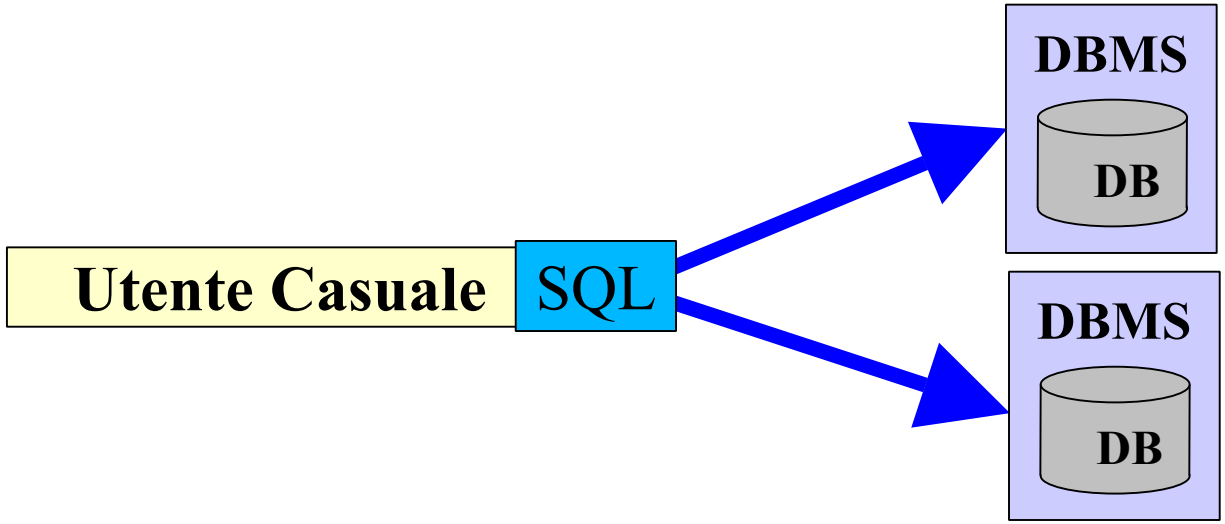
Dati relazionali e JAVA: API JDBC

Obiettivi

- Impareremo a utilizzare l'API JDBC, che permette a un programma scritto in Java di connettersi e manipolare dati relazionali.
- Motivazioni della nascita di JDBC.
- Connessione a una sorgente di dati tabellari.
- Manipolazione dei dati.

DBMS e SQL

- La necessita' di manipolare basi di dati indipendentemente dal sistema utilizzato ha portato alla standardizzazione di SQL.
- **Semplici comandi** SQL sono utilizzabili su qualsiasi sistema.

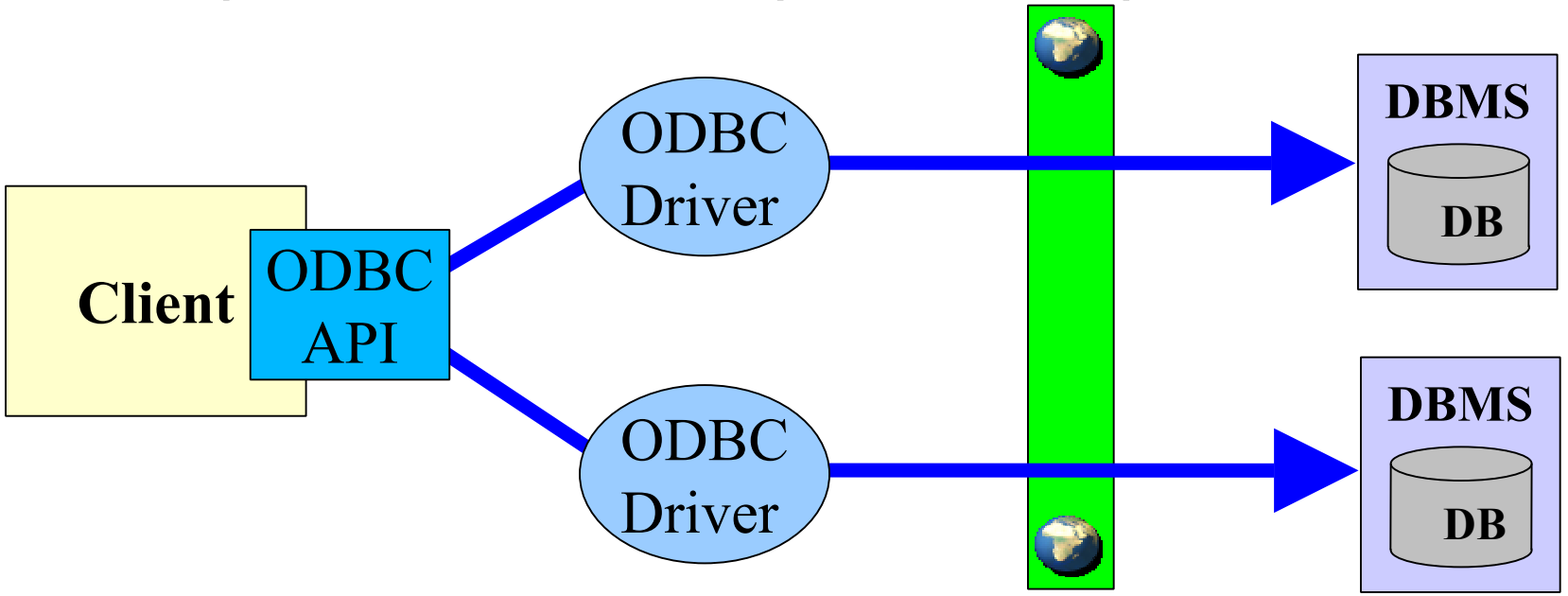


DBMS e SQL

- Purtroppo, SQL non e' sempre sufficiente.
- E' spesso necessario immergere SQL in altri linguaggi (applicazioni).
- Applicazioni che utilizzano SQL devono comunicare con i DBMS.
- Inizialmente, la connessione alle basi di dati variava a seconda del sistema.

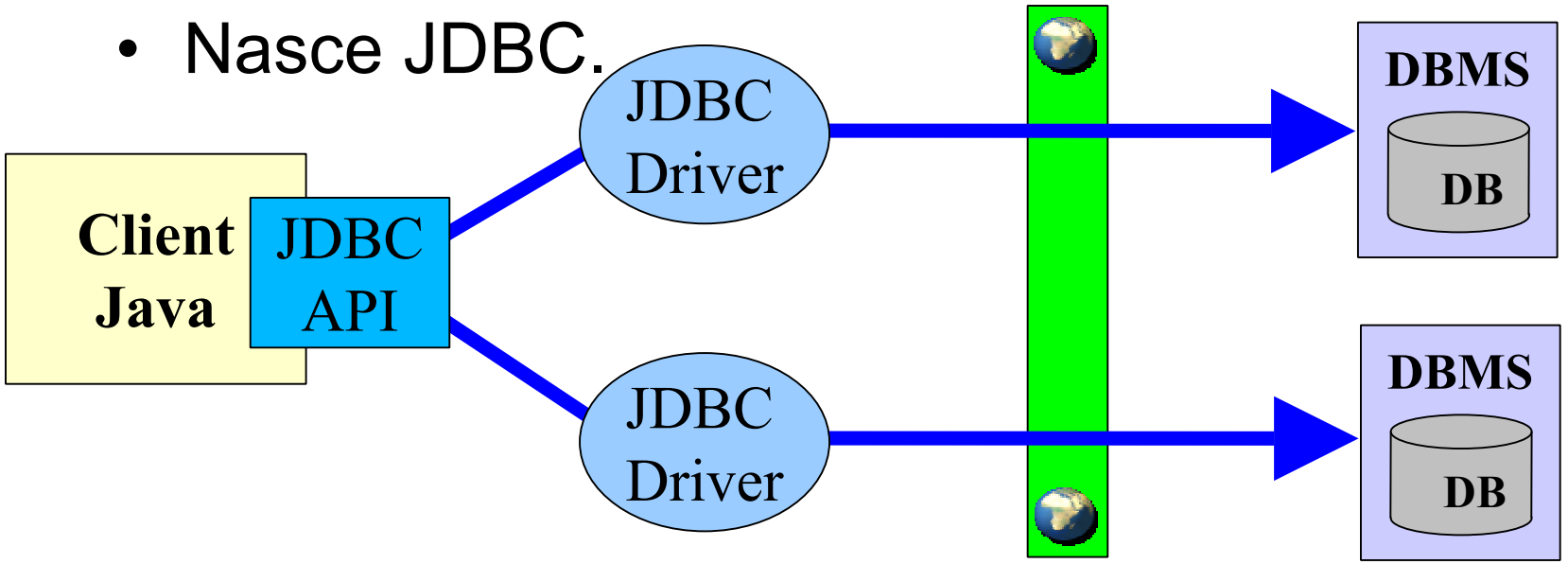
DBMS, SQL e ODBC

- Per ovviare a queste difficolta', Microsoft ha prodotto ODBC (Open Database Connectivity).
- ODBC e' un'API, scritta in C, che tramite driver specifici rende trasparente il tipo di DBMS.



DBMS, SQL, ODBC e Java

- Il linguaggio C non e' portabile, e i driver ODBC devono essere installati su ogni client.
- Problematico in caso di client via internet.
- Ricco di opzioni ➡ complicato.
- Nasce JDBC.



Principali passi nell'utilizzo di JDBC

- Apertura di una connessione con un DBMS.
- Scambio di dati (query e aggiornamenti).
- Chiusura della connessione.

Connessione a un database

- La connessione a un database avviene attraverso un Driver, che dipende dal sistema con il quale si vuole comunicare.
- Il Driver traduce i metodi dell'API JDBC in istruzioni comprensibili al DBMS utilizzato.
- Bisogna mettere a conoscenza il programma java di dove reperire il Driver necessario.
- Questa operazione si chiama *registrazione* del Driver.

Registrazione del Driver

- Inizialmente e' necessario creare una istanza del Driver.

Class.forName(*nome_del_driver*);

- Il nome del driver identifica una classe Java.
- La prima parte del nome identifica le directory dentro alle quali cercare la classe.
- "com.cloudscape.core.RmiJdbcDriver"
- "com.mysql.jdbc.Driver"
- "org.postgresql.Driver"

Registrazione del Driver

- Invece di utilizzare un metodo all'interno del programma, e' possibile registrare il Driver passandolo come parametro, al momento dell'esecuzione.

$ java -Djdbc.drivers = *Driver* App.*class*

Apertura della connessione

- A questo punto si apre la connessione.
- Il metodo DriverManager.getConnection() riceve come parametro un URL che identifica un server DBMS.
- A partire dalla struttura dell'URL, viene automaticamente cercato un Driver che possa gestirlo.
- Il metodo, tramite il Driver, apre una connessione e ritorna un oggetto di tipo Connection.

Apertura della connessione

- Gli url dipendono dal Driver e dal server, e tipicamente hanno una struttura simile ai seguenti esempi:
- jdbc:mysql://localhost/demo
- jdbc:postgresql://localhost:8000/test
- jdbc:cloudscape:rmi:db1

Riepilogo: ottenere una connessione

```
Class.forName(driver).newInstance();
try{
    con=DriverManager.getConnection(url,"","");
    // Qui utilizzare la connessione
}
catch (SQLException sql) {// gestione eccezione...}
finally {
    try {con.close();}
    catch (SQLException ex) {// gest. eccezione...}}
```

Riepilogo: ottenere una connessione

- In pratica, una connessione si puo' aprire tramite due righe di codice:

```
Class.forName(driver).newInstance();
con=DriverManager.getConnection(url,"","");
```

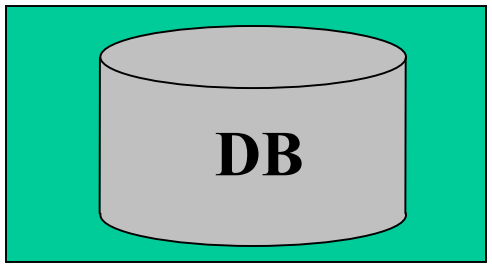
- Se il Driver viene registrato dall'esterno, e' addirittura sufficiente una sola istruzione.
- Vediamo brevemente come viene aperta la connessione.

Esmpio di apertura di una connessione

DriverManager

Classe del framework java

Applicazione java

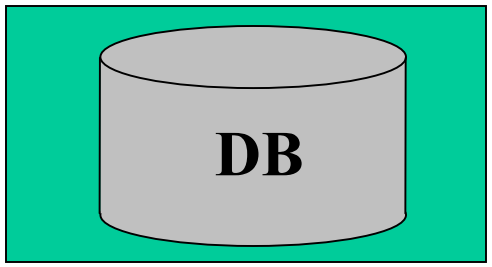


DBMS

Caricamento (creazione) di un Driver

DriverManager

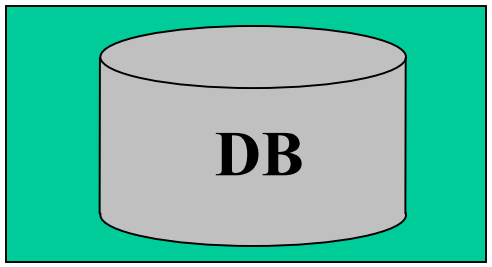
Class.forName(driver).
newInstance();



Registrazione del Driver

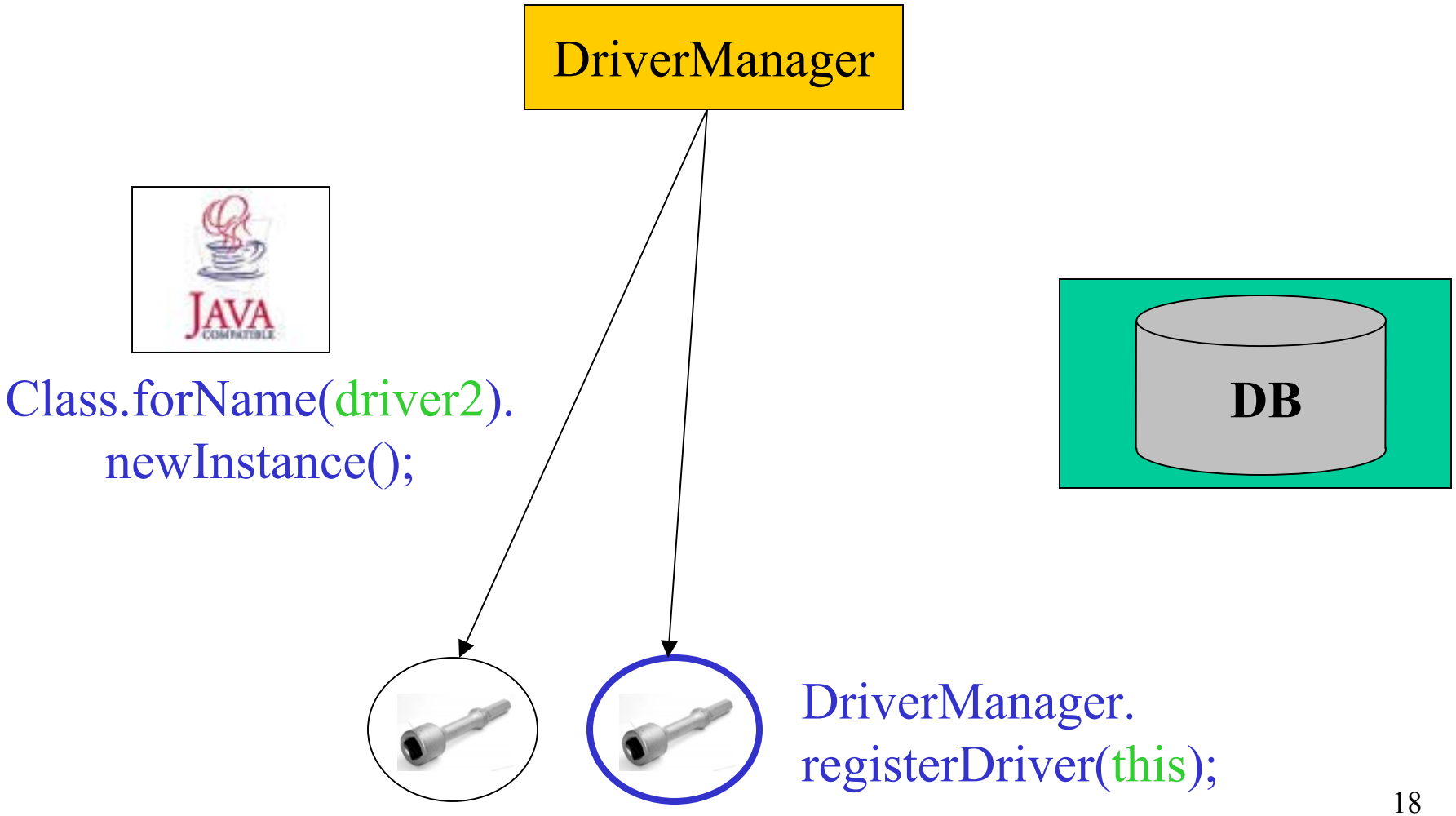
DriverManager

Class.forName(driver).
newInstance();

