CST8221 – Java Application Programming

Unit 11 – Java Database Programming (JDBC)

In 1996 Sun introduced the first version of JDBC API. This API lets programmers connect to a database and then query or update it, using the Structured Query Language (SQL) (SQL is an industry standard for relational database access). Since the introduction JDBC has became one of the most frequently used API in the Java library. JDBC has been updated several times. The most current version is JDBC 4.2 which is the version included in Java SE 1.8.

From the very start, the developers of the Java technology at Sun were aware of the potential that Java presented for working with databases. At the time there were too many databases, so that the idea to use "pure" Java for connection to any database was impossible to realize. It was decided that Sun will provide a pure Java API for SQL access along with a Driver Manager which will allow third-party drivers to connect to specific databases. As a result, two APIs were created. Application programmers use the JDBC API (java.sql) and the database vendors and database tool providers use JDBC Driver API.

The java database connectivity organization follows the very successful models ODBC (Open Database Connectivity) introduced by Microsoft. ODBC provides a C programming language interface for database access. Both JDBC and ODBC are based on the same fundamental idea: Programs written according to the API talk to the driver manager, which, in turn, uses a vendor provided driver to talk to the actual database. From programmer point of view that means that the JDBC API is all that the most programmers will ever have to deal with – see the figure on the next page.

Note: You can find a list of currently available JDBC drivers on the Internet. One place to visit is:

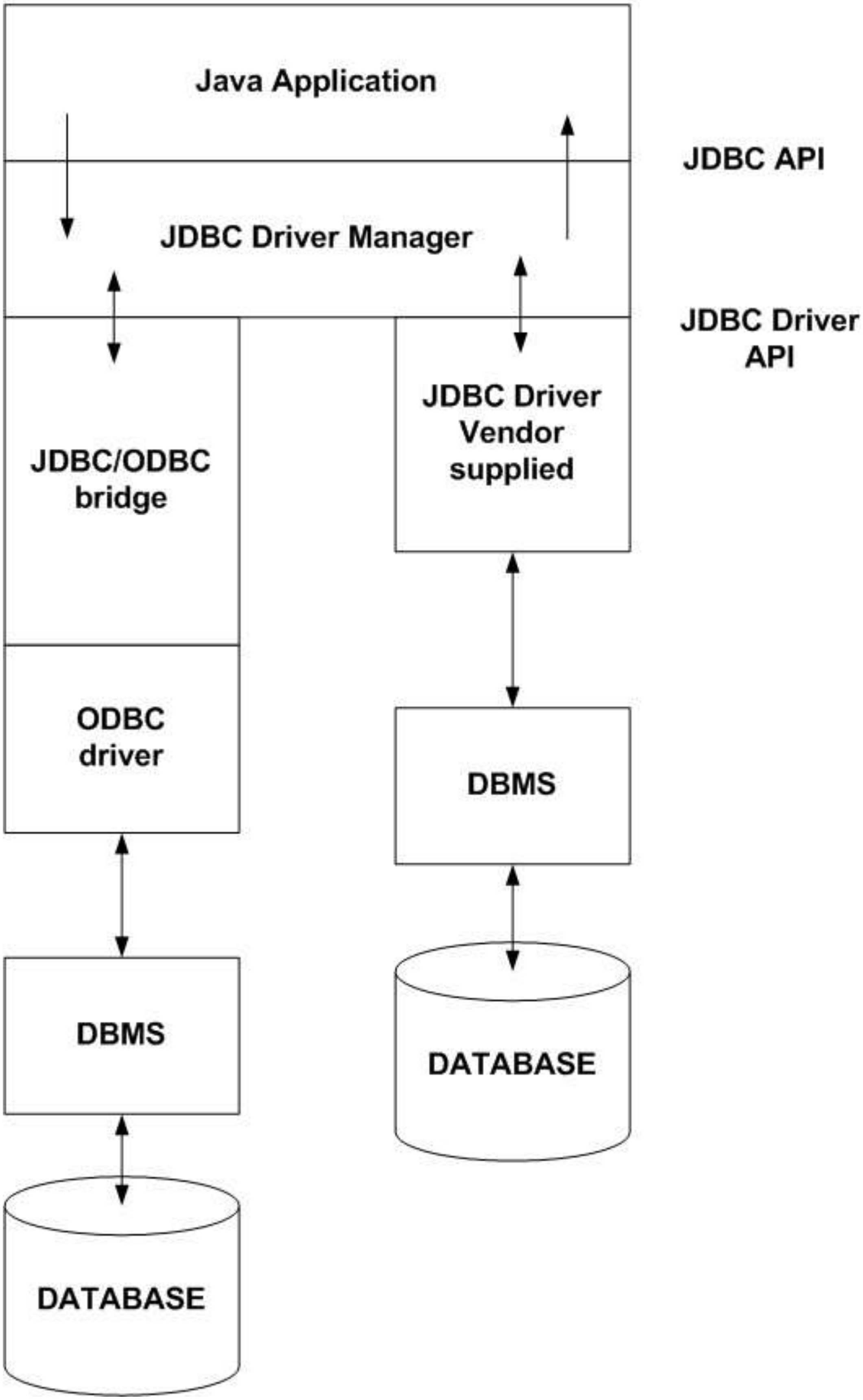
http://www.dbvis.com/doc/database-drivers/

The JDBC specification classifies drivers into following four types:

1. A type 1 driver translates JDBC to ODBC and relies on an ODBC driver to communicate with the database. It is called JDBC/ODBC bridge. You should avoid using the bridge if it is possible.
2. A type 2 driver is written partly in Java and partly in native code. The native code talks to the database. When you use such a driver you must install some platform-specific code onto the client in addition to the Java library.
3. A type 3 driver is a pure Java client library that uses database independent protocol to communicate database requests to a server component, which then translates the requests to a server component, which then translates the requests into a database specific protocol. In other words, a pure-java driver talks with the server-side middleware that then talks to database. This significantly simplifies the deployment because the platform specific code is located only on the server.
4. A type 4 driver is a pure Java API that translates JDBC requests directly to a database-specific protocol.

Most of the existing drivers are either type 3 or 4.

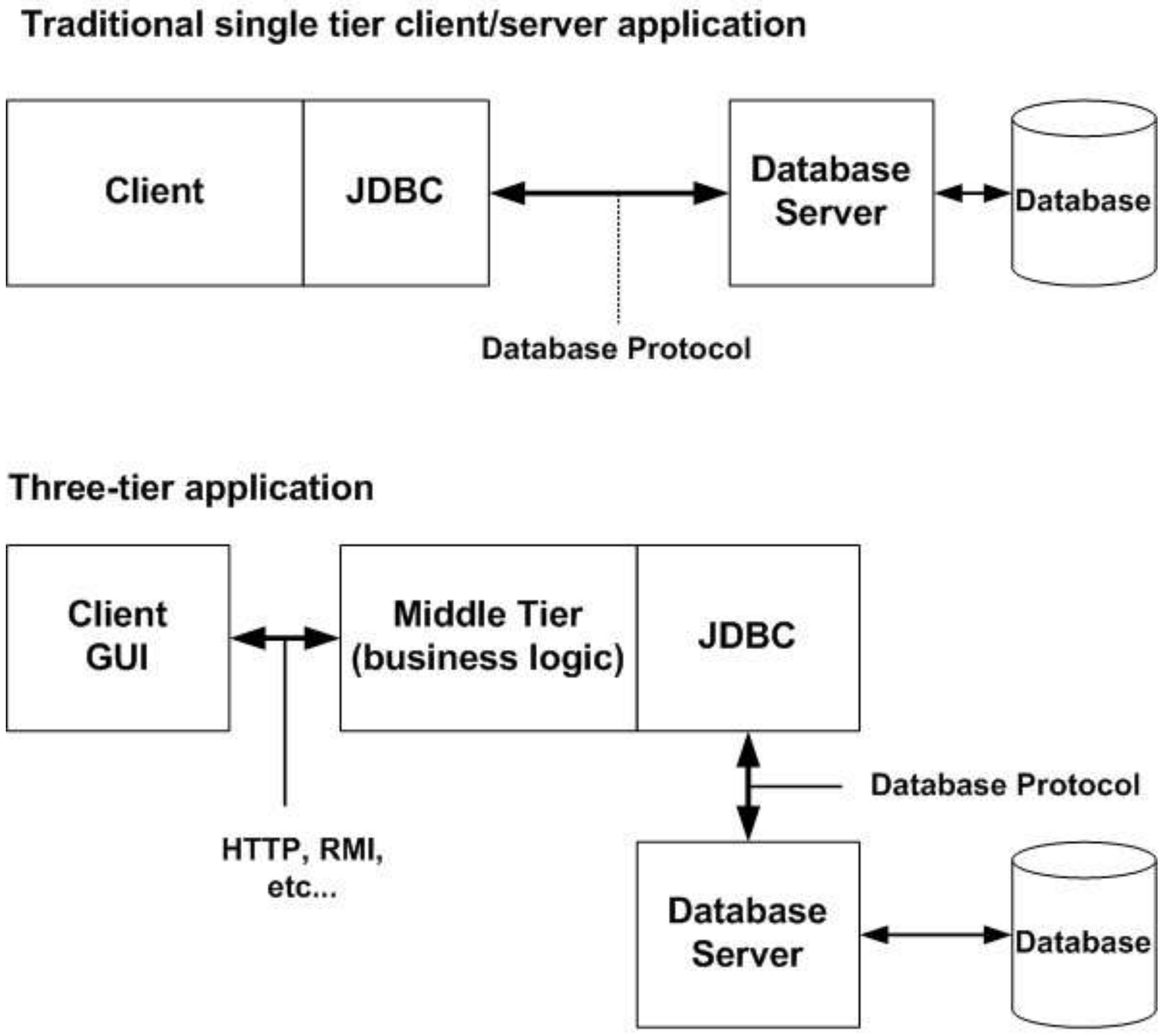
Java Application-to-Database Connection



In summary, the ultimate goal of JDBC is to make possible the following:

- The programmers can write applications in the Java programming language to access any database, using standard SQL statements while still following java language conventions.
- Database vendors and database tool vendors can supply the low-level drivers, and thus they can optimize their drivers for their specific products.

The typical uses of JDBC employ two broad models illustrated in the figure below.



Typical uses of JDBC

The traditional client/server model has a rich GUI on the client and a database on the server. In this model the JDBC driver is deployed on the client.

The other model is so called three-tier model or the more advanced n-tier model. In the three-tier model, the client does not make database calls. Instead, it calls on the middleware layer on the server that in turn communicates with the database. The multi -tier model has many advantages. It separates the visual presentations (the view) from the business logic (the model) and the row data in the database. Therefore, it becomes possible to access the same data and the same business rules from multiple clients, such as a Java application, or applet, or a web page form. Communication between the client and the middle tier can occur through HTTP (with a web browser as a client), or another mechanism sash as remote method invocation (RMI).
There are many variations of this model. In particular, the Java Enterprise edition defines a structure for application servers that manage code modules called Enterprise JavaBeans, and provides valuable services such as load balancing, request caching, security, and object-relational mapping. In that architecture JDBC still plays an important role for issuing complex database queries.

To use JDBC, you need a database for which a JDBC driver is available. There are many excellent choices, such as IBM DB2, Microsoft SQL server, MySQL, Oracle, PostgreSQL, Apache Derby, or Java DB which is provided with the standard Java SDK (Java DB is a SUN version of Apache Derby).

