DAY 1

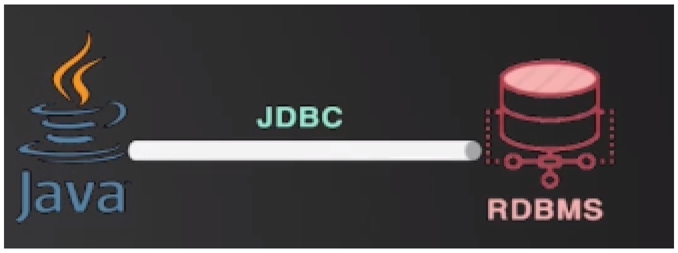
JDBC stands for **Java Database Connectivity.**

Why do we need JDBC?

We have seen that, whenever we had to save output of a Java program in a file, we were using File Handling. But even though file handling was a good approach, it had its own disadvantages when it comes to storing data or storing databases since we cannot store databases inside a file and hence it had the following disadvantages-

- Cannot store data in tabular form
- Multiple files cannot be related to each other i.e., we cannot store the data in the form of Relational Database.

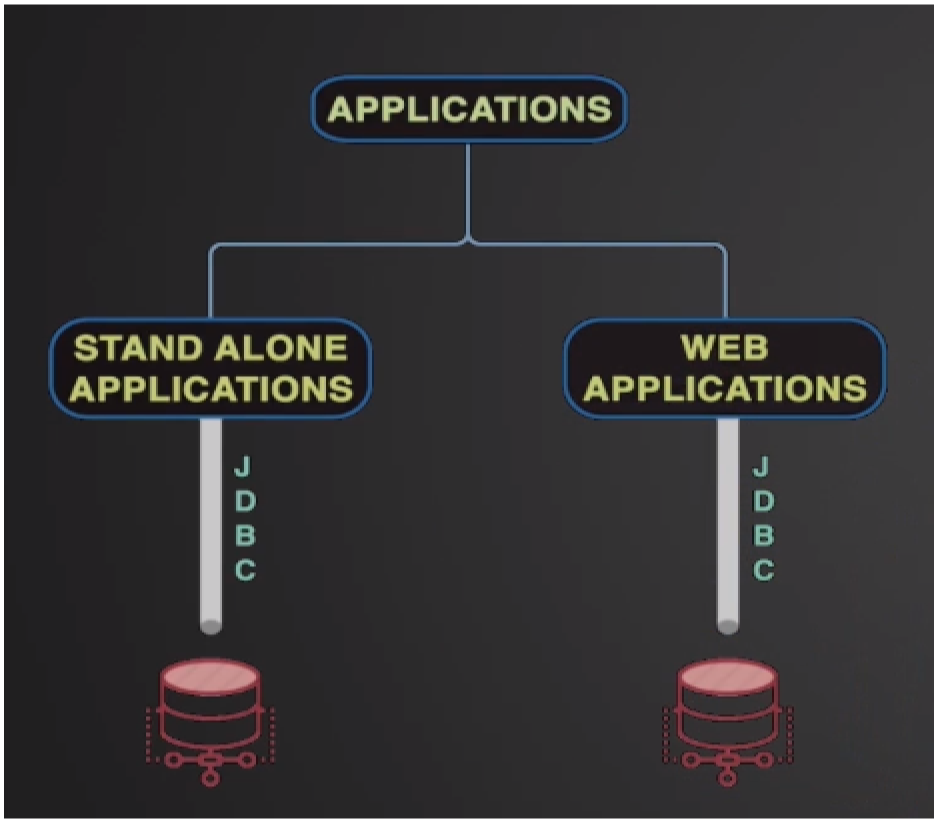
So, if we want to become a successful java developer, we should know how to establish a relation between the java program and the RBDMS. To establish this relation/data flow, we should have a connectivity called as **JDBC.**



We as a developer, will be developing two types of applications namely-

- Stand Alone Application
- Web Applications

Whenever we want to store or retrieve data, RDBMS plays a very important role. So, both standalone applications and web applications require connectivity to RDBMS and the concept of JDBC plays a very important role.



Program 1:

In any computer, there are two types of memory-

- RAM
- Hard disk

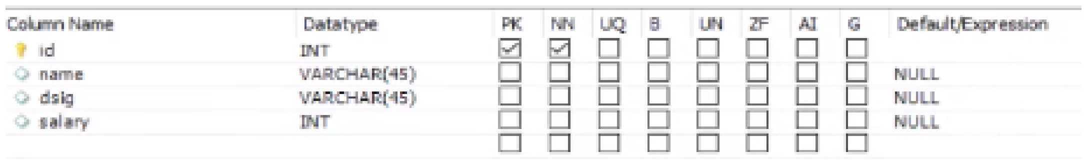
The database will be stored on the hard disk and this database will be managed by a software known as Relational database Management System (RDBMS).

There is a variety of RDBMS in the market but we will be using MySQL database in this course.

Every process which will be executing on the RAM will have a unique identification number known as **Process ID** and the process ID of MySQL is **3306.**

Now let us create a database in the GUI called **MySQL Workbench.**

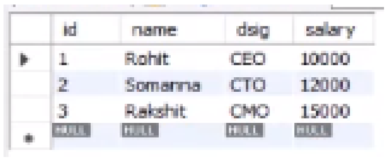
1. Create a schema (database), by clicking on “Create schema” and set the name of the database as “employee” and then click on “Apply”
2. Now a new schema(database) will be created.
3. Right-click on the schema and click on “Create Table”
4. Set the table name as “emp”
5. Add columns to emp table as shown below



Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	✓	✓	☐	☐	☐	☐	☐	☐	
name	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
dsig	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	☐	NULL
salary	INT	☐	☐	☐	☐	☐	☐	☐	☐	NULL
		☐	☐	☐	☐	☐	☐	☐	☐	

6. After adding all the columns and constraints to it, click on “Apply”.

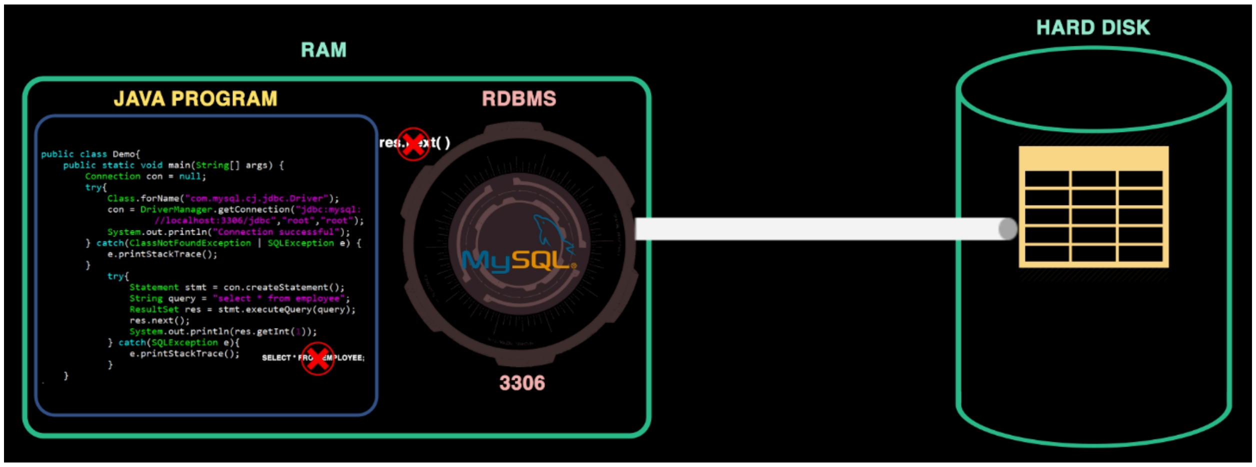
7. Now insert values inside the table like shown below.



id	name	dsig	salary
1	Rohit	CEO	10000
2	Somanna	CTO	12000
3	Rakshit	CMO	15000
NULL	NULL	NULL	NULL

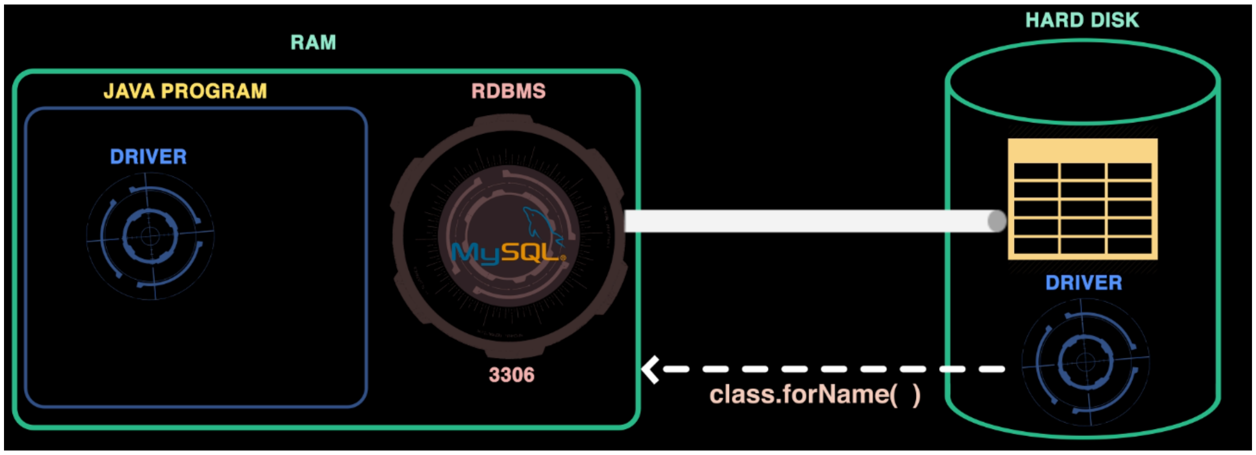
Now that we have created a table in the database.

Inside the RAM, a java program is executing and it wants to access the database, but the java program cannot directly access the data present inside the database as the data is in control of MySQL. The only way the java program can access the data is through MySQL. But MySQL can’t understand java and the java program can’t understand queries.



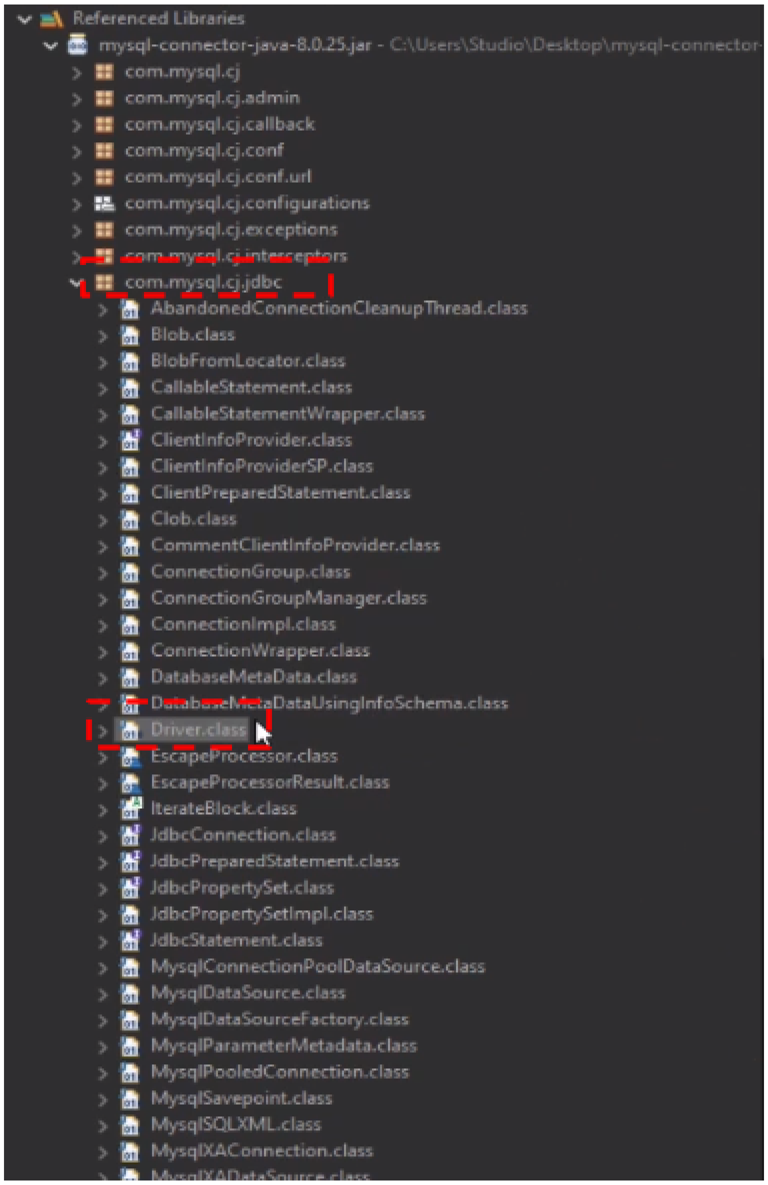
```
public class Demo{
    public static void main(String[] args) {
        Connection con = null;
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            con = DriverManager.getConnection("jdbc:mysql:
                    //localhost:3306/jdbc","root","root");
            System.out.println("Connection successful");
        } catch(ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
            try{
                Statement stmt = con.createStatement();
                String query = "select * from employee";
                ResultSet res = stmt.executeQuery(query);
                res.next();
                System.out.println(res.getInt(1));
            } catch(SQLException e){
                e.printStackTrace();
            }
    }
```

So, both of them can never directly interact with each other. To make the interaction possible we require a translator and the software responsible for this interaction is known as “Driver”.
Since we are using MySQL, there is a MySQL Driver software available on the internet, which is what we have downloaded and stored in the hard disk.
And if we want anything to be executed, it should be present on the RAM. In our case, it should be present inside our Java program
To achieve this we have to make use of a method Class.forName( ) in our java program.
forName( ) is a static method of a class called “Class”. forName( ) accepts the name of the driver as its parameter.



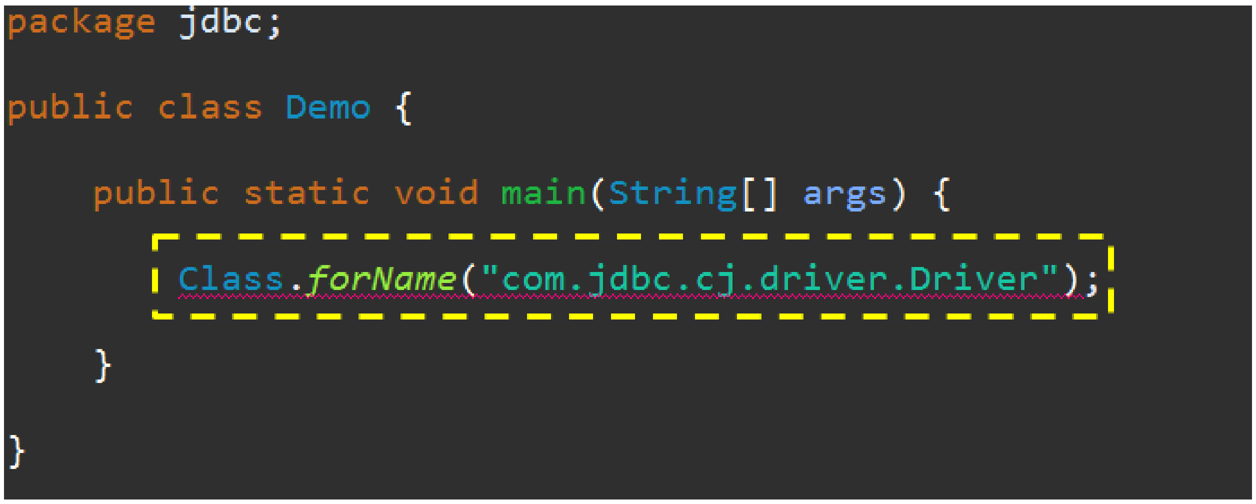
Now let’s see how to achieve this through our java program.
Step 1: Create a new java project and name the project as “jdbc”, and click on finish.
Step 2: Follow the steps provided in the PDF to download the jar file
Step 3: Right-click on the project “jdbc”, click on properties
Step 4: Click on ‘Java Build Path’, now click on the ‘Libraries’ tab
Step 5: Click on ‘Classpath’, (now all the buttons will be enabled).
Step 6: Click on the ‘Add External JARs’ button.
Step 7: Now select the ‘mysql connector’ file which you had previously downloaded and click on ‘open’.
Step 8: Click on ‘Apply’. And as soon as we click on ‘Apply’, a new folder “Referenced Libraries” will be created.
Step 9: Click on ‘Apply and Close’.

If we expand the ‘Referenced Libraries’ folder, we can see that ‘mysql-connector-java-8.0.25.jar’ jar file would have been added.
JAR file is a file that contains the ‘N’ number of Java class files. Now if we expand that JAR file, we can see that it contains many packages and if we expand any one package, we can see it contains many class files.
The Driver class is present in the **“com.mysql.cj.jdbc”** package.



```
Referenced Libraries
  mysql-connector-java-8.0.25.jar - C:\Users\Studio\Desktop\mysql-connector-
    com.mysql.cj
    com.mysql.cj.admin
    com.mysql.cj.callback
    com.mysql.cj.conf
    com.mysql.cj.conf.url
    com.mysql.cj.configurations
    com.mysql.cj.exceptions
    com.mysql.cj.interceptors
    com.mysql.cj.jdbc
      AbandonedConnectionCleanupThread.class
      Blob.class
      BlobFromLocator.class
      CallableStatement.class
      CallableStatementWrapper.class
      ClientInfoProvider.class
      ClientInfoProviderSP.class
      ClientPreparedStatement.class
      Clob.class
      CommentClientInfoProvider.class
      ConnectionGroup.class
      ConnectionGroupManager.class
      ConnectionImpl.class
      ConnectionWrapper.class
      DatabaseMetaData.class
      DatabaseMetaDataUsingInfoSchema.class
      Driver.class
      EscapeProcessor.class
      EscapeProcessorResult.class
      IterateBlock.class
      JdbcConnection.class
      JdbcPreparedStatement.class
      JdbcPropertySet.class
      JdbcPropertySetImpl.class
      JdbcStatement.class
      MysqlConnectionPoolDataSource.class
      MysqlDataSource.class
      MysqlDataSourceFactory.class
      MysqlParameterMetadata.class
      MysqlPooledConnection.class
      MysqlSavepoint.class
      MysqlSQLXML.class
      MysqlXAConnection.class
      MysqlXADataSource.class
```

Now let’s see how we can integrate the Driver.class inside our java program.
Create a Demo class inside your ‘jdbc’ project, and write the following code.