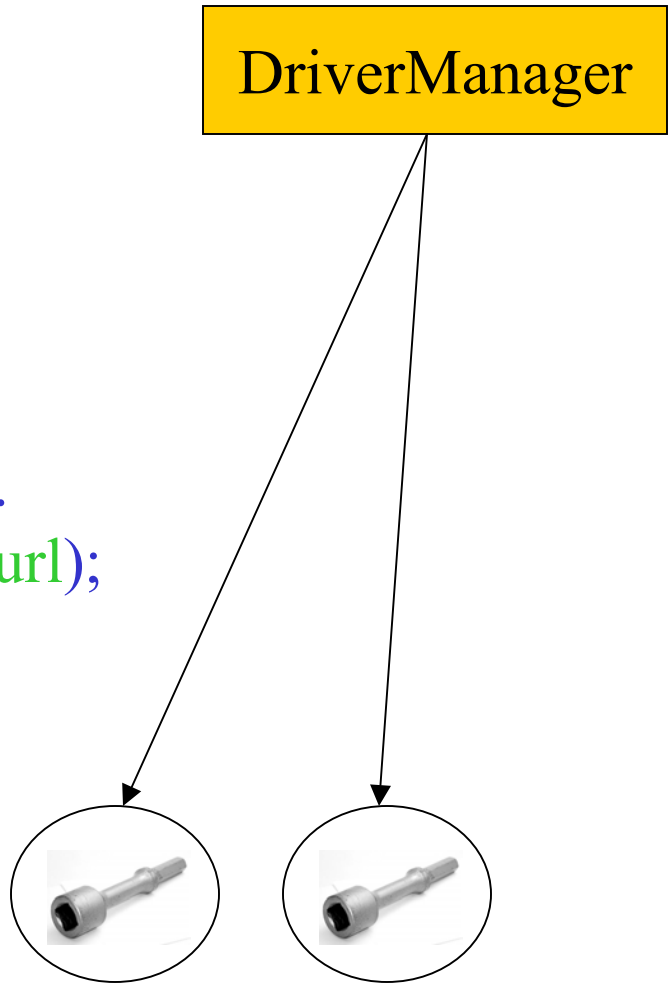


DriverManager.
registerDriver(this);

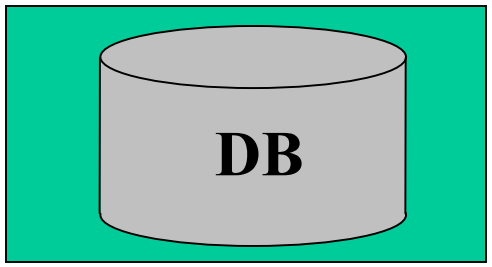
Eventuale registrazione di altri Driver



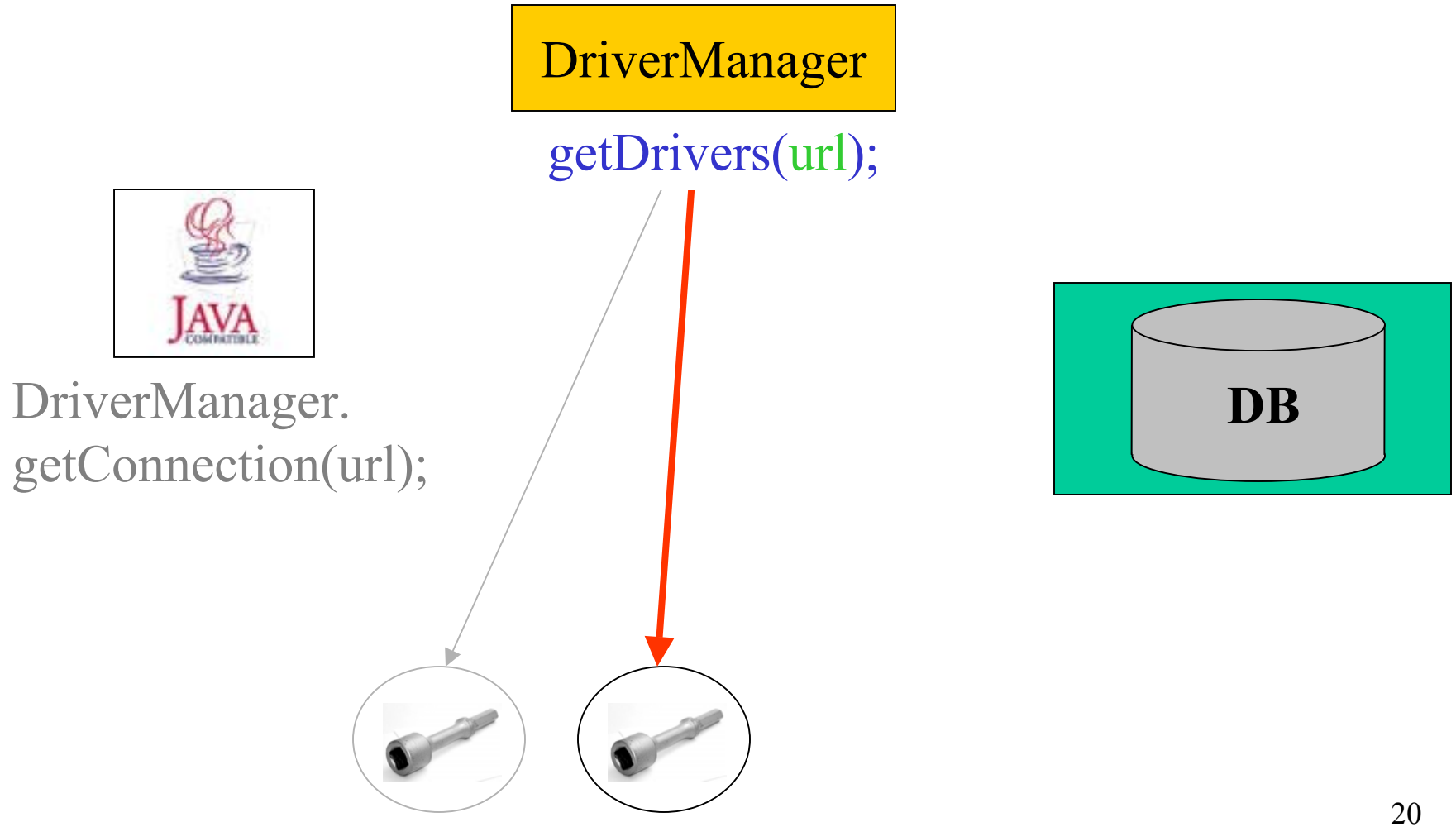
Richiesta di connessione



DriverManager.
getConnection(url);



