Java Reflection and JDBC

Introduction

Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.

- **java.lang.reflect** package.
- It gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
- Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

Meta-data Information

Reflection can be used to get information about

1. **Class** The getClass() method is used to get the name of the class to which an object belongs.
2. **Constructors** The getConstructors() method is used to get the public constructors of the class to which an object belongs.
3. **Methods** The getMethods() method is used to get the public methods of the class to which an objects belongs.

java.lang.Class

The java.lang.Class class performs mainly two tasks:

- provides methods to get the metadata of a class at run time.
- provides methods to examine and change the run time behavior of a class.

There are 3 ways to get the instance of Class class. They are as follows:

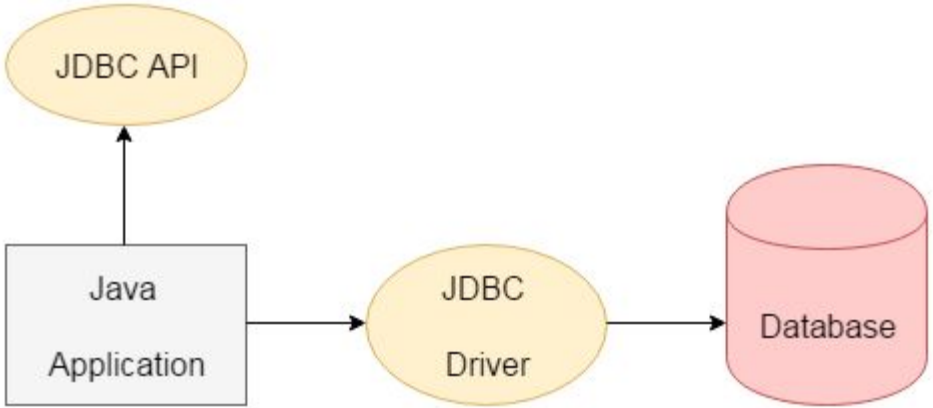
- forName() method of Class class
- getClass() method of Object class
- the .class syntax

Example

```
1.  class Simple{}
2.
3.  class Test{
4.   public static void main(String args[]){
5.    Class c=Class.forName("Simple");
6.    System.out.println(c.getName());
7.   }
8.  }
```

JDBC - Java Database Connectivity

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



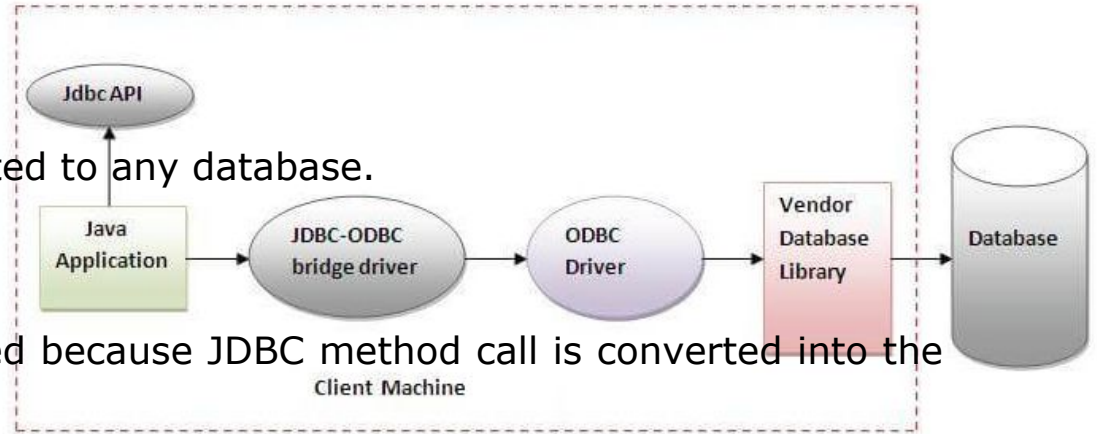
JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

JDBC-ODBC Bridge Driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver



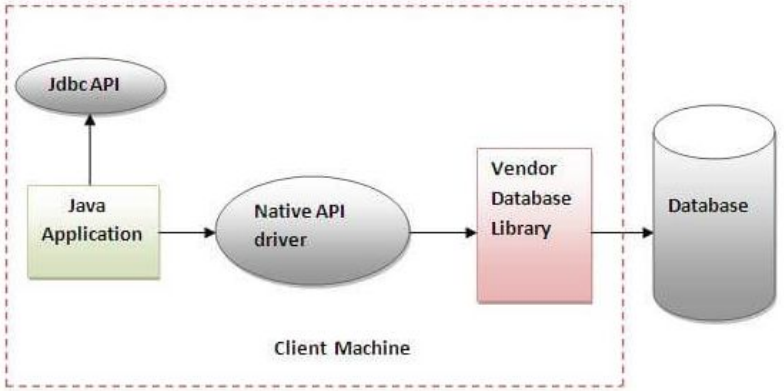
Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

Native API Driver



The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java

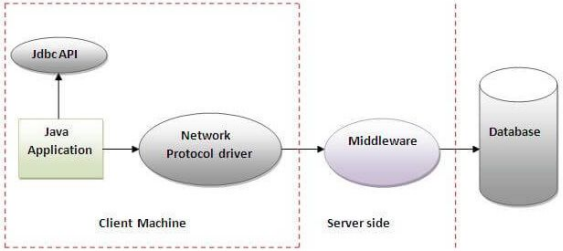
Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

Network Protocol Driver



The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

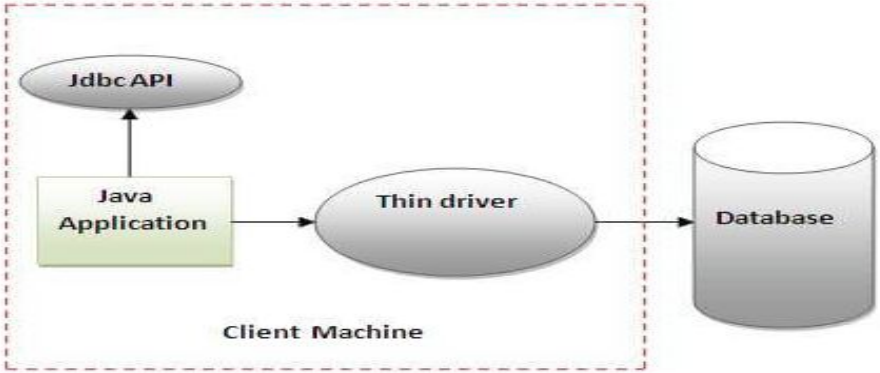
Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier

Thin Driver



The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

Connects to the Database

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection