

Establishing JDBC Connection in Java

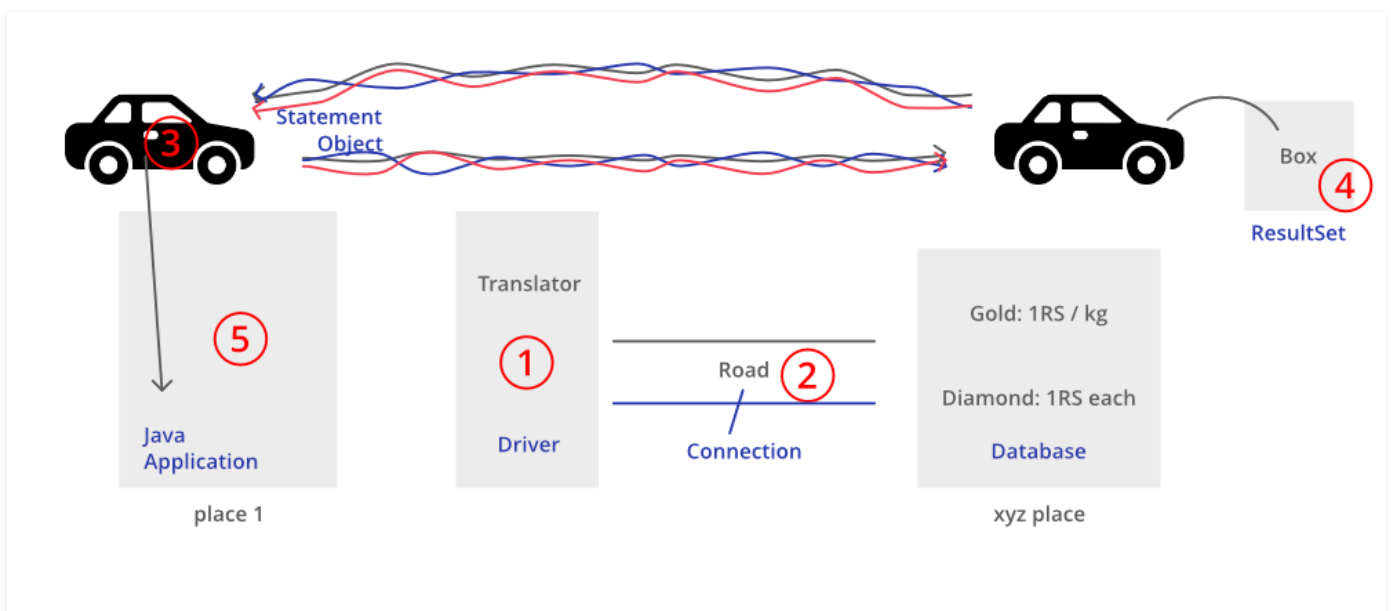
Difficulty Level : Easy Last Updated : 02 Nov, 2021

Before establishing a connection between the front end i.e your Java Program and the back end i.e the database we should learn what precisely a JDBC is and why it came to existence. Now let us discuss what exactly JDBC stands for and will ease out with the help of real-life illustration to get it working.

What is JDBC?

JDBC is an acronym for Java Database Connectivity. It's an advancement for ODBC (Open Database Connectivity). JDBC is a standard API specification developed in order to move data from frontend to backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

Illustration: Working of JDBC co-relating with real-time



Why JDBC Came into Existence?

As previously told JDBC is an advancement for ODBC, ODBC being platform-dependent had a lot of drawbacks. ODBC API was written in C, C++, Python, Core Java and as we know above languages (except Java and some part of Python)are platform dependent. Therefore to remove dependence, JDBC was developed by a database vendor which consisted of classes and interfaces written in Java.

Steps For Connectivity Between Java Program and Database

1. Import the database
2. Load the drivers using the *forName() method*
3. Register the drivers *using DriverManager*
4. Establish a connection *using the Connection class object*
5. Create a statement
6. Execute the query
7. Close the connections

Let us discuss these steps in brief before implementing by writing suitable code to illustrate connectivity steps for JDBC/

Step 1: Import the database

Step 2: Loading the drivers

In order to begin with, you first need to load the driver or register it before using it in the program. Registration is to be done once in your program. You can register a driver in one of two ways mentioned below as follows:

2-A Class.forName()

Here we load the driver's class file into memory at the runtime. No need of using new or create objects. The following example uses Class.forName() to load the Oracle driver as shown below as follows:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2-B DriverManager.registerDriver()

DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time. The following example uses DriverManager.registerDriver() to register the Oracle driver as shown below:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())
```

Step 4: Establish a connection *using the Connection class object*

After loading the driver, establish connections via as shown below as follows:

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

- **user:** Username from which your SQL command prompt can be accessed.
- **password:** password from which the SQL command prompt can be accessed.
- **con:** It is a reference to the Connection interface.
- **Url :** Uniform Resource Locator which is created as shown below:

```
String url = " jdbc:oracle:thin:@localhost:1521:xe"
```

Where oracle is the database used, thin is the driver used, @localhost is the IP Address where a database is stored, 1521 is the port number and xe is the service provider. All 3 parameters above are of String type and are to be declared by the programmer before calling the function. Use of this can be referred from the final code.

Step 5: Create a statement

Once a connection is established you can interact with the database. The JDBCStatement, CallableStatement, and PreparedStatement interfaces define the methods that enable you to send SQL commands and receive data from your database.

Use of JDBC Statement is as follows:

```
Statement st = con.createStatement();
```

***Note:** Here, con is a reference to Connection interface used in previous step .*

Step 6: Execute the query

Now comes the most important part i.e executing the query. The query here is an SQL Query. Now we know we can have multiple types of queries. Some of them are as follows:

- The query for updating/inserting table in a database.
- The query for retrieving data.

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

The executeUpdate(sql query) method of the Statement interface is used to execute queries of updating/inserting.

Pseudo Code:

```
int m = st.executeUpdate(sql);
if (m==1)
    System.out.println("inserted successfully : "+sql);
else
    System.out.println("insertion failed");
```

Here sql is SQL query of the type String

Step 7: Closing the connections

So finally we have sent the data to the specified location and now we are on the verge of completing of our task. By closing the connection, objects of Statement and ResultSet will be closed automatically. The close() method of the Connection interface is used to close the connection. It is as shown below as follows:

```
con.close();
```

Example:

```
// Java Program to Establish Connection in JDBC

// Importing database
import java.sql.*;
// Importing required classes
import java.util.*;

// Main class
class Main {

    // Main driver method
    public static void main(String a[])
    {

        // Creating the connection using Oracle DB
        // Note: url syntax is standard, so do grasp
        String url = "jdbc:oracle:thin:@localhost:1521:xe";

        // Username and password to access DB
```

```
// Custom initialization
String user = "system";
String pass = "12345";

// Entering the data
Scanner k = new Scanner(System.in);

System.out.println("enter name");
String name = k.next();

System.out.println("enter roll no");
int roll = k.nextInt();

System.out.println("enter class");
String cls = k.next();

// Inserting data using SQL query
String sql = "insert into student1 values('" + name
            + "'," + roll + "," + cls + "')";

// Connection class object
Connection con = null;

// Try block to check for exceptions
try {

    // Registering drivers
    DriverManager.registerDriver(
        new oracle.jdbc.OracleDriver());

    // Reference to connection interface
    con = DriverManager.getConnection(url, user,
                                    pass);

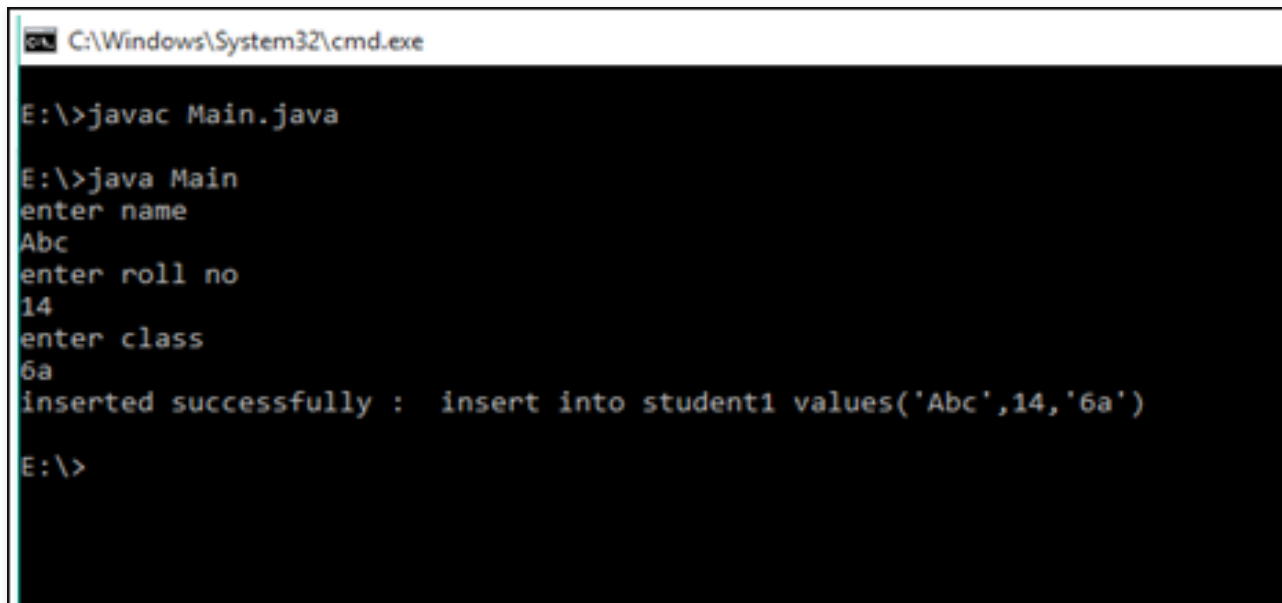
    // Creating a statement
    Statement st = con.createStatement();

    // Executing query
    int m = st.executeUpdate(sql);
    if (m == 1)
        System.out.println(
            "inserted successfully : " + sql);
    else
        System.out.println("insertion failed");

    // Closing the connections
    con.close();
}

// Catch block to handle exceptions
catch (Exception ex) {
    // Display message when exceptions occurs
    System.err.println(ex);
}
}
```

Output:



```
C:\Windows\System32\cmd.exe

E:\>javac Main.java

E:\>java Main
enter name
Abc
enter roll no
14
enter class
6a
inserted successfully : insert into student1 values('Abc',14,'6a')

E:\>
```

This article is contributed by **Shreya Gupta**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.