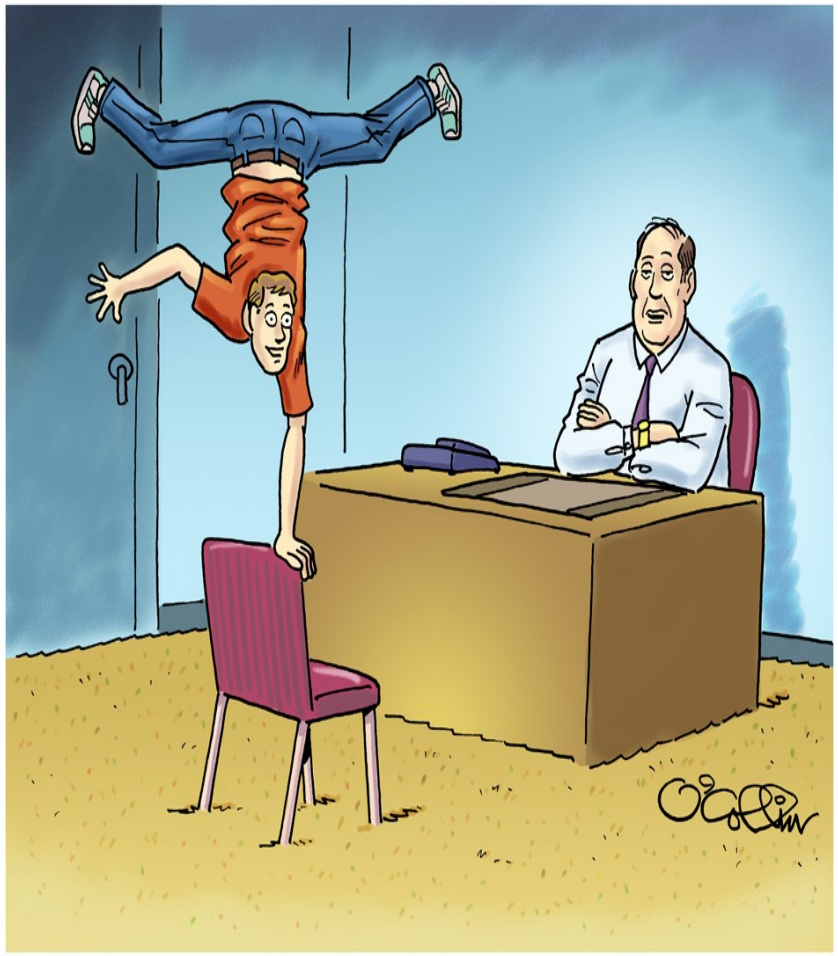
Connessione ai DB con JDBC

Prof. Francesco Viglietti
Www.in4matika.altervista.org



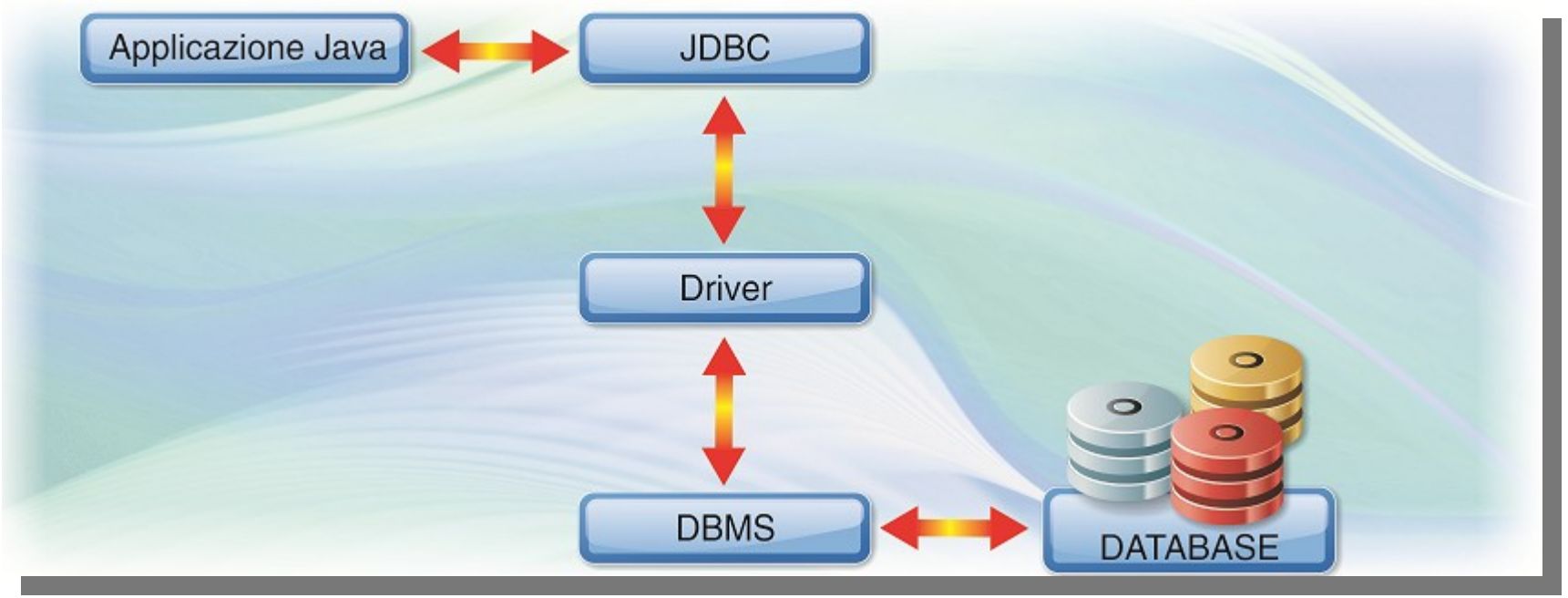
" Yes, you are a developer and yes, you're agile but that doesn't necessarily make you an agile developer. "

Connessione ai database

- **driver per database:** programma standard che consente di trasferire i dati presenti in un database.
- **ODBC** (*Open DataBase Connectivity*): interfaccia software standard in ambiente Windows.
- **DSN** (*Data Source Name*): nome della sorgente di dati.