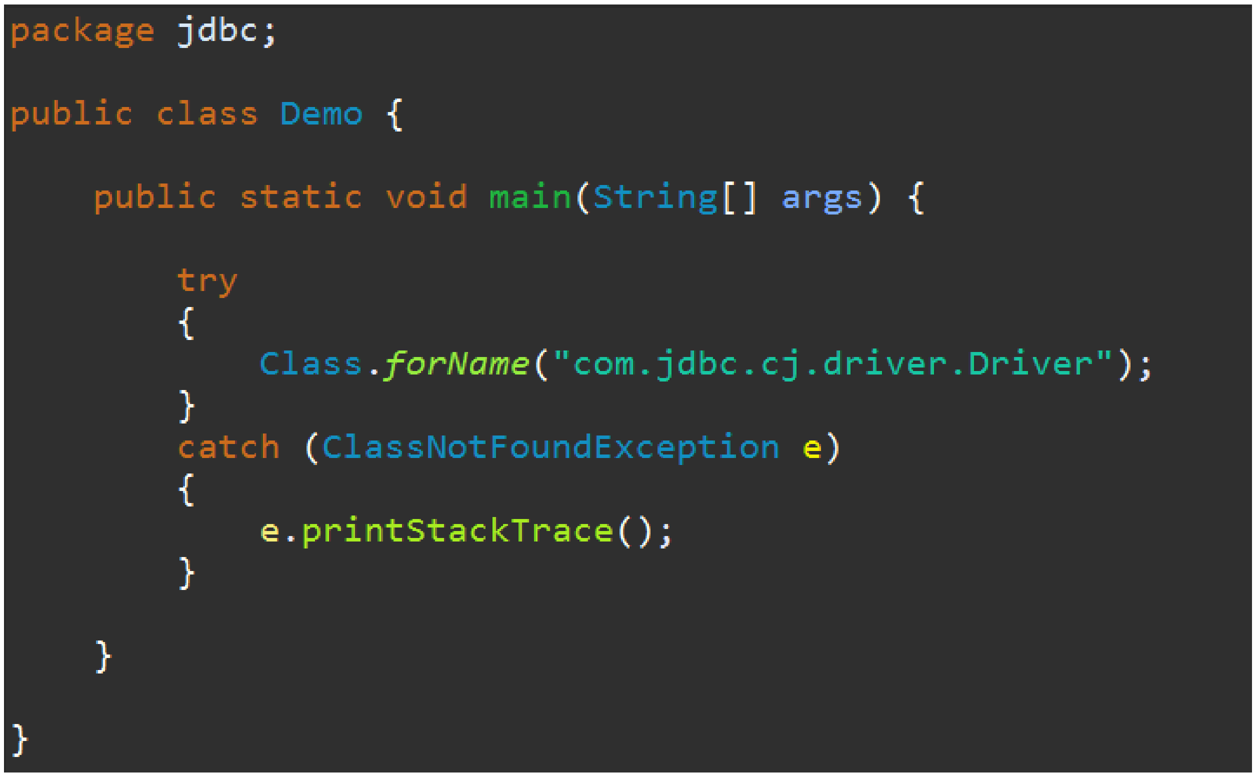


```java
package jdbc;

public class Demo {

    public static void main(String[] args) {
        Class.forName("com.jdbc.cj.driver.Driver");
    }

}
```

But as we can see, the code throws an error, because that line of code might throw an exception (ClassNotFoundException). The reason being, we are trying to load a class (Driver), but if that class is not present, then an exception may occur and the program might abruptly terminate. So to avoid this we can –

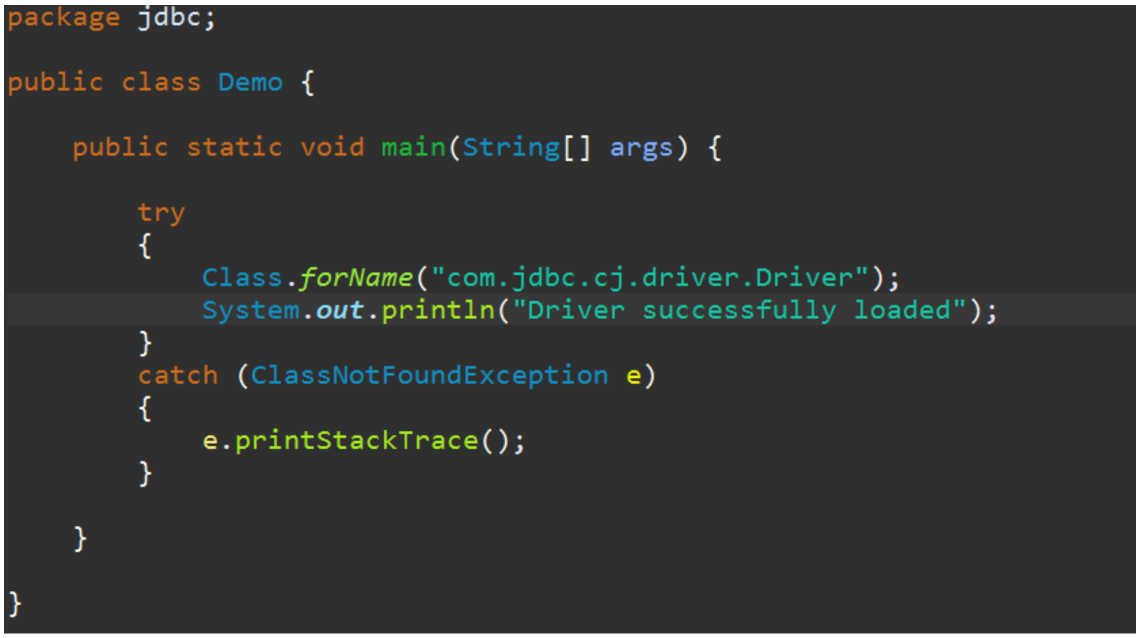
- Surround the risky code in a try-catch block
- Add throws declaration

We will surround the code in the try-catch block as shown below-



```java
package jdbc;

public class Demo {

    public static void main(String[] args) {

        try
        {
            Class.forName("com.jdbc.cj.driver.Driver");
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }

    }

}
```

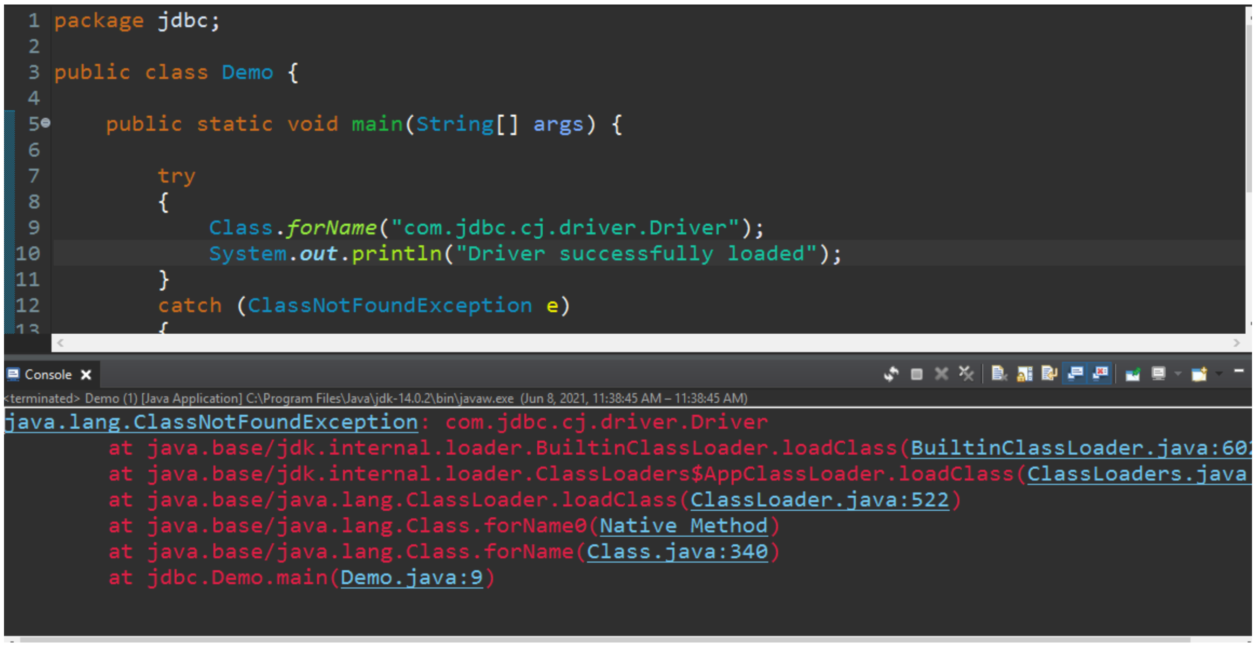
To verify whether the driver class has been successfully loaded, we will add a print statement as shown below-



```java
package jdbc;

public class Demo {

    public static void main(String[] args) {

        try
        {
            Class.forName("com.jdbc.cj.driver.Driver");
            System.out.println("Driver successfully loaded");
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }

    }

}
```

