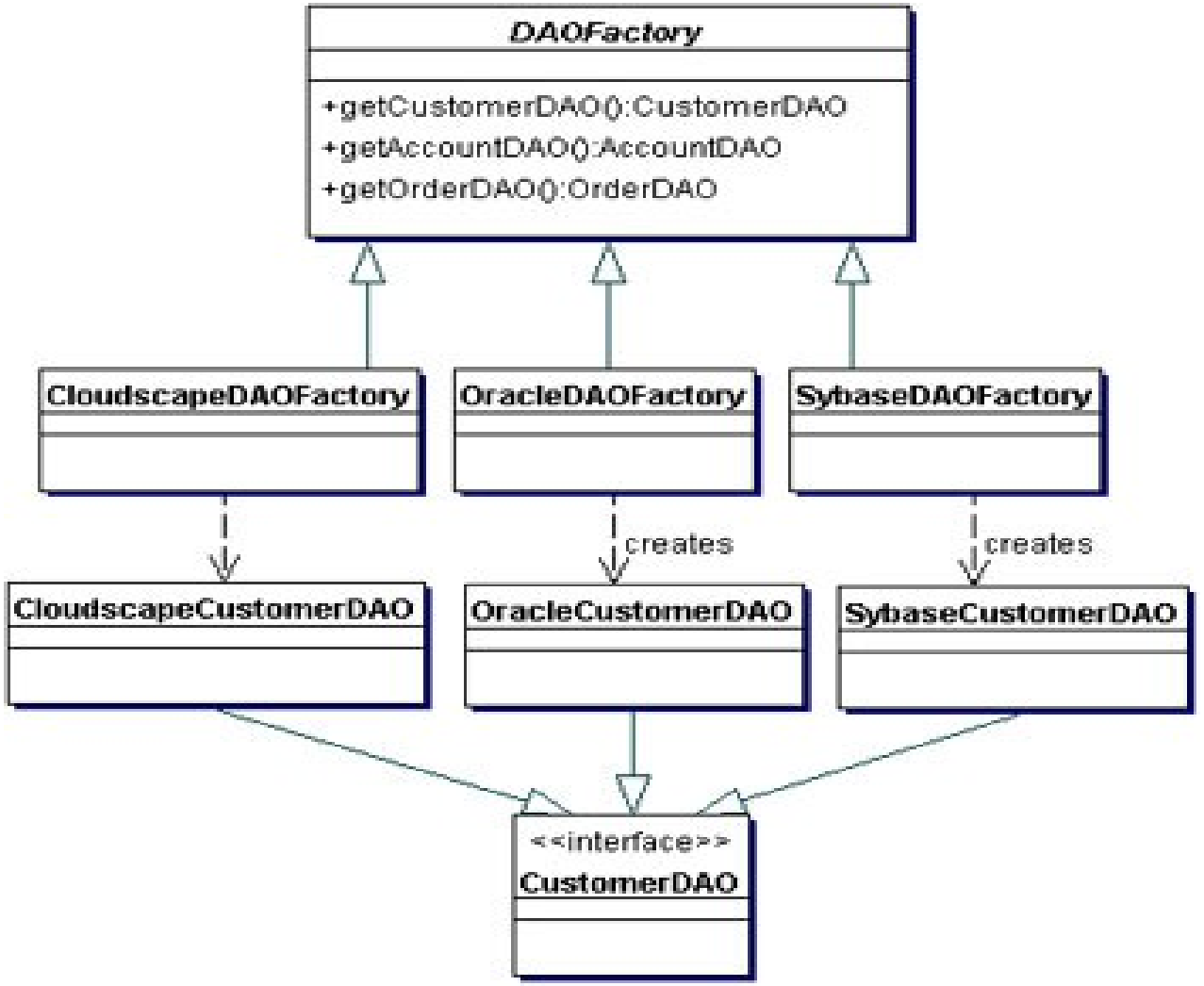
- *SQLWarning* (de exemplu, *DataTruncation*)
  Connection, Statement, ResultSet - getWarnings()

Data Acces Objects (DAO)



- *BusinessObject* - obiectul care doreste sa acceseze datele
- *DataAccessObject* - abstractizeaza si incapsuleaza toate operatiile legate de date
- *DataSource* - sursa datelor: RDBMS, OODBMS, XML, etc.
- *TransferObject* - reprezentare a datelor: entitati, beanuri, etc.

Abstract Factory



Java Tutorial

Trail: JDBC(TM) Database Access

http://docs.oracle.com/javase/tutorial/jdbc/TOC.html