JDBC

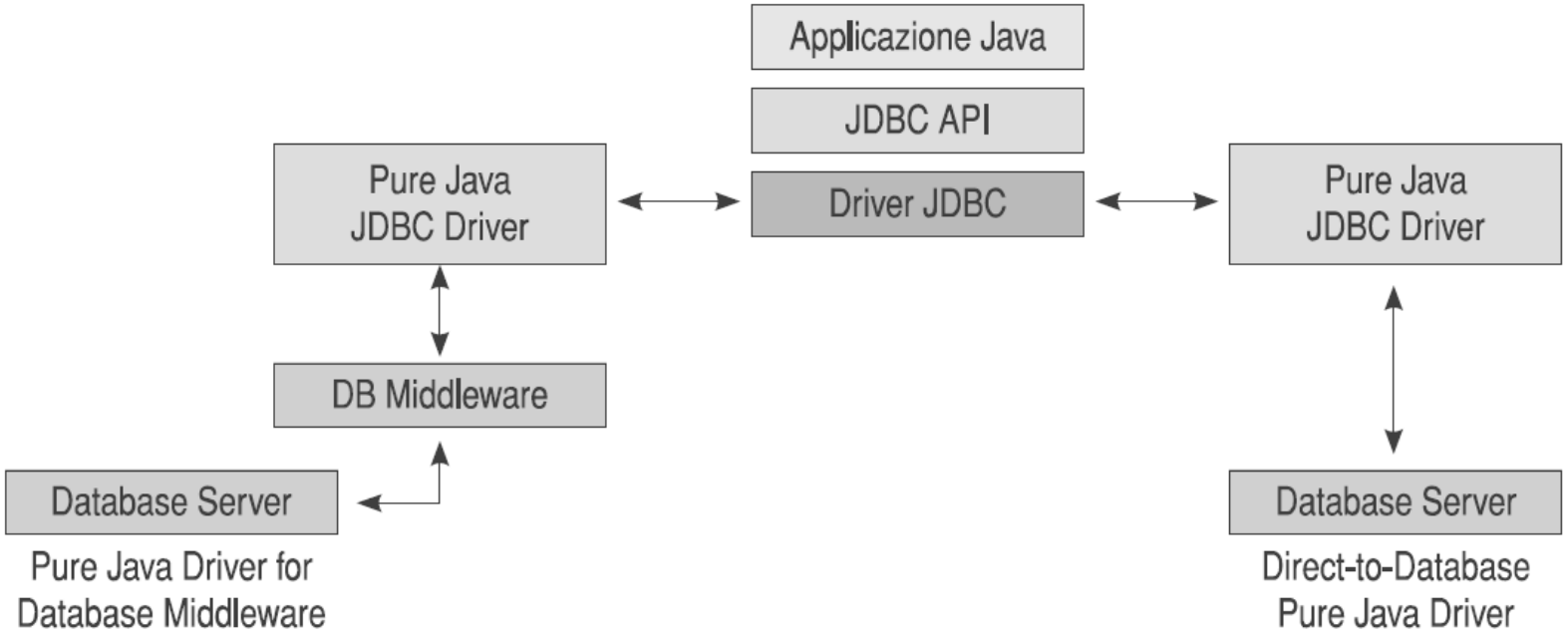
- **JDBC** (*Java DataBase Connectivity*): libreria di classi Java per interfacciare l'accesso ai database che usano il linguaggio standard SQL.



JDBC

Tipo 4. Direct-to-Database Pure Java Driver

Questa tipologia è quella maggiormente indicata alle applicaizoni Java in quanto il driver è realizzato totalmente in Java e trasforma le chiamate JDBC direttamente nel protocollo del database, quindi permette un collegamento diretto senza bisogno di middleware o client installati.



JDBC

Dal punto di vista del programmatore è completamente trasparente il tipo di DB che sta utilizzando in quanto interagisce con tutti mediante la stessa interfaccia, cioè JDBC, che si connette al DB fisico semplicemente cambiando il "nome del driver" da utilizzare nella sua applicazione.

Driver

Utilizziamo il driver JDBC chiamato *Connector/J*, che può essere prelevato dalla pagina http://dev.mysql.com/downloads/connector/j/.

Affinché tale driver possa essere visto dalle applicazioni Java, è necessario notificare alla macchina virtuale la presenza dell'archivio JAR contenuto nel pacchetto scaricato.
È possibile operare in diversi modi e il più semplice è depositare una copia dell'archivio JAR di Connector/J all'interno della *directory jre/lib/ext*, nella cartella che contiene l'installazione di Java.
La tecnica più efficace (e più corretta!) è quella di aggiungere il percorso dell'archivio JAR nella variabile di sistema CLASSPATH.

JDBC

- Importazione di **import java.sql.***;
- Classe **DriverManager:** driver disponibili
- Comando **Class.forName(*"nome del driver"*)**

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

Gestione delle eccezioni

```
try
{
 class.forName("com.mysql.jdbc.Driver");
}
catch(ClassNotFoundException e)
{
 System.err.print("ClassNotFoundException:");
 System.err.println(e.getMessage());
}
```

Connessione a un database

Come secondo passo si apre una connessione verso un database necessario all'applicazione, sfruttando il driver caricato al passo precedente.

Definiamo due variabili.