Selezione del Driver opportuno

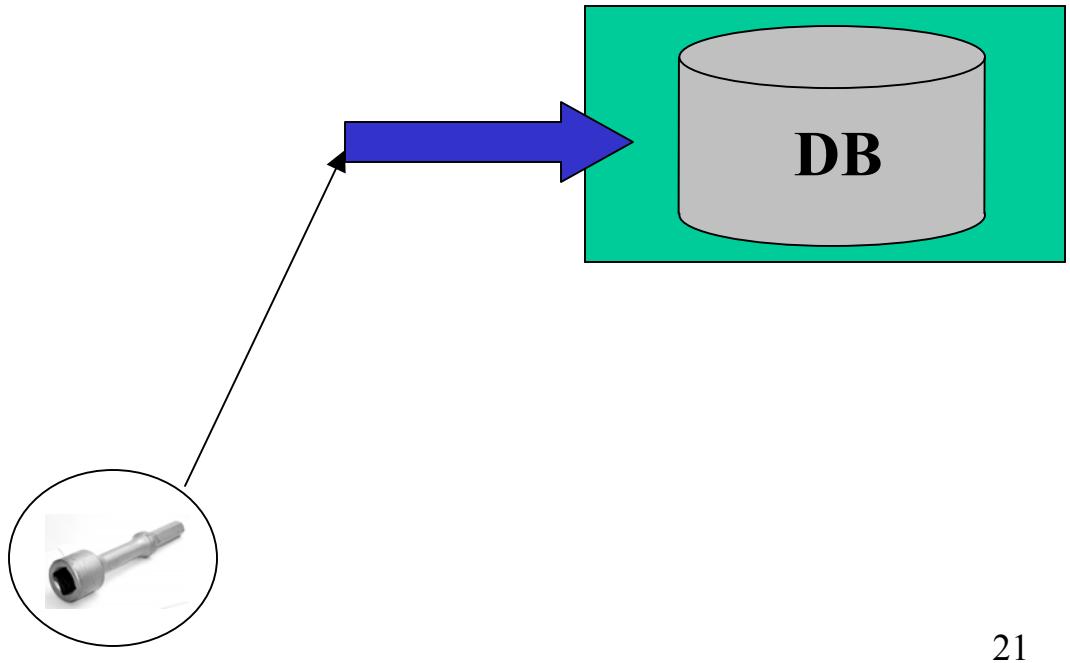


Apertura della connessione

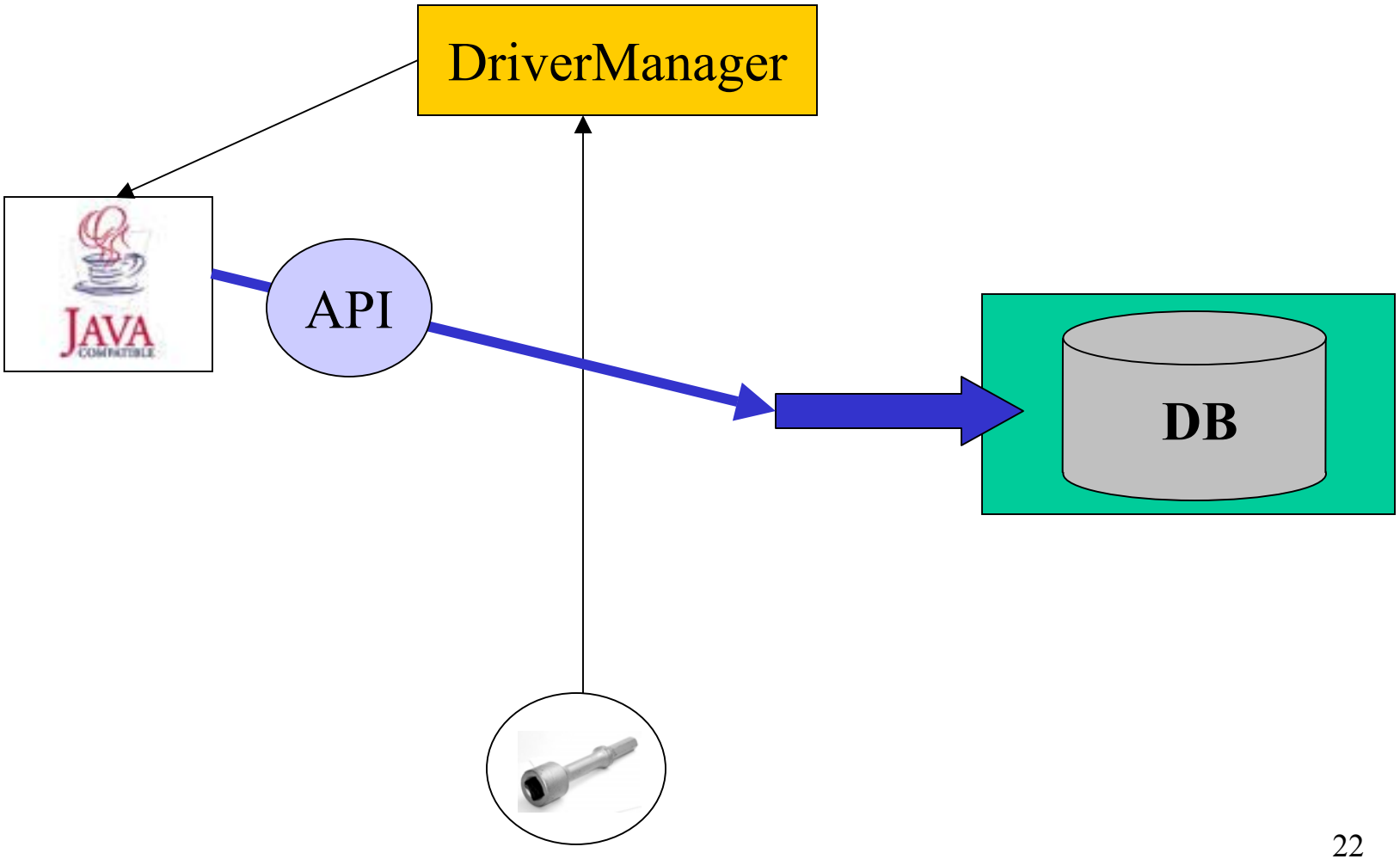
DriverManager

Driver.connect(url);

DriverManager.
getConnection(url);



Restituzione della connessione



Esempio di interrogazione SQL

- Prima di analizzare i dettagli di JDBC, vediamo un esempio di una tipica e semplice espressione SQL:

```
SELECT Cognome, Stipendio
FROM Impiegato
```

- Le istruzioni SQL che vogliamo valutare sono utilizzate direttamente dai metodi Java.

```
stringaSQL = "SELECT Cognome, Stipendio " +
             "FROM Impiegato"
```

Esempio di interrogazione SQL

- Questa interrogazione si esegue facilmente in due passi:
- Si ottiene dalla connessione un oggetto Statement:

```
Statement sttm = con.getStatement();
```

- Si esegue la query e si ottiene il risultato:

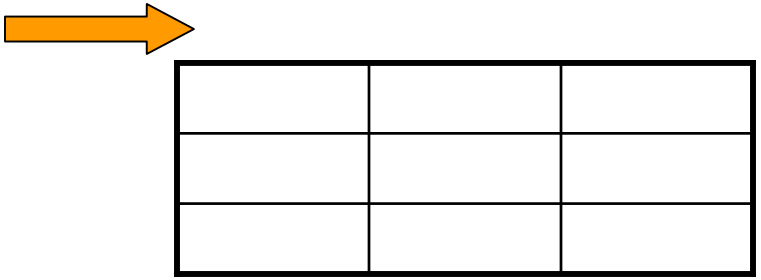
```
ResultSet rs = sttm.executeQuery(stringaSQL);
```

Ottenere il risultato di una interrogazione

```
Class.forName(driver).newInstance();
try{
    con=DriverManager.getConnection(url,"","");
    Statement sttm = con.getStatement();
    ResultSet rs = sttm.executeQuery(stringaSQL);
}
catch (SQLException sql) {// gestione eccezione...}
finally {con.close();}
```

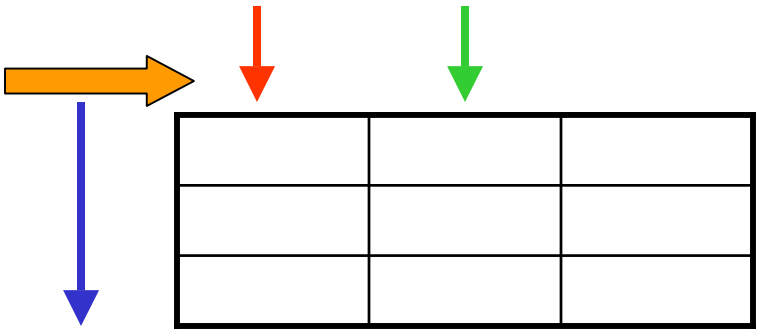
Utilizzo del risultato

- Un ResultSet contiene il risultato dell'interrogazione, accessibile tramite un **cursore** e alcuni metodi per recuperarne il contenuto.
- Il cursore inizialmente e' posizionato prima della prima riga.
- La riga successiva si ottiene tramite il metodo **next**(), che ritorna *true* quando tale riga e' presente.
- Quando viene raggiunta l'ultima riga, next() ritorna *false* e il ResultSet viene chiuso automaticamente.



Utilizzo del risultato

```
while( rs.next() ) {
    rs.getString(1);
    rs.getInt(2);
    // Estrarre valori da tutte le colonne richieste
}
```



Scrollable e Updatable Result Sets

- Gli oggetti di tipo ResultSet visti finora sono percorribili dalla prima all'ultima riga una volta soltanto, e non possono essere aggiornati.
- E' possibile utilizzare ResultSet percorribili tramite un accesso random e aggiornabili.
- Per maggiori informazioni:

  http://java.sun.com/j2se/1.4.2/docs/api/

  → ALL CLASSES → ResultSet

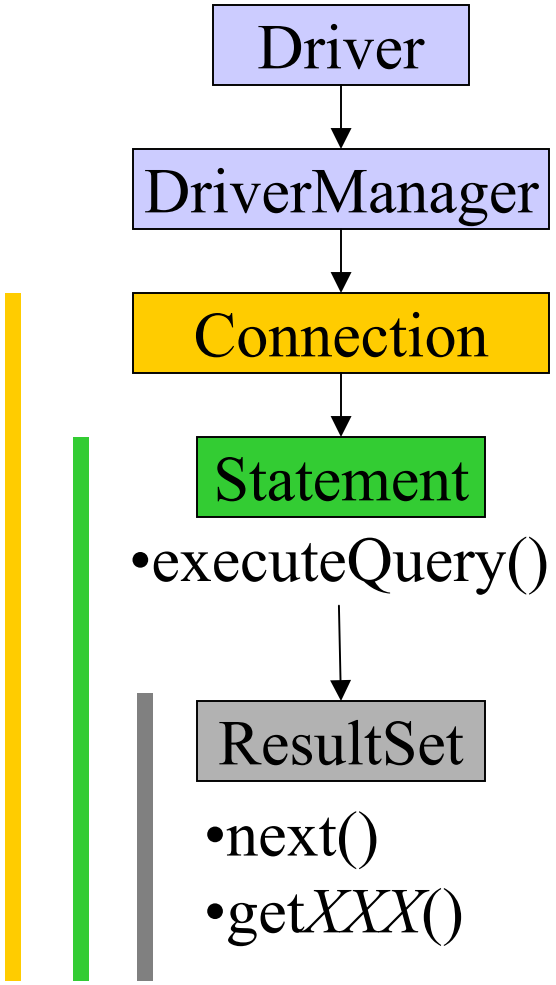
Equivalenze di tipo

Tipo SQL	**Metodo Java**
BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()

Equivalenze di tipo

LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

Punto della situazione



Un esempio di utilizzo

- Vedremo ora un esempio pratico di come JDBC possa contribuire al fine di riutilizzare la stessa applicazione con diversi DBMS.
- Per rendere l'esempio piu' significativo, introduciamo una nuova classe che ci permette di ottenere informazioni sul DBMS a cui ci connettiamo.
- Tale classe si puo' ottenere tramite il metodo Connection.getMetaData().