Let’s execute the program now,

But unfortunately, we got an exception as the path which we had mentioned in forName( ) was wrong. So we have to be careful whenever we add a path to avoid such problems.

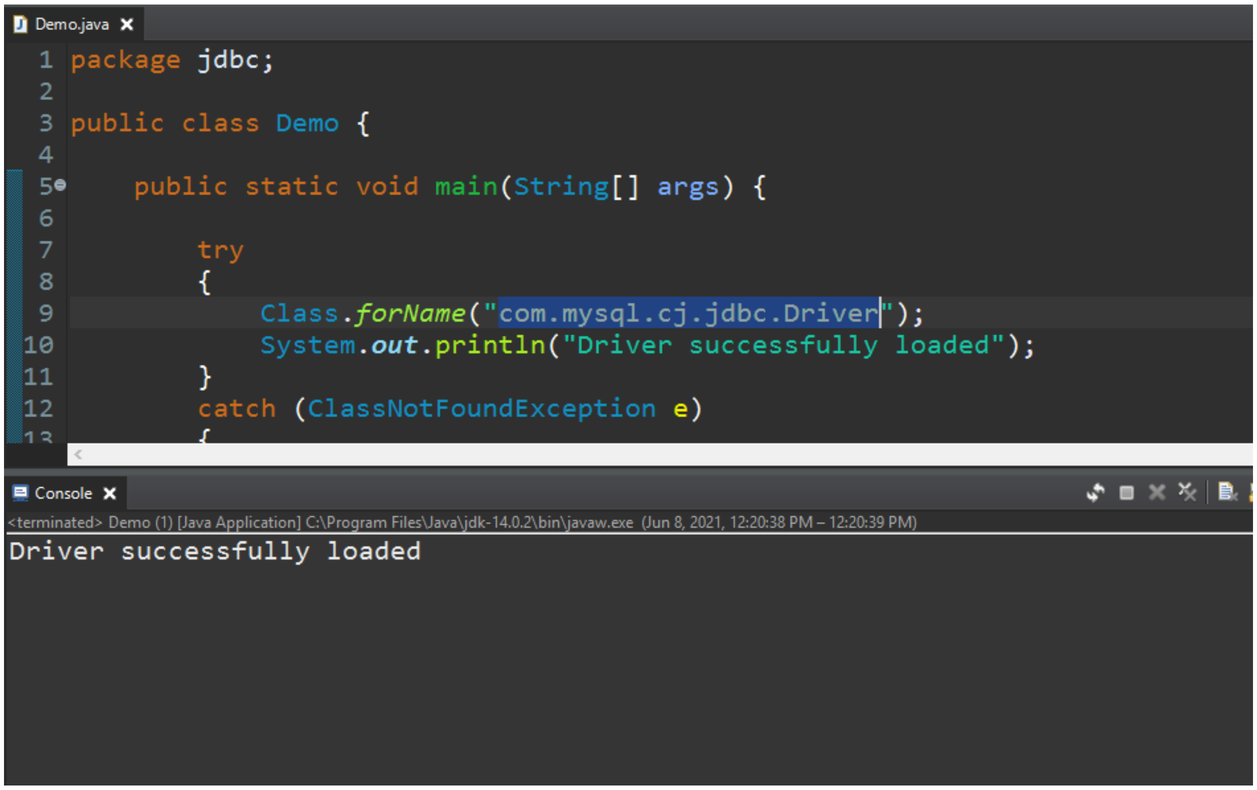


```java
package jdbc;

public class Demo {

    public static void main(String[] args) {

        try
        {
            Class.forName("com.jdbc.cj.driver.Driver");
            System.out.println("Driver successfully loaded");
        }
        catch (ClassNotFoundException e)
        {
```

Console

```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (Jun 8, 2021, 11:38:45 AM – 11:38:45 AM)
java.lang.ClassNotFoundException: com.jdbc.cj.driver.Driver
	at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java:60
	at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.java
	at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:522)
	at java.base/java.lang.Class.forName0(Native Method)
	at java.base/java.lang.Class.forName(Class.java:340)
	at jdbc.Demo.main(Demo.java:9)
```

Let us now give the correct path(“com.mysql.cj.jdbc.Driver”) and try executing the program again.



```
package jdbc;

public class Demo {

    public static void main(String[] args) {

        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver successfully loaded");
        }
        catch (ClassNotFoundException e)
        {
```

```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (Jun 8, 2021, 12:20:38 PM – 12:20:39 PM)
Driver successfully loaded
```

And as we can see, the driver was successfully loaded into the RAM.

Let us proceed to the next step-

Now that the driver is successfully loaded into the Java program, the Java program and MySQL can now interact with each other. But to achieve this, we have to establish a connection between our java program and MySQL.

We will call this connection ‘con’. Imagine this connection as a road through with information that can be sent from the java program to MySQL and vice versa.