1. La prima per memorizzare l'indirizzo del database: la sintassi desiderata da Connector/J è la seguente:

```
jdbc:mysql://[hostname][:port]/[dbname][?param1=val1][&param2=val2]...
```

nel nostro caso stiamo eseguendo MySQL nella stessa macchina che eseguirà la classe (localhost) e la porta ascoltata dal DBMS è la numero 3306, valore di default predefinito di MySQL.

```
String url = "jdbc:mysql://localhost:3306/db1";
```

Se hostname è assente verrà contattato 127.0.0.1(localhost), Se è assente la porta sarà usata la 3306 (default di mysql)
Jdbc:mysql:///pluto si connette al db pluto sulla macchina locale dalla porta 3306
Jdbc:mysql://db.rcost.in4matika.org:3388/personale connette al db personale sulla macchina db.rcost.in4matika.org dalla porta 3388

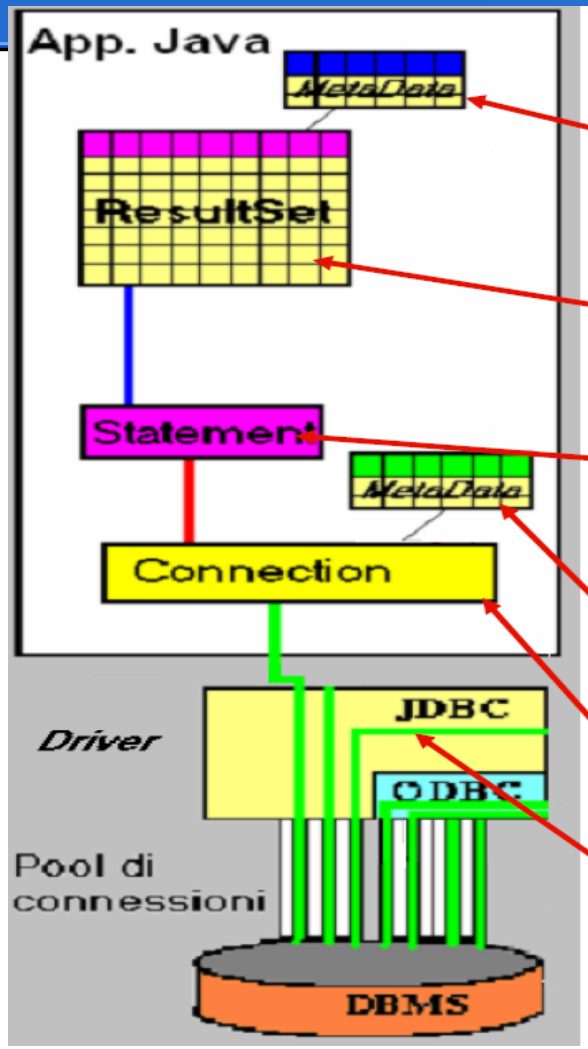
Connessione a un database

```
Connection conn = null;
try{
conn=DriverManager.getConnection(url,"username","pswd";
}
catch(Exception e){
System.out.print(“Errore di connessione”+e);
System.exit(1);
}
```

Siccome le connessioni occupano risorse, in un ambiente multiutente e multitasking è opportuno adottare la seguente politica:

- aprire una connessione solo quando necessario;
- assicurarsi di chiuderla al termine del suo utilizzo;
- e soprattutto non aprire/chiudere connessioni inutilmente.

Riepilogo



Gli oggetti **ResultSetMetaData** contengono informazioni circa il ResulSet (numero e tipo delle colonne, possibilità di movimento, possibilità di scrittura)

Gli oggetti ResultSet contengono le tabelle risultanti dall'esecuzione dei comandi SELECT.

Gli oggetti Statement rappresentano i comandi che è possibile far eseguire al DBMS

Gli oggetti ConnectionMetaData permettono di accedere a informazioni circa il DB ed il DBMS (SQL supportato, possibilità, schemi ....)

Gli oggetti Connection rappresentano i canali di comunicazione con il DBMS

I driver sono responsabili di tradurre i comandi dati tramite le API JDBC in opportune istruzioni al DBMS

L’invio di comandi

Tramite una connessione è possibile inviare al database comandi modellati da tre diversi tipi di oggetti:

1. Statement: sono creati dal metodo ***createStatement***, rappresentano comandi SQL senza parametri.

2. PreparedStatement: creati dal metodo ***prepareStatement***, rappresentano comandi SQL che possono essere precompilati nel database ed accettare parametri (di INPUT) al momento dell’esecuzione

3. CallableStatement: creati dal metodo ***prepareCall***, che possono essere usati per eseguire ***stored procedure*** ed accettano anche parametri di OUTPUT e INPUT/OUTPUT.

Statement

Se la connessione va a buon fine, viene creato un oggetto java.sql.Statement, necessario per interagire con il DBMS attraverso delle query espresse in linguaggio SQL che memorizzano il risultato in un oggetto java.sql.ResultSet:

- Metodo **createStatement (**crea un'istanza della classe **Statement**)

```
Statement stmt = conn.createStatement();//crea statem.
```

Metodo per la Manipolazione dei dati

- Metodo **executeUpdate** per i comandi SQL: *Insert*, *Update*, *Delete*. (comandi per inserimento,aggiornamento e cancellazione)

```
String cmdSQL = "INSERT INTO Anagrafica . . . "
stmt.executeUpdate(cmdSQL);//esegue comando
System.out.print("Record registrato");
stmt.close();
conn.close();
```

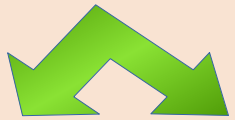
Ancora su Manipolazione

- Vediamo come fornire un input alla query senza usare i parametri,ma usando create **Statement**

```
Statement stmt = conn.createStatement();
```

- Creazione di una stringa che contenga il comando da eseguire, a cui si accoda l'input dell'utente. Alla fine si deve eseguire il metodo **executeUpdate()**

```
int id=in.nextInt();
String tel=in.next();
String tmp="UPDATE Anagrafica SET Telefono='"+tel+"'
WHERE ID="+id;
stmt.executeUpdate();
```



- N.B. tutti i comandi statement devono essere inseriti in eccezioni per essere eseguiti!

Manipolazione con parametri

- modo per fornire parametri direttamente alla query
- Metodo **prepareStatement**

```
PreparedStatement stmt = conn.prepareStatement(
"UPDATE Anagrafica SET Telefono=? WHERE ID=?");
```

- Per assegnare i valori ai parametri identificati con **?**: scegliere il progressivo ed associare il metodo **set*XXX*** sostituendo ad XXX il tipo di dato. Alla fine si deve eseguire il metodo executeUpdate()

```
stmt.setString(1, "02/8883917");
stmt.setInt(2, 365);
stmt.executeUpdate();
```

N.B. tutti i comandi statement devono essere inseriti in eccezioni per essere eseguiti!

Interrogazioni in SQL

- Metodo **executeQuery**

```
String cmdSQL = "SELECT a,b,c FROM Tabella1";
ResultSet rs = stmt.executeQuery(cmdSQL);
```

- Valore di ritorno: **ResultSet.**

Il metodo next() di un oggetto resultSet, ha duplice funzionalità:

- scorre in avanti l'elenco dei record ottenuti;
- restituisce true fin quando non si giunge al termine dell'esplorazione.

Ciclo di lettura

- Metodi **get*XXX***, dove XXX va sostituito con uno dei tipi di dato.

```
ResultSet rs =
stmt.executeQuery("SELECT a,b,c FROM Tabella1");
while (rs.next()) {
 int i = rs.getInt("a");
 float f = rs.getFloat("b");
 String s = rs.getString("c");
 System.out.print(i+”:“+f+”:“+s+”\n”);
}
```

