Introduction

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

The java.sql package contains classes and interfaces for JDBC API.
A list of popular interfaces of JDBC API are given below: -

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular classes of JDBC API are given below:

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

Connect to the database.
Execute queries and update statements to the database.
Retrieve the result received from the database.

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

JDBC Driver

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

- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)

Java Database Connectivity

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class.
- Create connection.
- Create statement.
- Execute queries.
- Close connection.

1) Register the driver class

The forName() method of Class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

Example to register the OracleDriver class.

```
// Load the driver
Class.forName("com.mysql.cj.jdbc.Driver");
```

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Example: -

```
    String url ="jdbc:mysql://localhost:3306/jdbc";
    String username="root";
    String password="12006011";
    Connection con=DriverManager.getConnection(url,username,password);
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Example: -
Statement stmt=con.createStatement();

This is the example for creating the connection.

```
Connection con = DriverManager.getConnection(url,username,password);
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Example

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

This is for check the connection is created or not.

```
if(con.isClosed()) {
                System.out.println("Connection is closed");
            } else {
                System.out.println("Connection created...");
            }
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Example
con.close();

The steps for working database (creating connection).

```
1) load the driver:
method 1st:
-------------
Class.forName("com.mysql.jdbc.Driver")

method 2nd
----------
```

```
DriverMAnager.registerDriver(new com.mysql.jdbc.Driver());


2)Create a Connection:
 Connection
con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/dbname","root","r
oot");


3)Create query, Statement , PreparedStatement , CallableStatement
eg
 String q="select * from students";

 Statement stmt=con.createStatement();

 ResultSet set=stmt.executeQuery(q);

4)Process the data :

 while(set.next())
 {
   int id=set.getInt("studentID");
   String name=set.getString("studentName");
   System.out.println(id);
   System.out.println(name);

 }

5) Close the connection:
 con.close();
```

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySQL as the database. So, we need to know following informations for the MySQL database:

- **Driver class:** The driver class for the MySQL database is com.mysql.jdbc.Driver.
- **Connection URL:** The connection URL for the MySQL database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
- **Username:** The default username for the mysql database is root.

- **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database
In this example, sonoo is the database name, root is the username and password both.
Play

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:
Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:
There are two ways to set the classpath:

- temporary
- permanent

How to set the temporary classpath
open command prompt and write:
C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

How to set the permanent classpath.

Go to environment variable then click on new tab. In variable name write classpath and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

DriverManager class

The DriverManager class is the component of JDBC API and also a member of the java.sql package. The DriverManager class acts as an interface between users and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains all the appropriate methods to register and deregister the database driver class and to create a connection between a Java application and the database. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver(). Note that before interacting with a Database, it is a mandatory process to register the driver; otherwise, an exception is thrown.

Methods of the DriverManager Class

Method	Description
1) public static synchronized void registerDriver(Driver driver):	is used to register the given driver with DriverManager. No action is performed by the method when the given driver is already registered.
2) public static synchronized void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager. If the given driver has been removed from the list, then no action is performed by the method.
3) public static Connection getConnection(String url) throws SQLException:	is used to establish the connection with the specified url. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager.
4) public static Connection getConnection(String url,String userName,String password) throws SQLException:	is used to establish the connection with the specified url, username, and password. The SQLException is thrown when the corresponding Driver class of the given database is not registered with the DriverManager.
5) public static Driver getDriver(String url)	Those drivers that understand the mentioned URL (present in the parameter of the method) are returned by this method provided those drivers are mentioned in the list of registered drivers.
6) pubic static int getLoginTimeout()	The duration of time a driver is allowed to wait in order to establish a connection with the database is returned by this method.

7) pubic static void setLoginTimeout(int sec)	The method provides the time in seconds. sec mentioned in the parameter is the maximum time that a driver is allowed to wait in order to establish a connection with the database. If 0 is passed in the parameter of this method, the driver will have to wait infinitely while trying to establish the connection with the database.
8) public static Connection getConnection(String URL, Properties prop) throws SQLException	A connection object is returned by this method after creating a connection to the database present at the mentioned URL, which is the first parameter of this method. The second parameter, which is "prop", fetches the authentication details of the database (username and password.). Similar to the other variation of the getConnection() method, this method also throws the SQLException, when the corresponding Driver class of the given database is not registered with the DriverManager.

Connection interface

A Connection is a session between a Java application and a database. It helps to establish a connection with the database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData, i.e., an object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback(), setAutoCommit(), setTransactionIsolation(), etc.

By default, connection commits the changes after executing queries.
Commonly used methods of Connection interface:
1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
Pause

3) public void setAutoCommit(boolean status): is used to set the commit status. By default, it is true.

4) public void commit(): saves the changes made since the previous commit/rollback is permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Connection Interface Fields

There are some common Connection interface constant fields that are present in the Connect interface. These fields specify the isolation level of a transaction.

TRANSACTION_NONE: No transaction is supported, and it is indicated by this constant.

TRANSACTION_READ_COMMITTED: It is a constant which shows that the dirty reads are not allowed. However, phantom reads and non-repeatable reads can occur.

TRANSACTION_READ_UNCOMMITTED: It is a constant which shows that dirty reads, non-repeatable reads, and phantom reads can occur.

TRANSACTION_REPEATABLE_READ: It is a constant which shows that the non-repeatable reads and dirty reads are not allowed. However, phantom reads and can occur.

TRANSACTION_SERIALIZABLE: It is a constant which shows that the non-repeatable reads, dirty reads as well as the phantom reads are not allowed.

Statement interface

The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:
The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.
2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.
3) public boolean execute(String sql): is used to execute queries that may return multiple results.
4) public int[] executeBatch(): is used to execute batch of commands.