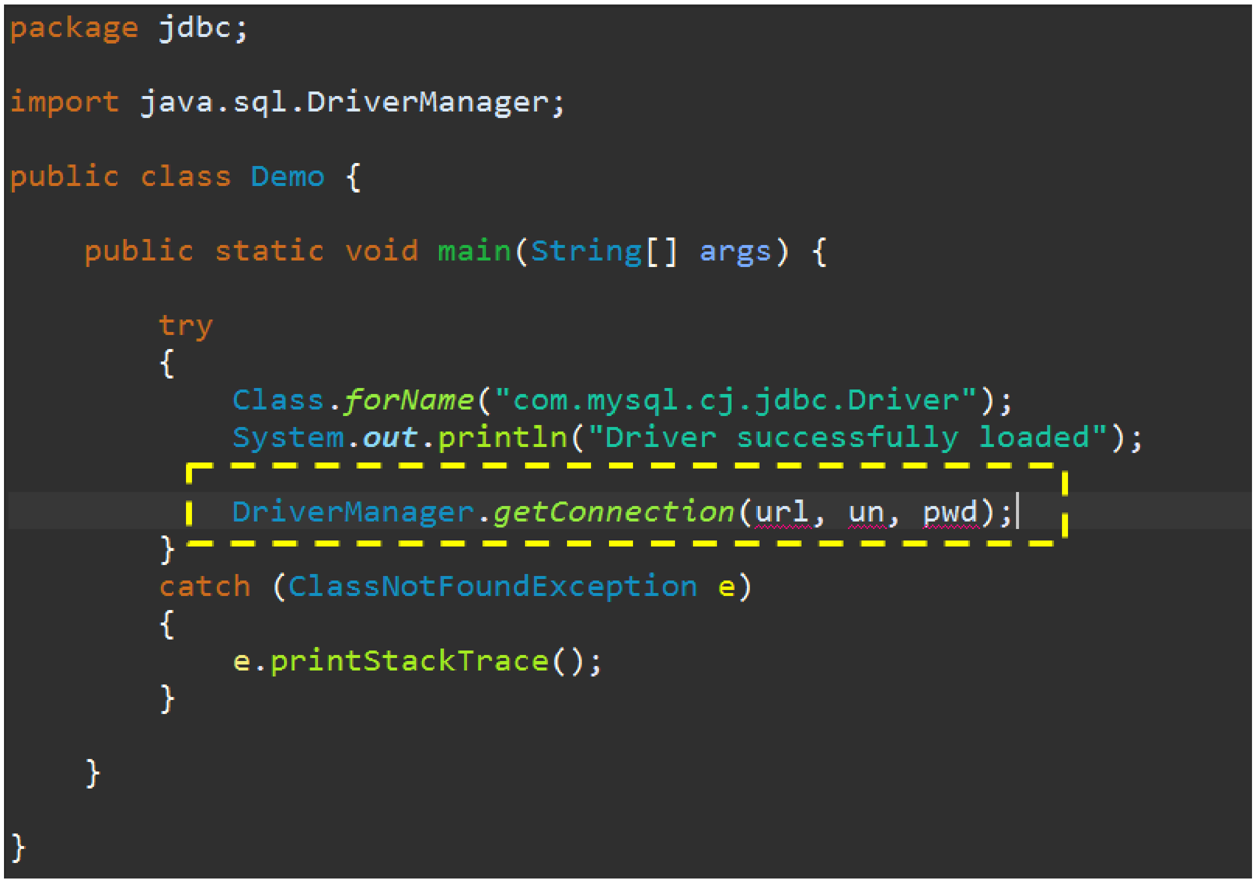
So how do we create this connection??

For that, we have to make use of a method called as getConnection( ).

getConnection( ) is a static method which is present inside DriverManager class.

Let us see how to establish the connection using the getConnection( ).

getConnection( ) method accepts 3 parameters as shown below-

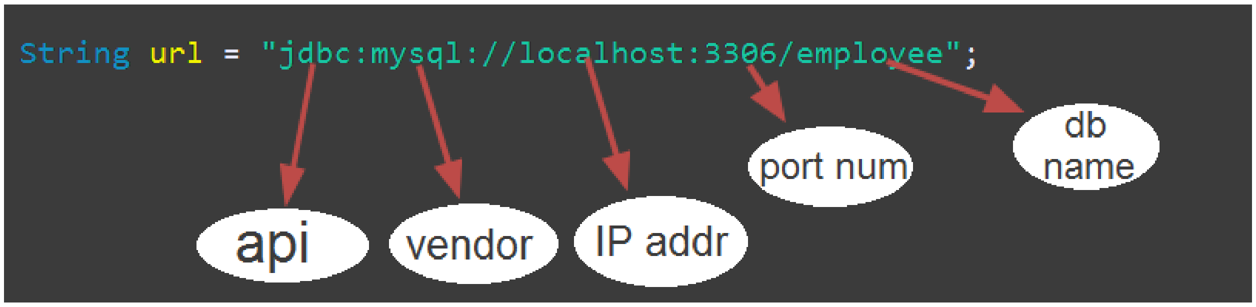


```java
package jdbc;

import java.sql.DriverManager;

public class Demo {

    public static void main(String[] args) {

        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver successfully loaded");

            DriverManager.getConnection(url, un, pwd);
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }

    }

}
```

In this scenario, both our java program and the database is present on the same computer, but in reality, it is hardly the case. The java program could be in one computer and the database could be in a different computer located in a different location. So if we want to establish a connection, we need a URL (Uniform Resource Locator).

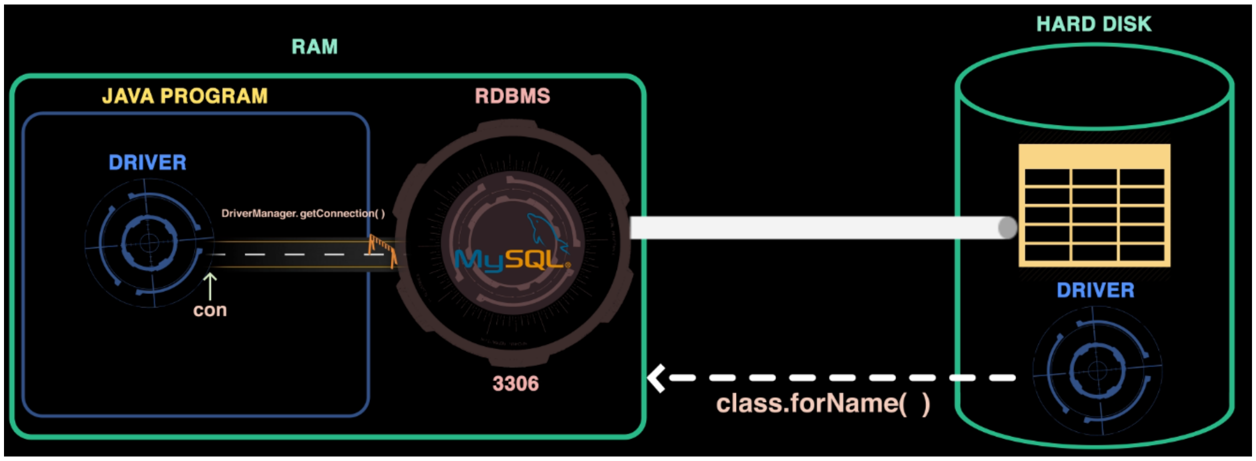
But even though the code and database are present in the same computer, we still need the URL to establish the connection.