DatabaseMetaData

- Gli oggetti di tipo DatabaseMetaData mettono a disposizione moltissimi metodi, che potranno essere scelti volta per volta utilizzando la documentazione della classe.
- Nell'esempio utilizzeremo i due seguenti:
  - getDatabaseProductName()
  - getDatabaseProductVersion()

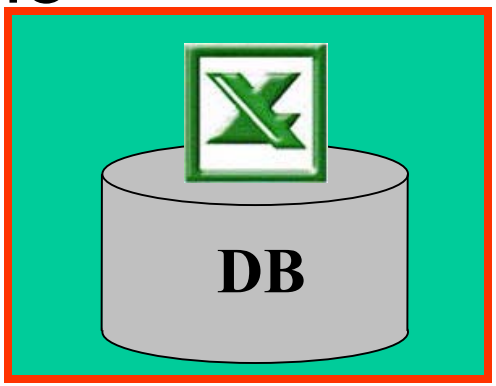
Elementi dell'esempio

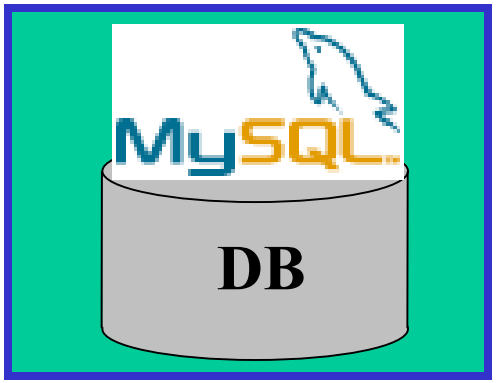
JDBCDemo.class

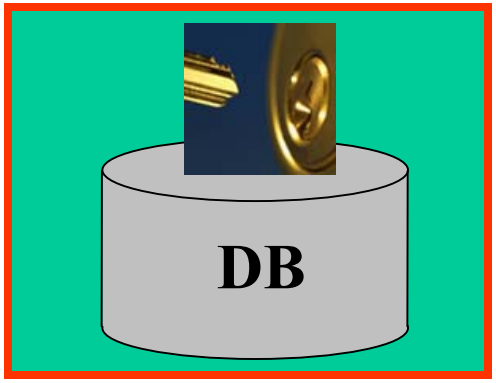
properties.cfg
properties2.cfg
properties3.cfg

com/mysql/jdbc/Driver

sun/jdbc/odbc/JdbcOdbcDriver







I file properties.cfg

properties.cfg

```
Driver: com.mysql.jdbc.Driver
url: jdbc:mysql://localhost/DEMO
```

properties2.cfg

```
Driver: sun.jdbc.odbc.JdbcOdbcDriver
url: jdbc:odbc:db1
```

properties3.cfg

```
Driver: sun.jdbc.odbc.JdbcOdbcDriver
url: jdbc:odbc:Impiegato
```

JDBCDemo.java (1)

```
String cfgFile = "properties.cfg";
// getting connection parameters
if (args.length != 0)
    cfgFile = args[0];
init(cfgFile);
```

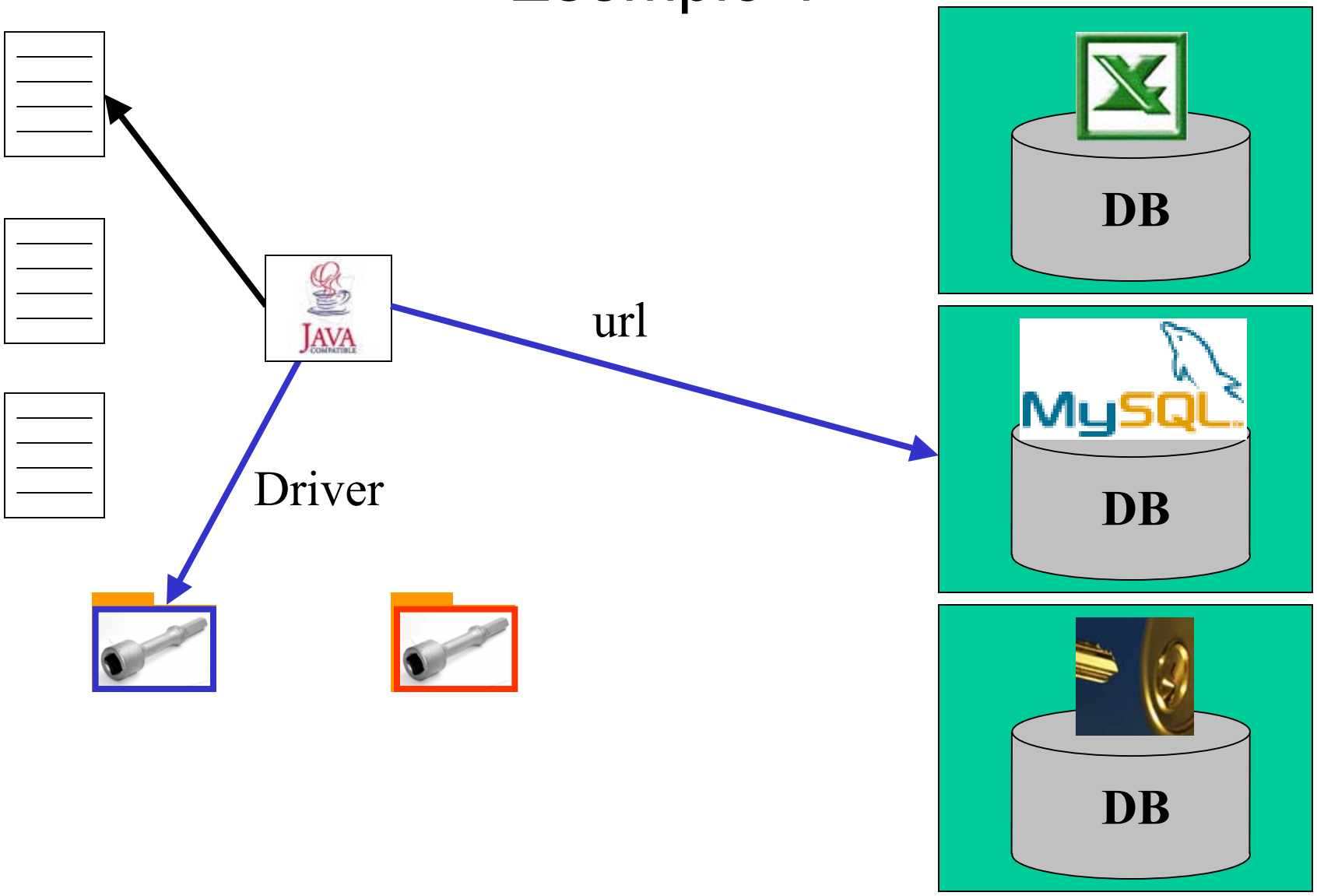
JDBCDemo.java (2)

```
String sqlQuery = "SELECT Cognome,
    Stipendio FROM Impiegato";
con = DriverManager.
    getConnection(url,"","");
DatabaseMetaData db =
    con.getMetaData();
System.out.println("DBMS: " +
    db.getDatabaseProductName() +
    " version " +
    db.getDatabaseProductVersion());
```