Input nelle interrogazioni

- Come fatto precedentemente per la manipolazione possono essere inviati input utente nelle interrogazioni SQL concatenando la stringa che contiene la query. Teniamo presente che le stringhe e le date in sql devono essere racchiuse tra singoli apici ‘ ‘.

```
prov = in.nextLine();
//inserimento provincia e creazione statement
Statement stmt = con.creteStatement();
//esecuzione query e restituzione del resultset
ResultSet rs = stmt.executeQuery(
“SELECT ID, Cognome, Nome, CodiceFiscale FROM
Anag WHERE Provincia =’”+prov+”’”);
```

Parametri nelle query

- Comando **prepareStatement : viene preparata la** Select SQL con parametro. Come in precedenza si sostituisce al posto del parametro indicato con ? Il progressivo ed il valore e quindi si invia la query

```
PreparedStatement stmt = con.prepareStatement(
"SELECT ID, Cognome, Nome, CodiceFiscale FROM
Anag WHERE Provincia = ? ");
prov = in.nextLine();//inserimento provincia
stmt.setString(1, prov);//inserimento nella query
//invio query e ritorno resultset
ResultSet rs = stmt.executeQuery();
```

Classi e metodi

- Classi e metodi per la connessione al database e per le operazioni di manipolazione e interrogazione.

Metadati del database

- Classe **DatabaseMetaData:** ci permette di avere le informazioni sulla struttura del database (**metadati**)
- Metodo **getMetaData** di un **ResultSet** ottenuto con una query:
  restituisce un oggetto **ResultSetMetaData**

```
ResultSet rs = stmt.executeQuery(query);
ResultSetMetaData rsmd = rs.getMetaData();
```

Metadati del database

Alcunu metodi forniti dalla classe **DatabaseMetaData**, applicati all'oggetto **ResultSetMetaData**:

- **getColumnCount**
- **getColumnName**
- **GetColumnTypeName**
- **...**

Esempio classe di gestione DB

```
public class GestDatabase {
    private String nome, url;
    // oggetti tipici per integrare DB nell'applicazione Java
    private Connection conn;
    private Statement stm;
    private ResultSet rs;
    private ResultSetMetaData rsmd;
    //costruttore della classe si carica il driver e l'indirizzo del DB
    public GestDatabase(String nome){
        this.nome=nome;
        try {  //carico il driver
            Class.forName("com.mysql.jdbc.Driver"); }
        catch(ClassNotFoundException e1) {
          System.out.println("Driver non trovato....."); System.exit(1);
        }
        // nome ed indirizzo del database
        url="jdbc:mysql:///"+nome;
    }
```

Esempio classe di gestione DB

```
// si inseriscono i vari getter e setter….
// metodo per inizializzazione connessione, ed invio query d'interrogazione
public void dirQuery(String query){
      try {  //apre la connessione verso il DB
          setConn(DriverManager.getConnection(url,"root",""));
      }
      catch (SQLException e) {
          System.out.println("Errore nella connessione"+e); System.exit(1);
      }
//creazione statement per interagire con il DB
      try {
          stm=conn.createStatement();
     //interrogo il DB con una query sql e ricevo il risultato nel resultset
          rs=stm.executeQuery(query);
    //richiedo anche i metadati relativi al resultset
          rsmd=rs.getMetaData();
      }
      catch (SQLException e) {
          System.out.println("Errore nella query"); System.exit(1);
      }
}
```

Esempio classe di gestione DB

```
public void dirExec(String query){//metodo per inizio connessione, e invio query
        try {  //apre la connessione verso il DB
            setConn(DriverManager.getConnection(url,"root",""));
        }
        catch (SQLException e) {
            System.out.println("Errore nella connessione"+e); System.exit(1);
        }
       try { //creo uno statement per interagire con il DB e lo invio direttamente
           stm=conn.createStatement();
           stm.executeUpdate(query);
           stm.close();
        }
        catch (SQLException e) {
            System.out.println("Errore nella query"); System.exit(1);
        }
    }
public void closeConn() throws SQLException{
        conn.close();
    }
public void closeRs() throws SQLException{
        rs.close();
    }
}
```