This URL has a format, as shown below-



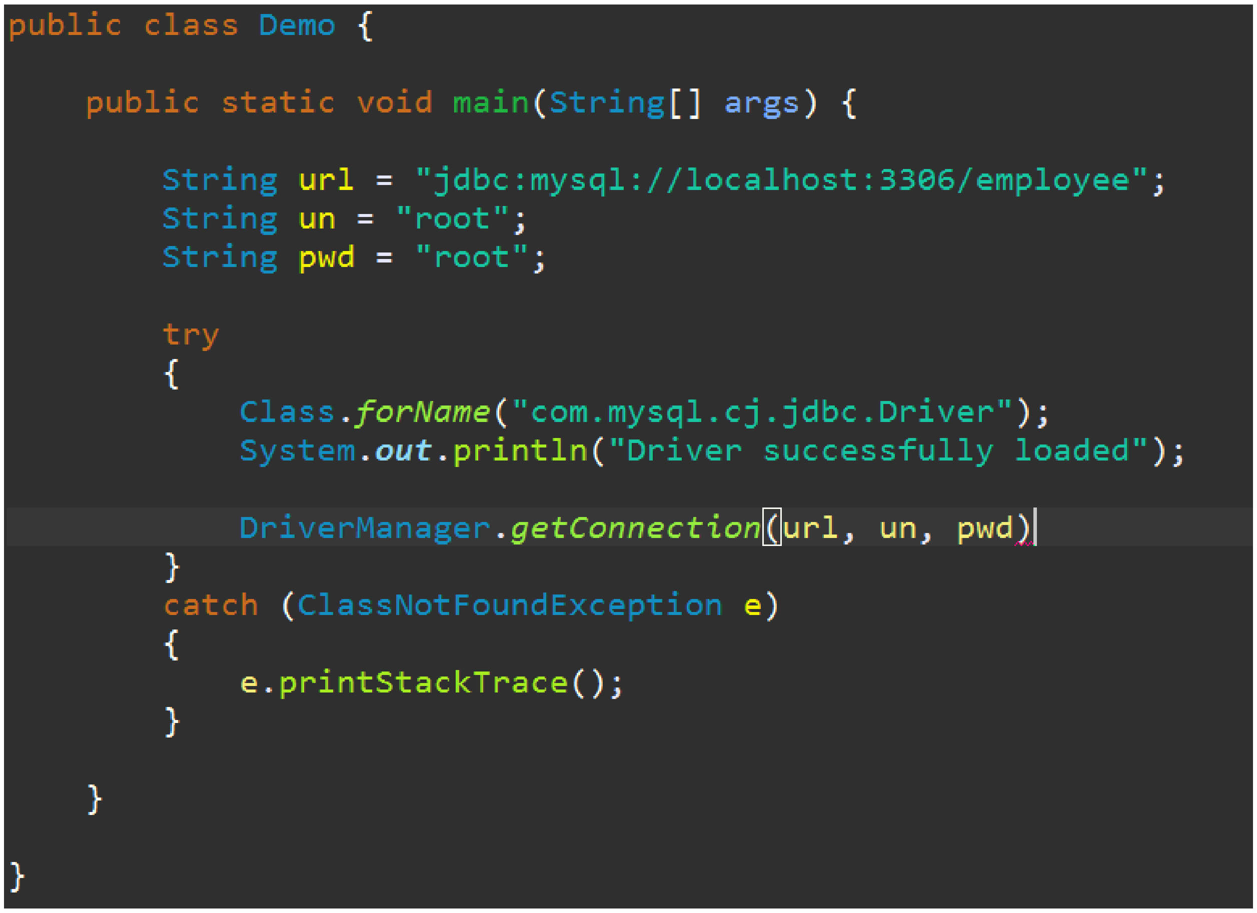
```java
String url = "jdbc:mysql://localhost:3306/employee";
```

Even though we have the URL, we cannot establish the connection to the database as the database will allow only those users who have the username and password of the database. So until the user provides the right username and password, it will barricade the entry point as shown below-



By default, the username and password for MySQL is **“root”.**

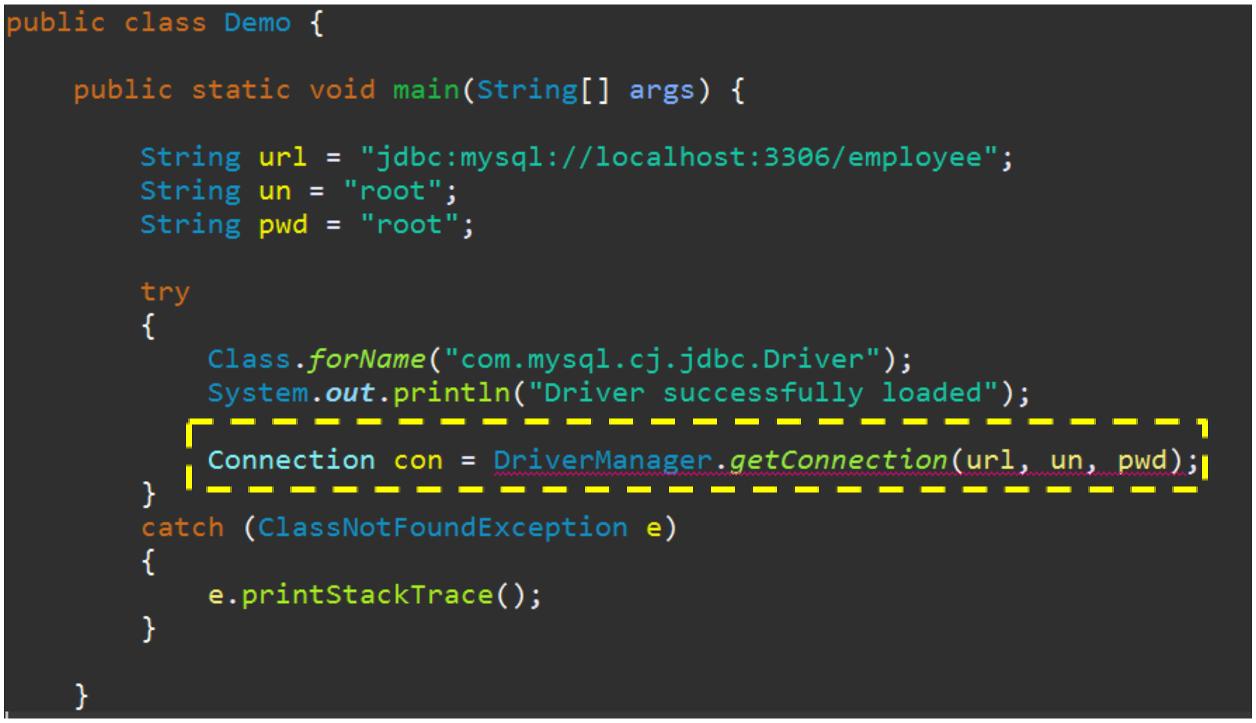
Let us provide the correct username and password in the code as shown below-



```
public class Demo {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/employee";
        String un = "root";
        String pwd = "root";

        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver successfully loaded");

            DriverManager.getConnection(url, un, pwd)
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }

    }

}
```

