Connecting to MySQL Using JDBC Driver

In this tutorial, you will learn how to connect to MySQL database using JDBC Connection object.

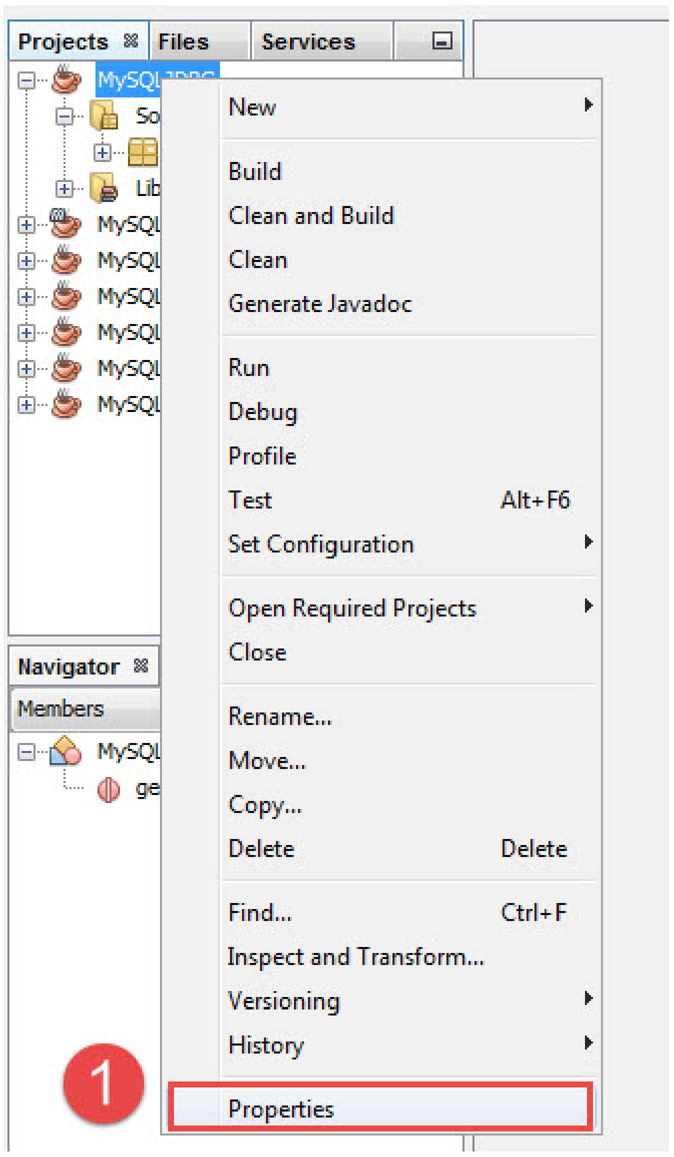
To connect to MySQL database from a Java program, you need to do the following steps:

1. Load the MySQL Connector/J into your program.
2. Create a new Connection object from the `DriverManager`class. Then you can use this Connection object to execute queries.