You need a number of item before you can write a database program.

Database URLs

When connecting to a database, you must use various database specific parameters such as host names, port numbers, and database names. JDBC uses a syntax similar to that of ordinary URLs to describe data sources. Here is one example:

```
jdbc:derby://localhost:1527/JAPDB;create=true
```

This JDBC URL specifies a Derby database named JAPDB.

The general syntax is

```
jdbc:subprotocol:other parameters
```

where the *subprotocol* selects the specific driver for connecting to the database.

Driver jar files

You need the JAR file in which the driver for your database is located. For example, if you want to use Derby, you need the file ***derbyclient.jar***.
You must include the driver classpath when running a program that accesses the database. You do not need the jar file for compiling your programs. You can set the CLASSPATH environment variable, or simply use the command:

```
java –cp .;driver_jar_file_name.jar ProgramName
```

Starting the Database

The database server needs to be started before you can connect to it. The details depend on your database. For example, to start Derby (JavaDB) you should type the command:

```
java  -jar “c:\...\derbyrun.jar server start
```

When you are done using the database, stop the server with the command

```
java  -jar “c:\...\derbyrun.jar server stop
```

If you use different database, you need to consult the documentation to find out how to start and stop your database server, and how to connect to it and issue SQL commands.

Registering the Driver Class

Some JDBC JAR files (such as Derby driver that is included with JAVA SE 1.7) automatically register the driver class when a connection to the database is established. In other cases you must register the driver class manually before you can dun your Java programs. A JAR file can automatically register the driver class if it contains a file META-INF/services/java.sql.driver.

if your driver JAR does not support automatic registration, you need to find out the name of the JDBC driver class used by your database vendor. There are two methods to register the driver with the **DriverManager**. One way is to load the driver class in your Java program using *Class.forName(“driver_name”);* This statement will couse the driver class to be loaded, thereby executing a static initialized that register the driver.

Alternatively, you can set the *jdbc.drivers* property. . You can specify the property with a command line argument, such as

```
java –Djdbc.drivers=driver_name ProgramName
```

Connecting to the Database

In your Java program, you open a database connection with code that is similar to the following example:

```
String dbURL = “jdbc:…”;
String username = “dbuser”;
String password = “bigsecret”;
Connection conn = DriverManager.getConnection(url,username,password);
```

The *getConnection()* method returns a *Connection* object. You can use the connection object to execute SQL statements and retrieve the result from the statements.

Code Example: TestDB.java (Assignment/Lab14 code).