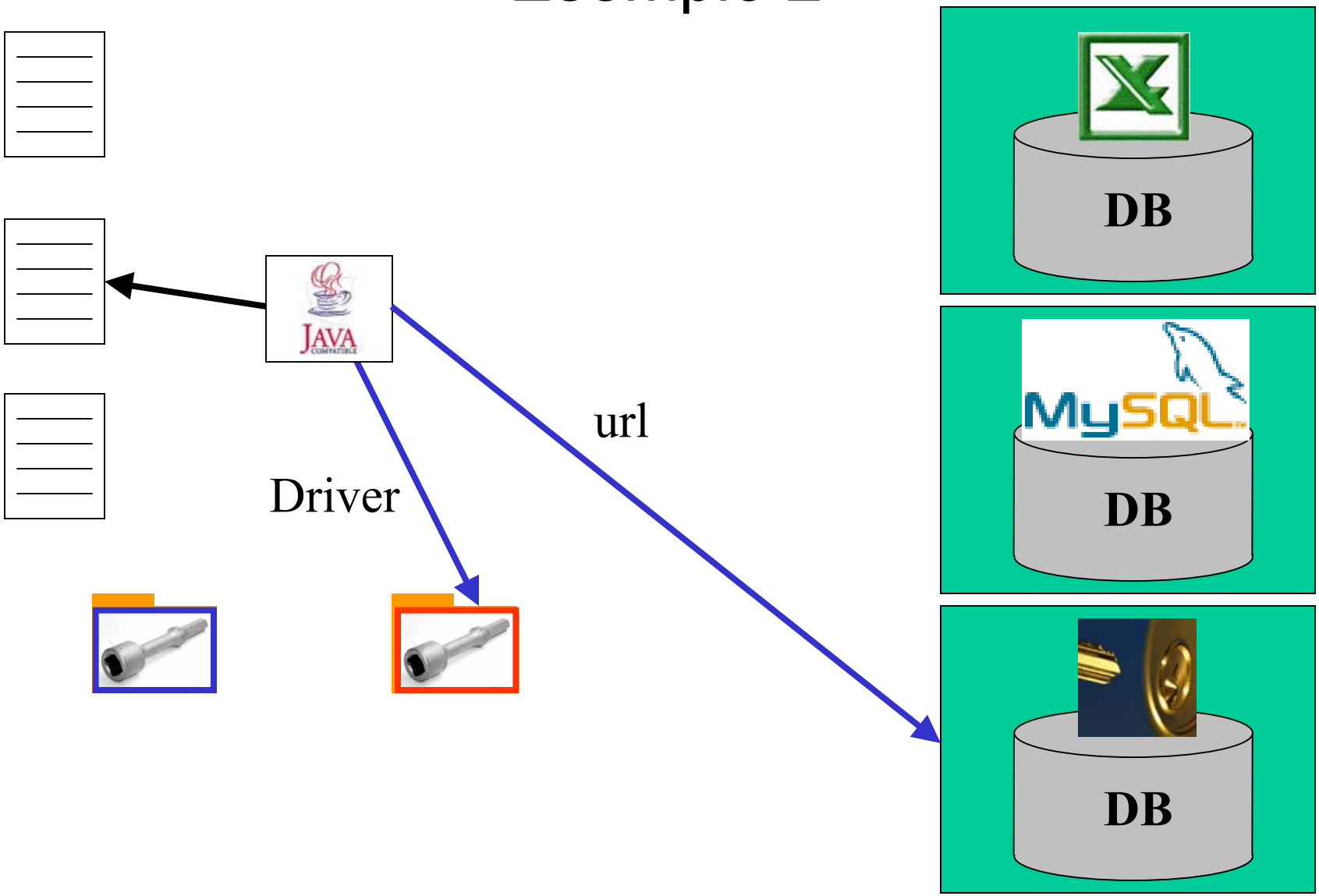
JDBCDemo.java (3)

```
Statement stmt =
    con.createStatement();
System.out.println(sqlQuery);
ResultSet rs =
    stmt.executeQuery(sqlQuery);
while (rs.next()) {
    System.out.print(rs.getString(1));
    System.out.println("\t" +
        rs.getInt(2));
}
```

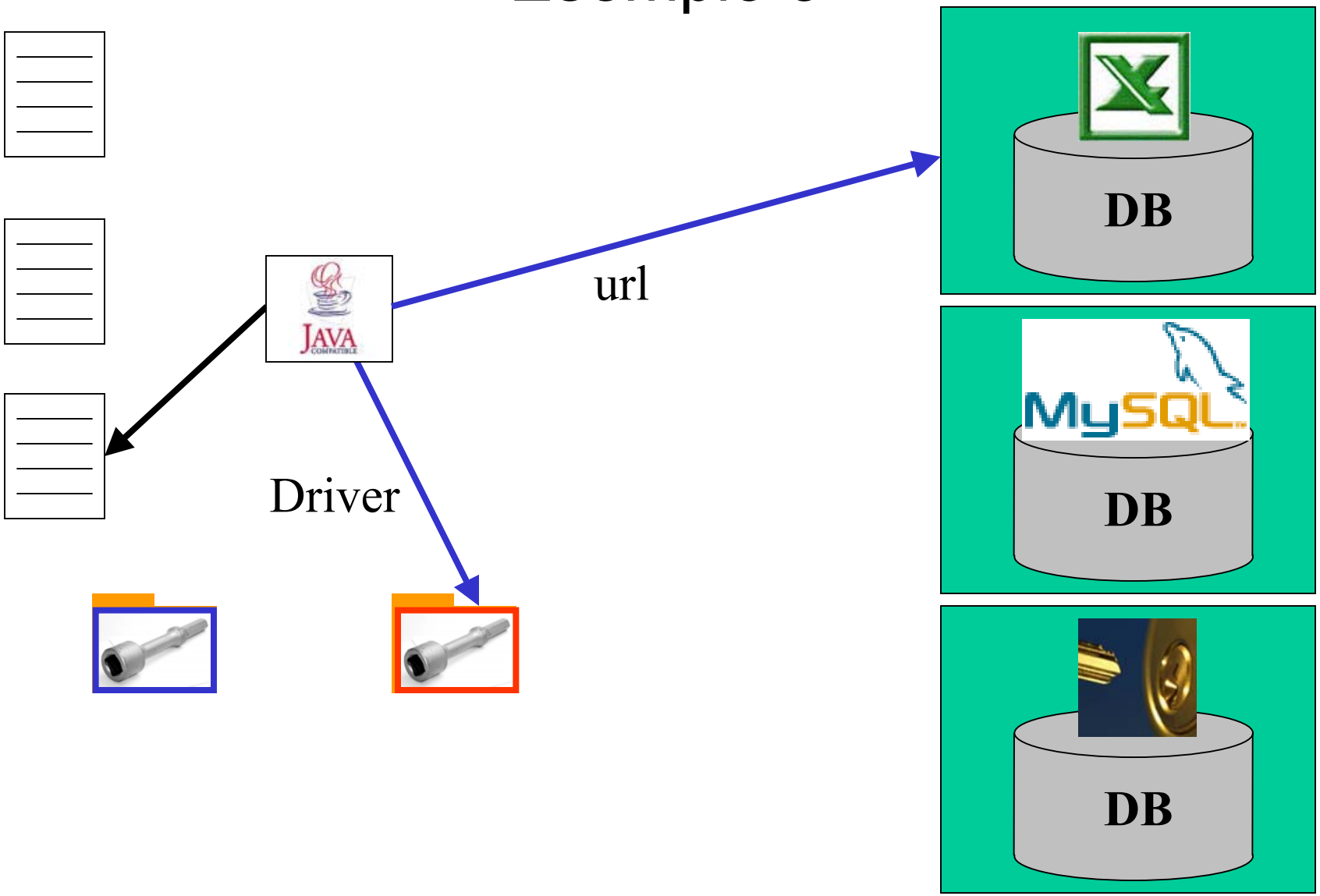
Esempio 1



Esempio 2



Esempio 3



Esecuzione esempio 1

```
E:\jdbc>java JDBCDemo properties.cfg
DBMS: MySQL version 4.0.16-nt
SELECT Cognome, Stipendio FROM Impiegato
Rossi   45
Bianchi 36
Verdi   40
Neri    45
Rossi   80
Lanzi   73
Borroni 40
Franco  46
```

Esecuzione esempio 2

```
E:\jdbc>java JDBCDemo properties2.cfg
DBMS: ACCESS version 04.00.0000
SELECT Cognome, Stipendio FROM Impiegato
Rossi   45
Bianchi 36
Verdi   40
Neri    45
Rossi   80
Lanzi   73
Borroni 40
Franco  46
```

Esecuzione esempio 3

```
E:\jdbc>java JDBCDemo properties3.cfg
DBMS: EXCEL version 08.00.0000
SELECT Cognome, Stipendio FROM Impiegato
Bianchi 36
Rossi   80
Neri    45
Verdi   40
Lanzi   73
Franco  46
Rossi   45
Borroni 40
```

Statement

- Abbiamo visto come un oggetto di tipo Statement possa essere utilizzato per compiere una interrogazione, tramite il metodo executeQuery().
- Lo stesso oggetto puo' anche essere utilizzato per aggiornare il database.
- A questo scopo si utilizza il metodo executeUpdate().

