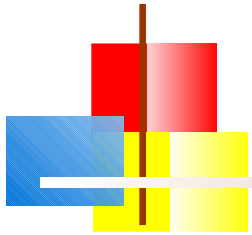
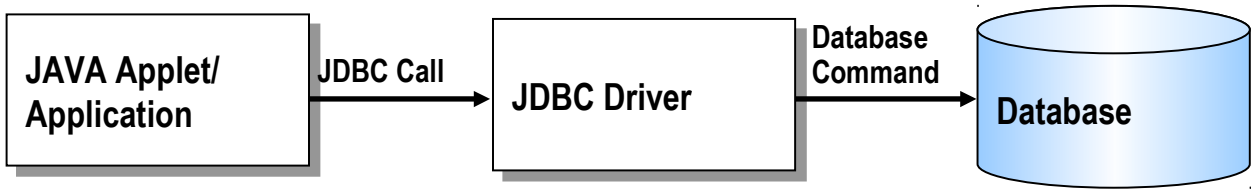
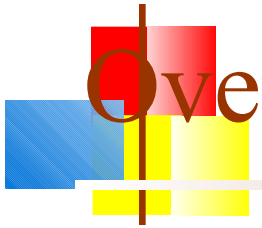


JDBC
(Java Database Connectivity)

Overview (1/2)

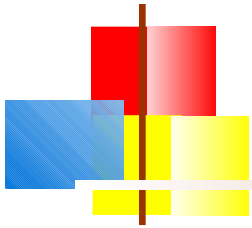
- JDBC
  - JDBC is a standard interface for connecting to relational databases from Java
  - The JDBC Classes and Interfaces are in the ***java.sql*** package
  - JDBC is Java API for executing SQL statements
    - Provides a standard API for tool/database developers
    - Possible to write database applications using a pure Java API
    - Easy to send SQL statements to virtually any relational database
- What does JDBC do?
  - Establish a connection with a database
  - Send SQL statements
  - Process the results





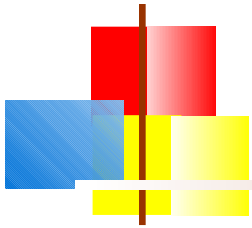
Overview (2/2)

- Reason for JDBC
  - Database vendors (Microsoft Access, Oracle etc.) provide proprietary (non standard) API for **sending** SQL to the server and **receiving** results from it
  - Languages such as C/C++ can make use of these proprietary APIs directly
    - High performance
    - Can make use of non standard features of the database
    - All the database code needs to be rewritten if you change database vendor or product
  - JDBC is a **vendor independent** API for accessing relational data from different database vendors in a consistent way



History of JDBC (1/2)

- JDBC 1.0 released 9/1996.
  - Contains basic functionality to connect to database, query database, process results
  - JDBC classes are part of java.sql package
  - Comes with JDK 1.1
- JDBC 2.0 released 5/1998
  - Comes with JDK 1.2
  - javax.sql contains additional functionality
  - Additional functionality:
    - Scroll in result set or move to specific row
    - Update database tables using Java methods instead of SQL commands
    - Send multiple SQL statements to the database as a batch
    - Use of SQL3 datatypes as column values

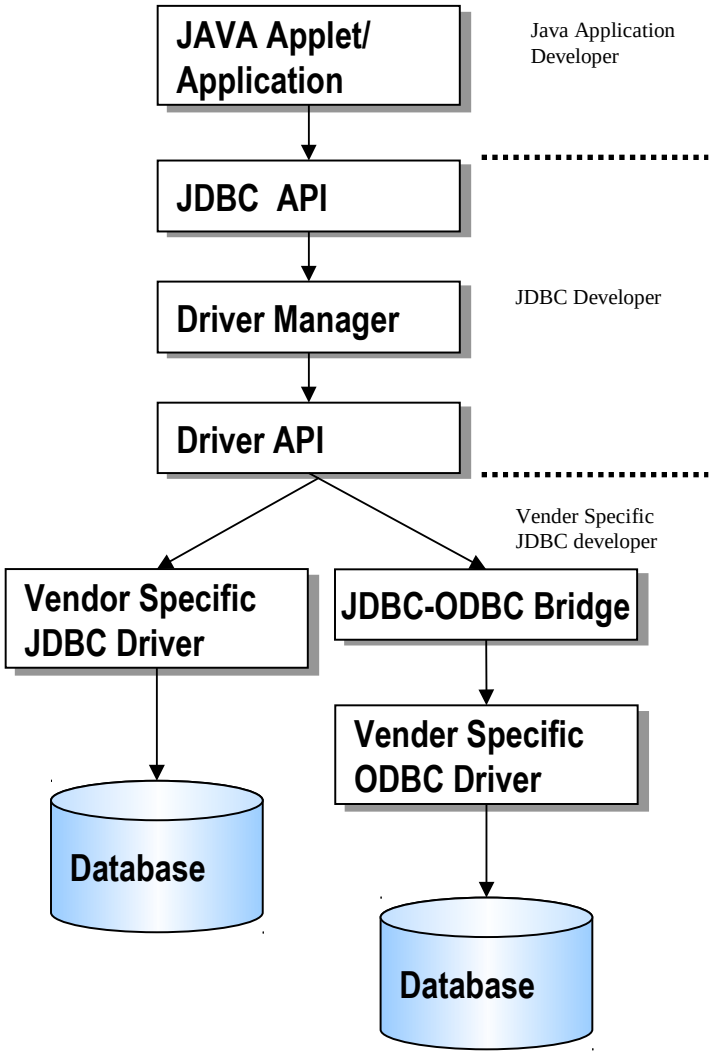


History of JDBC (2/2)

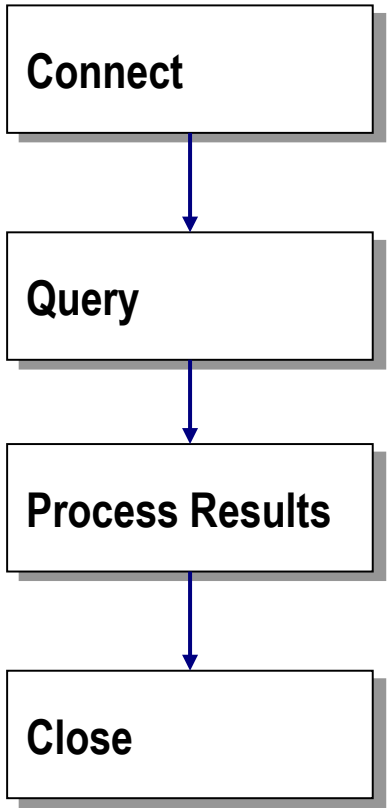
- JDBC 3.0 released 2/2002
  - Comes with Java 2, J2SE 1.4
  - Support for:
    - Connection pooling
    - Multiple result sets
    - Prepared statement pooling
    - Save points in transactions

JDBC Model

- JDBC consists of two parts:
  - JDBC API, a purely Java-based API
  - JDBC driver manager
    - Communicates with vendor-specific drivers
- Connection con = DriverManager.getConnection("jdbc:myDriver:myDatabase", username, password);



JDBC Programming Steps



1) Register the driver
2) Create a connection to the database

1) Create a statement
2) Query the database

1) Get a result set
2) Assign results to Java variables

1) Close the result set
2) Close the statement
3) Close the connection

Skeleton Code

```
Class.forName(DRIVERNAME);

Connection con = DriverManager.getConnection(
                    CONNECTIONURL, DBID, DBPASSWORD);

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM member);

While(rs.next())
{
      Int x = rs.getInt("a");
      String s = rs.getString("b");
      Float f = rs.getFloat("c");
}

rs.close();
stmt.close();
con.close();
```

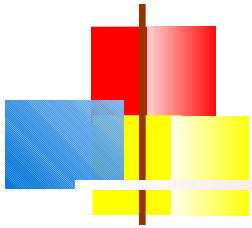
Loading a JDBC driver

Connecting to a database

Executing SQL

Processing the result set

Closing the connections



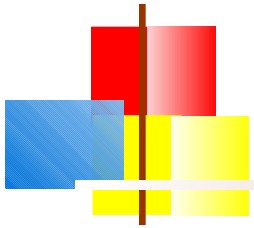
Step 1 : Loading a JDBC Driver

- A JDBC driver is needed to connect to a database
- Loading a driver requires the class name of the driver.
  Ex) JDBC-ODBC: sun.jdbc.odbc.JdbcOdbcDriver
  Oracle driver: oracle.jdbc.driver.OracleDriver
  MySQL: com.mysql.jdbc.Driver