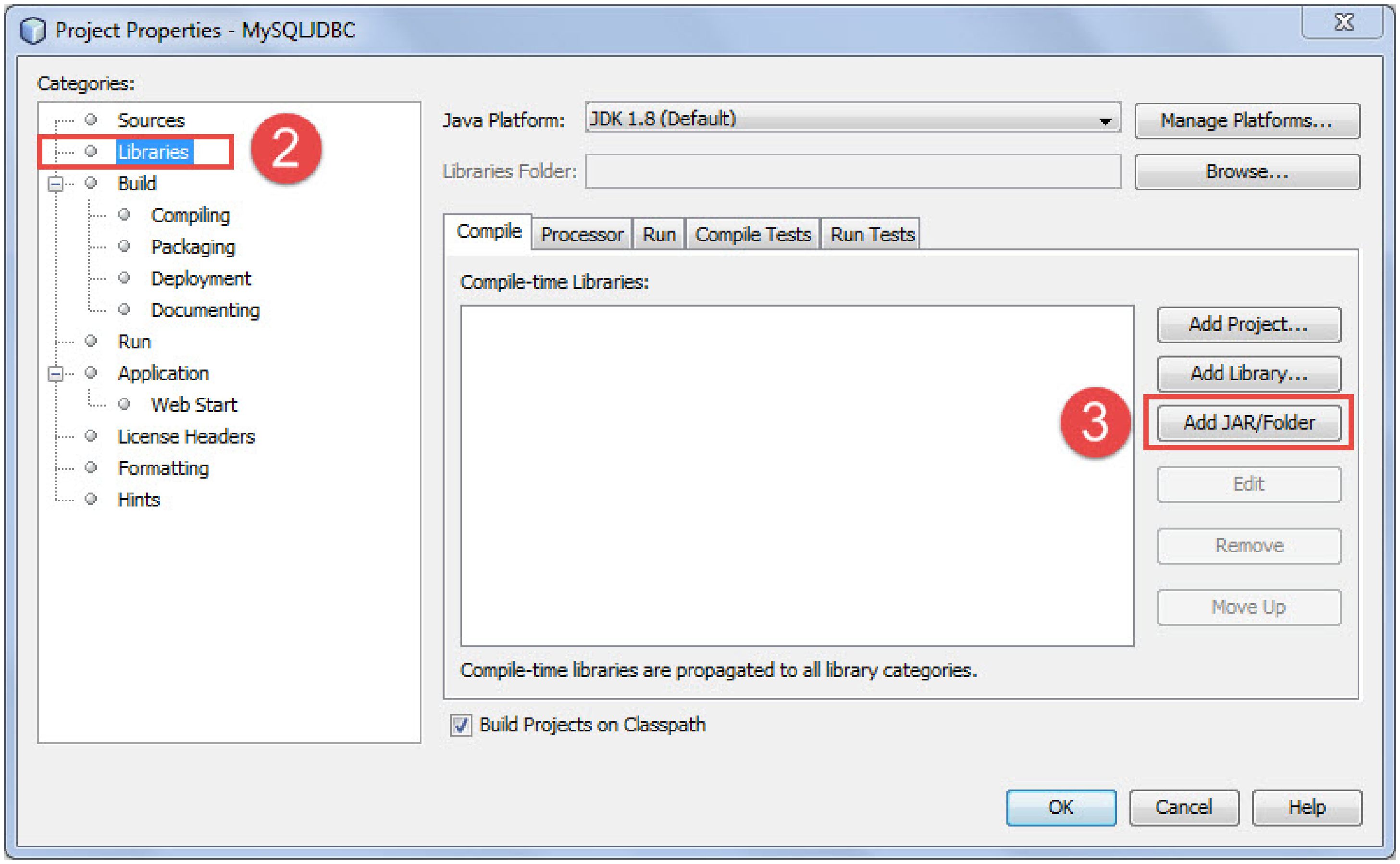
Loading MySQL Connector/J into your program

To load MySQL Connector/J into your program you follow three steps below:

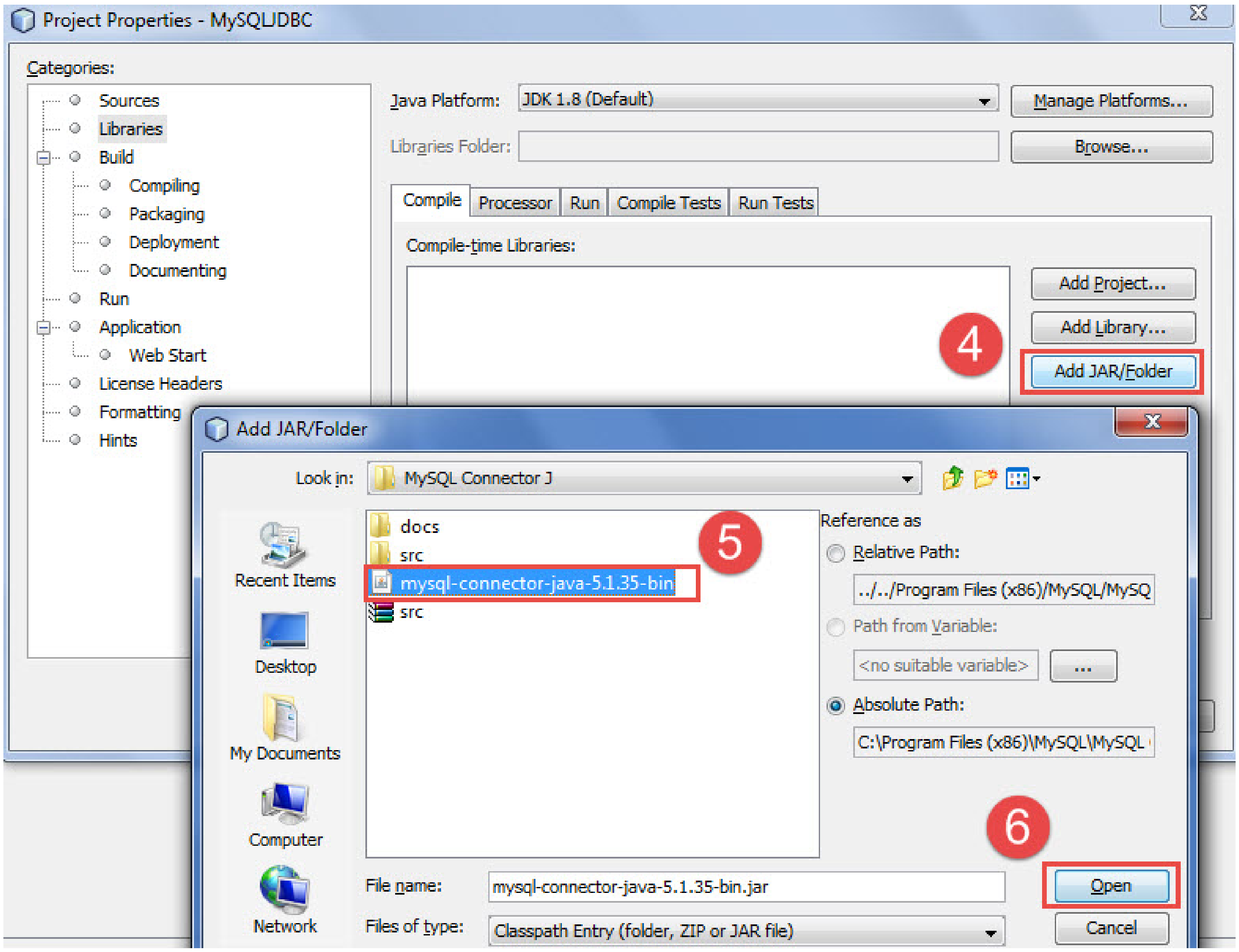
First, in NetBeans IDE, from project name, right mouse click and choose properties menu item. The project properties dialog will appear.



Second, on the left hand side of the project properties dialog, from the Categories section, choose Libraries item.



Third, click on the Add JAR folder button, browse to the location where you installed MySQL Connector/J, and choose the JAR file as screenshot below; after that click OK button.

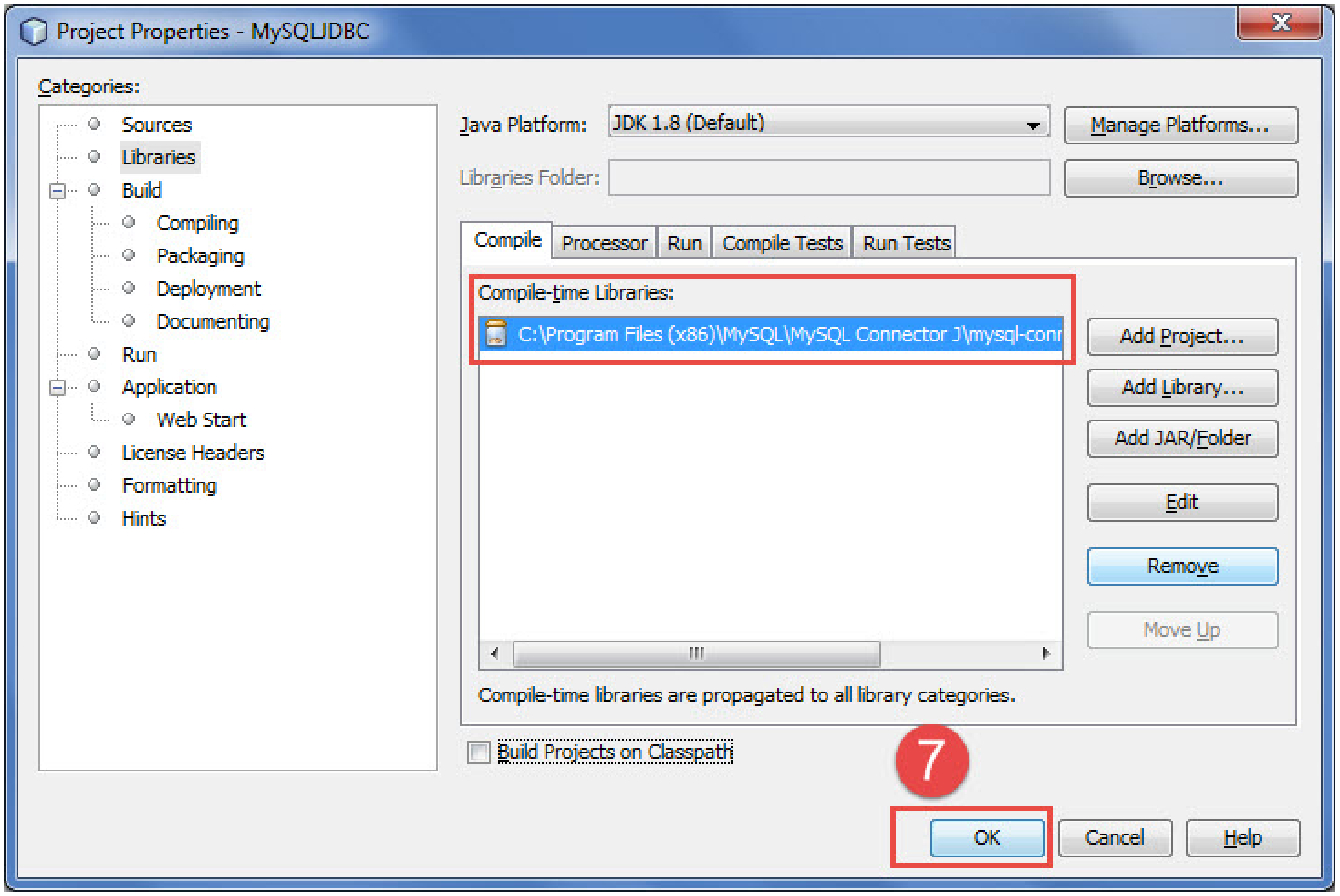


Java Platform: JDK 1.8 (Default)
Libraries Folder:

Look in: MySQL Connector J

Relative Path: ../../Program Files (x86)/MySQL/MySQ
Path from Variable: <no suitable variable> ...
Absolute Path: C:\Program Files (x86)\MySQL\MySQL

File name: mysql-connector-java-5.1.35-bin.jar
Files of type: Classpath Entry (folder, ZIP or JAR file)



Java Platform: JDK 1.8 (Default)

Libraries Folder:

Compile-time Libraries:

C:\Program Files (x86)\MySQL\MySQL Connector J\mysql-conr

Compile-time libraries are propagated to all library categories.

☐ Build Projects on Classpath

Connecting to MySQL database

First, you need to import three classes: SQLException, DriverManager, and Connection from the java.sql.* package.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Second, you call the getConnection() method of the DriverManager class to get the Connection object. There are three parameters you need to pass to the getConnection() method:

1. url: the database URL in the form jdbc:subprotocol:subname. For MySQL, you use the jdbc:mysql://localhost:3306/mysqljdbc i.e., you are connecting to the MySQL with server name localhost, port 3006, and database mysqljdbc.
2. user: the database user that will be used to connect to MySQL.
3. password: the password of the database user.

```java
Connection conn = null;
try {
    // db parameters
    String url       = "jdbc:mysql://localhost:3306/mysqljdbc";
    String user      = "root";
    String password  = "secret";

    // create a connection to the database
    conn = DriverManager.getConnection(url, user, password);
    // more processing here
    // ...
} catch(SQLException e) {
   System.out.println(e.getMessage());
} finally {
 try{
          if(conn ! null)
            conn.close()
 }catch(SQLException ex){
          System.out.println(ex.getMessage())
 }
}
```

When connecting to MySQL, anything could happens e.g., database server is not available, wrong user name or password, etc. in such cases, JDBC throws a `SQLException` . Therefore, when you create a Connection object, you should always put it inside a try catch block. Also you should always close the database connection once you complete interacting with database by calling `close()` method of the Connection object.

From Java 7, there is another nice statement called try-with-resources that allows you to simplify the code above as follows:

```java
// db parameters
String url       = "jdbc:mysql://localhost:3306/mysqljdbc";
String user      = "root";
String password  = "secret";
```

```

Connection conn = null;

try(conn = DriverManager.getConnection(url, user,
 password);) {
 // processing here
} catch(SQLException e) {
   System.out.println(e.getMessage());
}
```

It is automatically calls the close() method of the Connection object once program finishes. As you can see it’s cleaner and more elegant. However…

It is not secure as well as flexible when you hard coded the database parameters inside the code like above. In case you change the database server or password; you have to change the code, compile it again, which is not a good design.

To avoid hard coding all the database parameters in the code, you can use a Java properties file to store them. In case of changes, you just need to change them in the properties file and you don’t have to recompile the code.

Let’s take a look at the properties file named db.properties:

```
# MySQL DB parameters
user=root
password=secret
url=jdbc:mysql://localhost:3306/mysqljdbc
```

You can rewrite the code for creating a Connection object with parameters from a properties file as follows:

```
Connection conn = null;

try(FileInputStream f = new FileInputStream("db.p
roperties")) {
    // load the properties file
    Properties pros = new Properties();
    pros.load(f);
```