Statement

- Utilizzo di Statement.executeUpdate():
- UPDATE
- INSERT
- DELETE
- CREATE TABLE
- DROP TABLE
- ALTER TABLE

Statement

- Esempio di utilizzo:

```
stringaSQL =  "DELETE FROM Vigili " +
                    "WHERE Matricola=0012";
int righe = sttm.executeUpdate(stringaSQL);
```

- righe contiene il numero di righe modificate dal comando di aggiornamento.

Prepared Statement

- Nel caso si debba ripetere piu' volte un'istruzione SQL in cui cambiano solo alcuni valori, si puo' preparare uno statement SQL parametrizzato, in cui sia possibile assegnare volta per volta i valori di alcune variabili.

Prepared Statement

2 FASI:

Preparazione:

SELECT Nome FROM Vigili WHERE Matricola = ☐

Utilizzo:

parametro_1 = 1024;

execute(); // SELECT Nome FROM Vigili WHERE Matricola = 1024

parametro_1 = 1027;

execute(); // SELECT Nome FROM Vigili WHERE Matricola = 1027

Prepared Statement (Java)

Preparazione:

```
update = con.prepareStatement(
   "UPDATE Vigili SET Nome = ? " +
   "WHERE Matricola = ?" );
```

Utilizzo:

```
update.setString(1, “Marco" );
update.setInt(2, 1024 );
update.executeUpdate();
```

Prepared Statements

Preparazione:

```
select = con.prepareStatement(
   "SELECT Nome FROM Vigili " +
   "WHERE Matricola = ? ");
```

Utilizzo:

```
select.setInt(1, 1027 );
select.executeQuery();
```

Callable Statement

- Per completezza, bisogna ricordare un terzo tipo di Statement: CallableStatement.
- Un CallableStatement serve per eseguire StoredProcedures.
- Non approfondiremo ulteriormente questo argomento.

SQLException

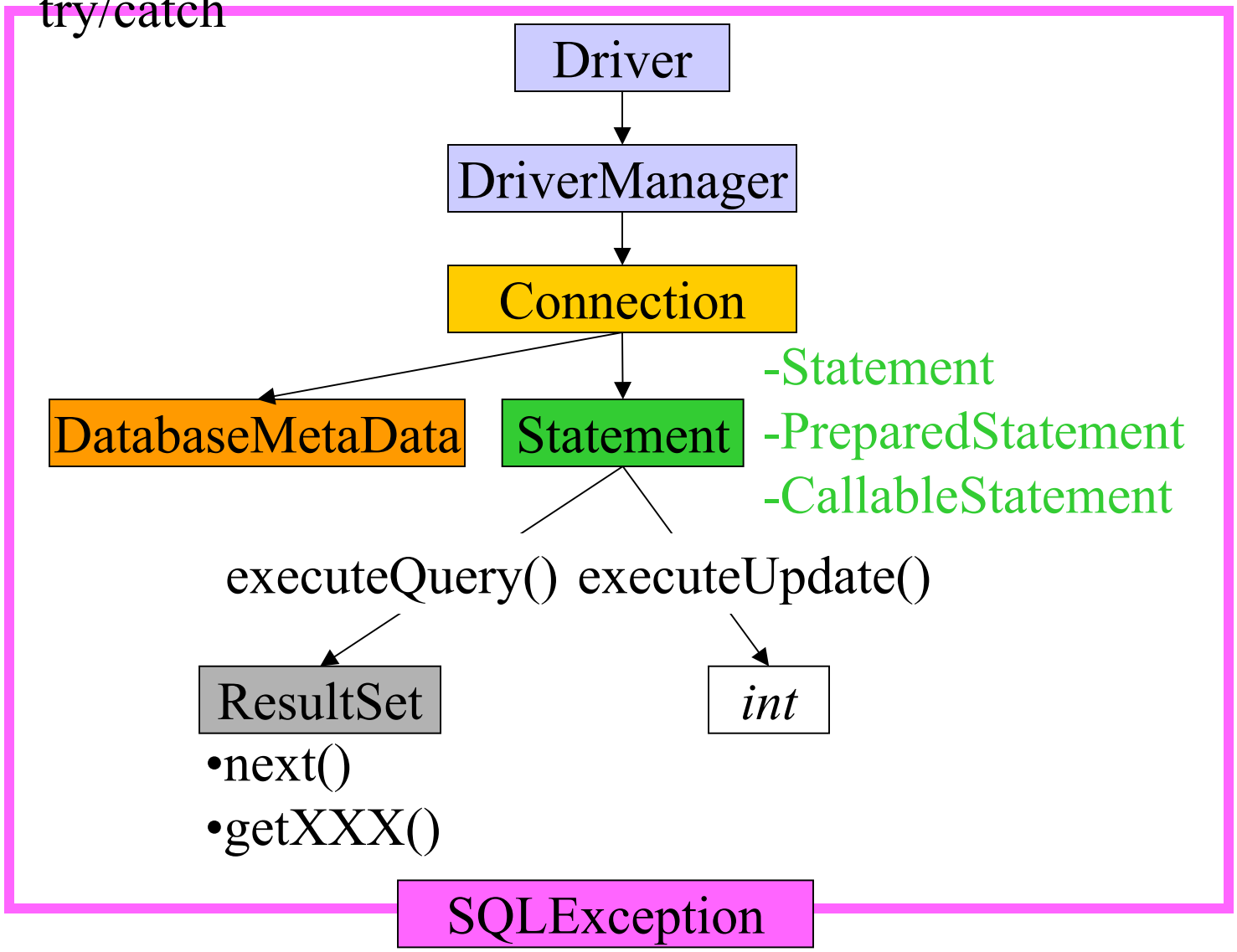
- JDBC fornisce 4 tipi specifici di eccezioni:

  BatchUpdateException
  DataTruncation
  SQLException
  SQLWarning
- SQLException e' la classe da cui derivano le altre, e mette a disposizione due metodi per recuperare i codici di errore:
  - getErrorCode()
  - getSQLState()

Gestione di una SQLException

```
try { // codice JDBC }
catch ( SQLException SQLe) {
    while( SQLe != null) {
        System.out.println(SQLe.getMessage());
        System.out.print("EC: "+SQLe.getErrorCode());
        System.out.println (" SS: "+SQLe.getSQLState());
        SQLe = SQLe.getNextException();
    }
}
```

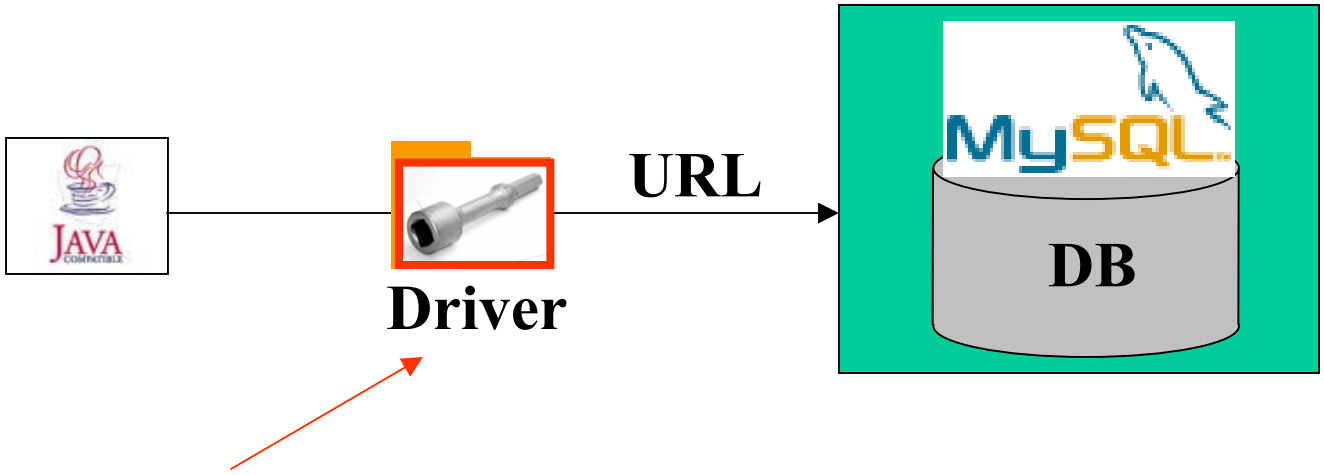
Punto della situazione



DataSource

- Abbiamo visto come utilizzare un Driver e un indirizzo di un database per creare una connessione.
- E' possibile introdurre un livello di astrazione superiore.
- L'utilizzo di un oggetto DataSource, oltre a rendere il codice piu' portabile, puo' aumentare l'efficienza dell'applicazione, utilizzando particolari tipi di connessione.
- Per approfondimenti: http://java.sun.com/j2se/1.4.2/docs/api/ → PooledConnection e XAConnection.

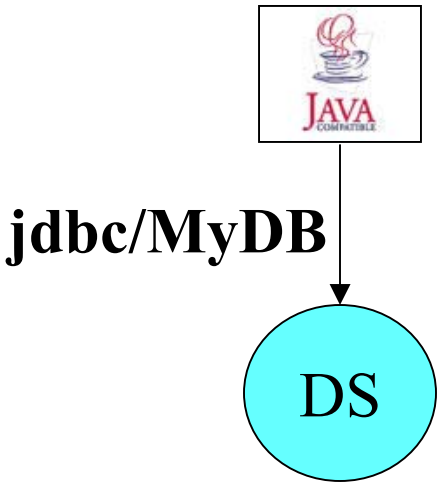
Connessione con Driver e DriverManager



La connessione e' effettuata a basso livello, dovendo specificare la locazione del Driver e l'indirizzo del database.

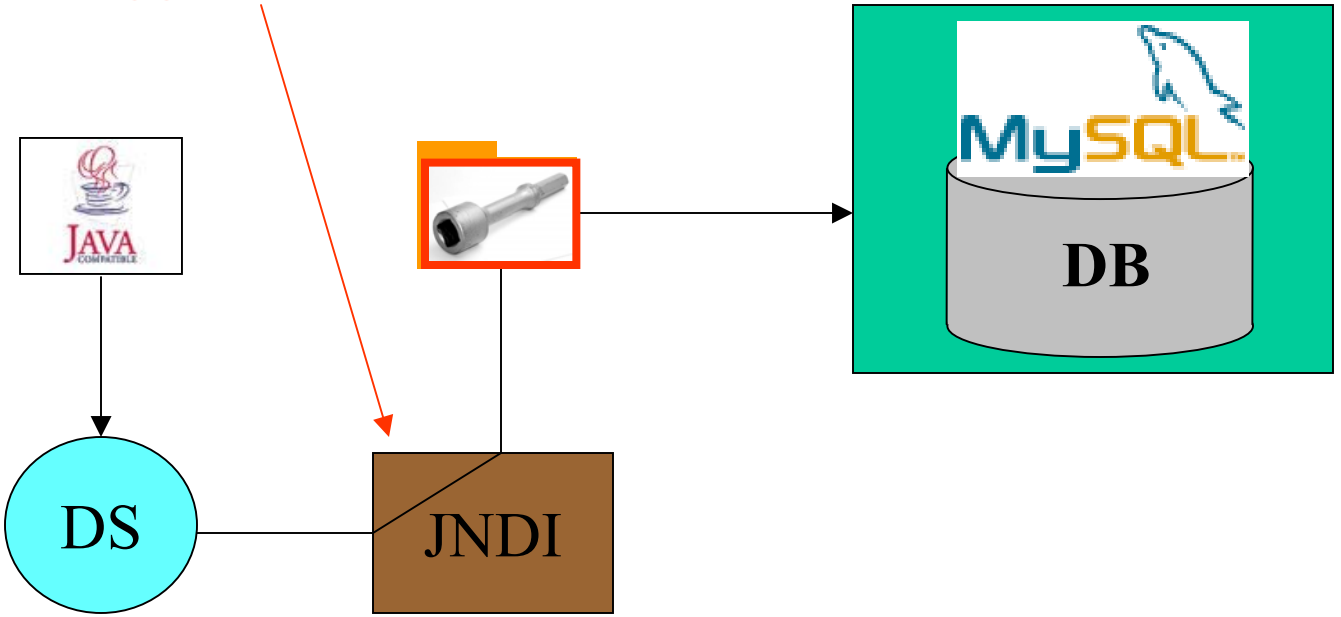
Connessione tramite DataSource

Una DataSource e' un oggetto *logico* indipendente dalla **locazione** e dal **tipo** di base di dati.



Connessione tramite DataSource

Un servizio Java Naming and Directory Interface permette di gestire la sorgente dei dati, provvedendo a passare all'applicazione l'oggetto richiesto.

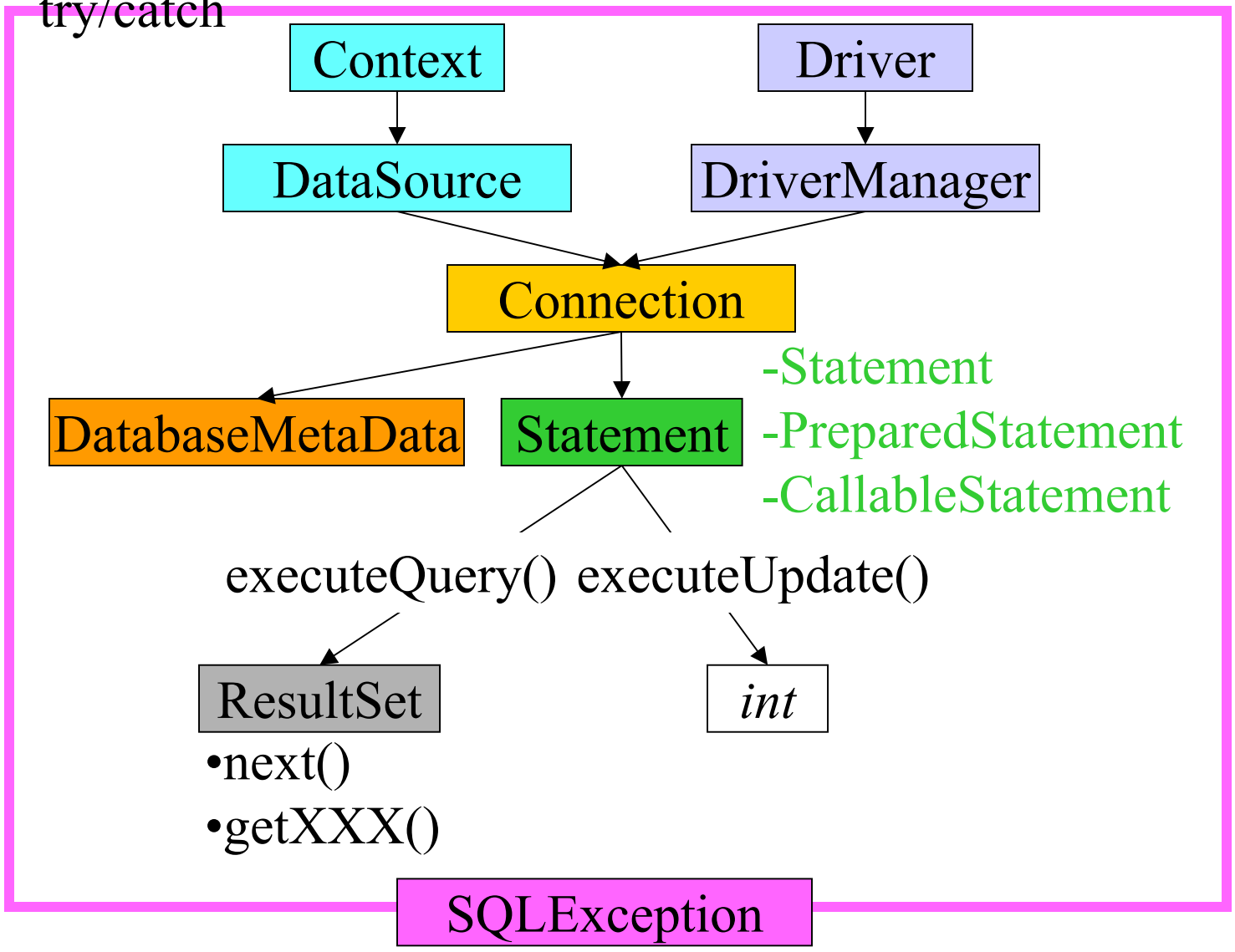


DataSource

```
import javax.naming.*;
import javax.sql.*;

Context ctx = new InitialContext();
DataSource ds =
        (DataSource)ctx.lookup("jdbc/MyDB" );
Connection con =
        ds.getConnection("utente", "password" );
```

Punto della situazione



Concetti avanzati

- Quanto visto finora e' sufficiente per connettersi e interagire con una base di dati, in modo semplice ma efficace.
- JDBC offre pero' funzionalita' avanzate, di cui si potrebbe avere bisogno.
  - Transazioni.
  - Batch updates.
  - Gestione di altri tipi di dato (CLOB, BLOB, Array).
  - Rowsets.
- Riferirsi alla webliografia per approfondimenti.

Webliografia

- Pagina principale su JBDC:

http://java.sun.com/products/jdbc/

- Elenco di risorse didattiche su JDBC:

http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/index.html

- Driver disponibili:

http://servlet.java.sun.com/products/jdbc/drivers