- Loaing the driver class

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

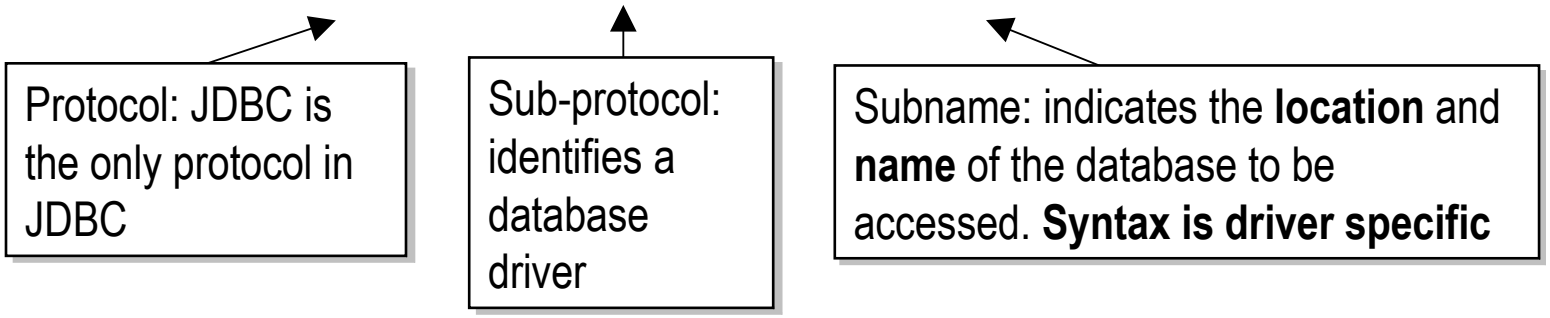
- It is possible to load several drivers.
- The class *DriverManager* manages the loaded driver(s)



Step 2 : Connecting to a Database (1/2)

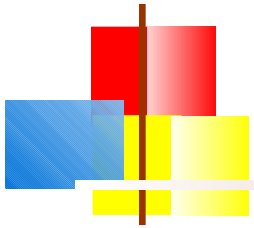
- JDBC URL for a database
  - Identifies the database to be connected
  - Consists of three-part:

jdbc:<subprotocol>:<subname>



Ex) jdbc:mysql://oopsla.snu.ac.kr/mydb

The syntax for the name of the database is a little messy and is unfortunately vendor specific



Step 2 : Connecting to a Database (2/2)

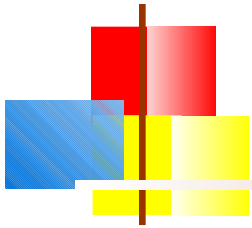
- The *DriverManager* allows you to connect to a database using the specified JDBC driver, database location, database name, username and password.
- It returns a *Connection* object which can then be used to communicate with the database.

```
Connection connection =
DriverManager.getConnection("jdbc:mysql://oopsla.snu.ac.kr/mydb", "userid","password");
```

JDBC URL
Vendor of database, Location of database server and name of database

Username

Password

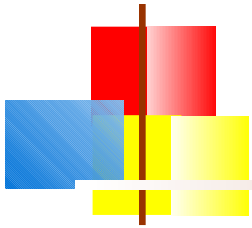


Step 3 : Executing SQL (1/2)

- Statement object
  - Can be obtained from a Connection object

Statement statement = connection.createStatement();

  - Sends SQL to the database to be executed
- Statement has three methods to execute a SQL statement:
  - **executeQuery()** for QUERY statements
    - Returns a ResultSet which contains the query results
  - **executeUpdate()** for INSERT, UPDATE, DELETE statements
    - Returns an integer, the number of affected rows from the SQL
  - **execute()** for either type of statement



Step 3 : Executing SQL (2/2)

- Execute a select statement

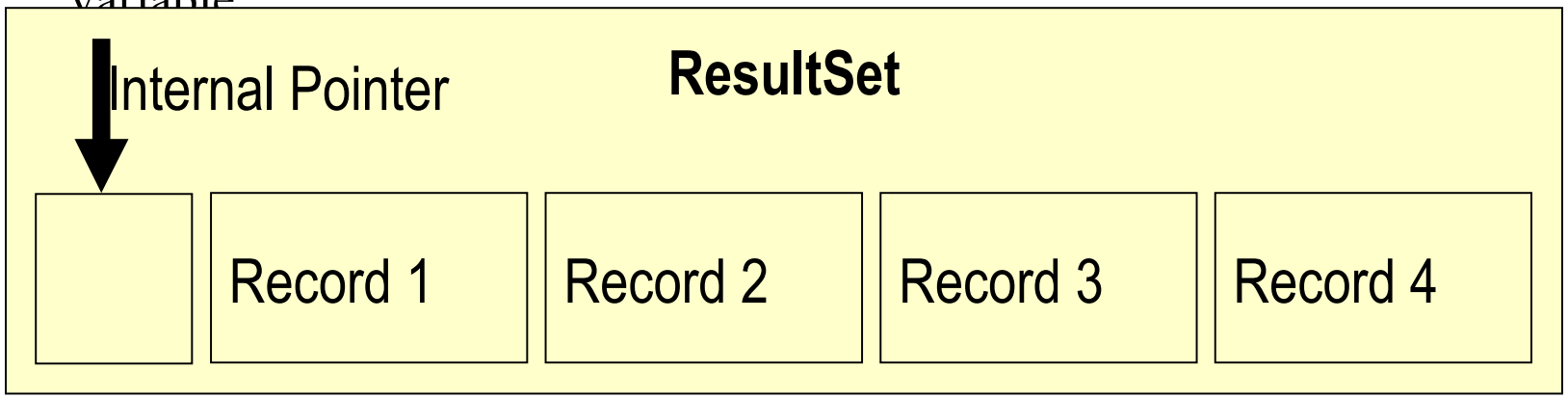
```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery
  ("select RENTAL_ID, STATUS from ACME_RENTALS");
```

- Execute a delete statement

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate
                    ("delete from ACME_RENTAL_ITEMS
                      where rental_id = 1011");
```

Step 4 : Processing the Results (1/2)

- JDBC returns the results of a query in a ResultSet object
  - ResultSet object contains all of the rows which satisfied the conditions in an SQL statement
- A ResultSet object maintains a cursor pointing to its current row of data
  - Use next() to step through the result set row by row
    - next() returns TRUE if there are still remaining records
  - getString(), getInt(), and getXXX() assign each value to a Java variable



The internal pointer starts one before the first record

Step 4 : Processing the Results (2/2)

- Example

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT ID, name, score FROM table1");

While (rs.next()){
        int id = rs.getInt("ID");
        String name = rs.getString("name");
        float score = rs.getFloat("score");
        System.out.println("ID=" + id + " " + name + " " + score);}
```

NOTE

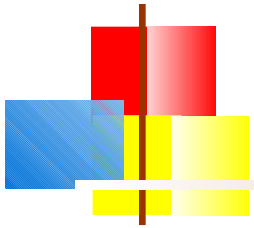
You must step the cursor to the first record before read the results
This code will not skip the first record

ID	name	score
1	James	90.5
2	Smith	45.7
3	Donald	80.2

Table1

Output

ID=1 James 90.5
ID=2 Smith 45.7
ID=3 Donald 80.2



Step 5 : Closing Database Connection

- It is a good idea to close the Statement and Connection objects when you have finished with them
- Close the ResultSet object

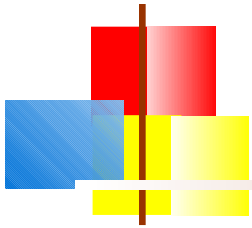
```
rs.close();
```

- Close the Statement object

```
stmt.close();
```

- Close the connection

```
connection.close();
```



The PreparedStatement Object

- A PreparedStatement object holds precompiled SQL statements
- Use this object for statements you want to execute more than once
- A PreparedStatement can contain variables (?) that you supply each time you execute the statement

```
// Create the prepared statement
PreparedStatement pstmt = con.prepareStatement(“
            UPDATE table1 SET status = ? WHERE id =?”)
// Supply values for the variables
pstmt.setString (1, “out”);
pstmt.setInt(2, id);
// Execute the statement
pstmt.executeUpdate();
```