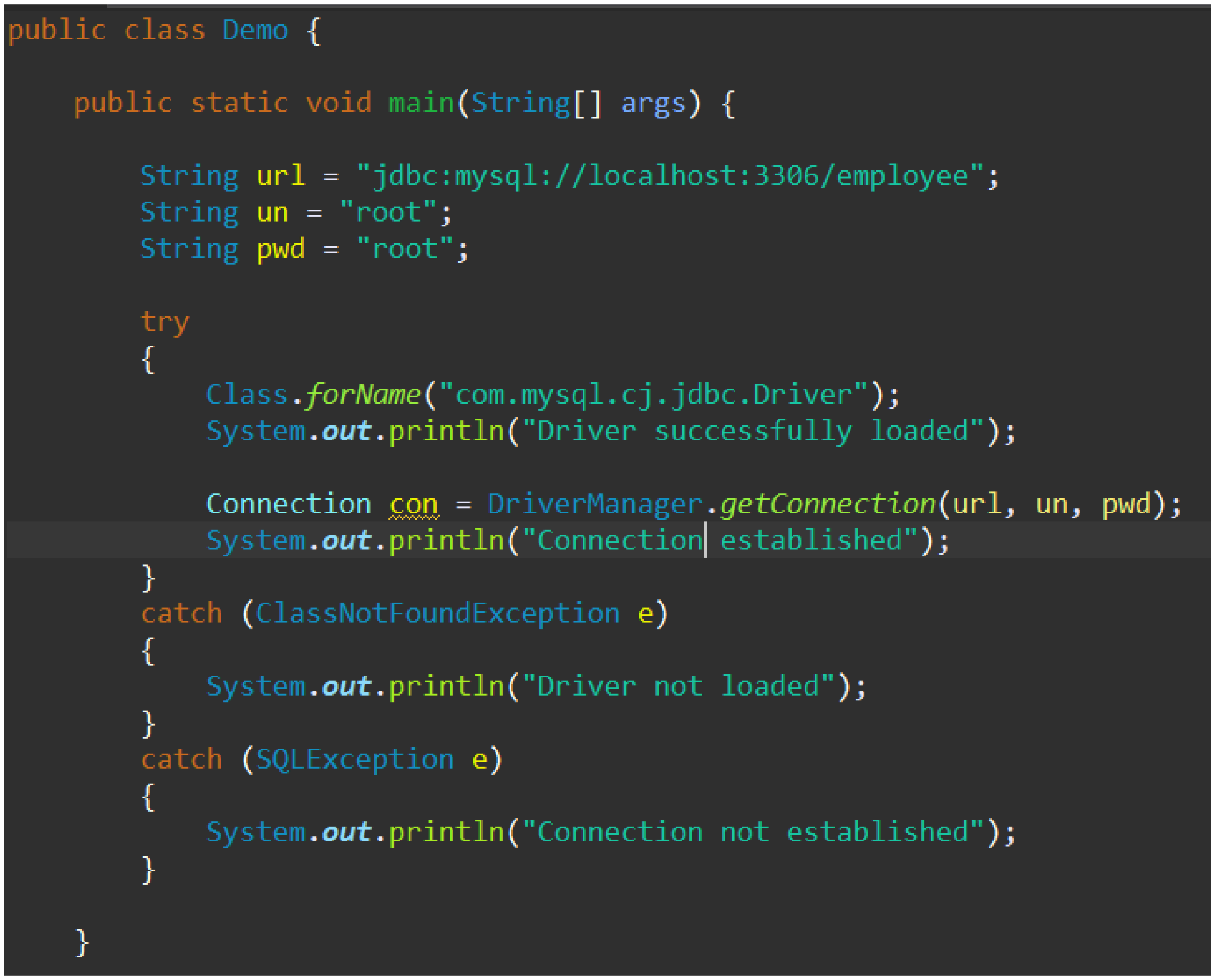
To establish the connection, we will give reference to the connection as ‘con’. Con is of type Connection.

Let us implement this in the code-


```
public class Demo {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/employee";
        String un = "root";
        String pwd = "root";

        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver successfully loaded");

            Connection con = DriverManager.getConnection(url, un, pwd);
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }

    }
```

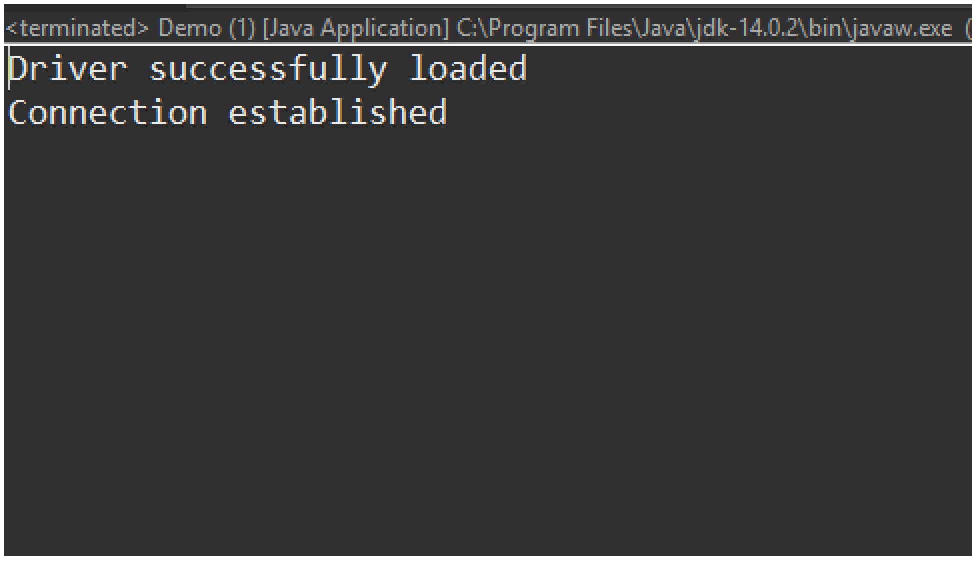
But as we can see, we are getting an error, as that line of code can throw an exception(url/un/pwd could be wrong or not provided) and the connection might not be established properly.

So let us surround the code in a try-catch block.



```
public class Demo {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/employee";
        String un = "root";
        String pwd = "root";

        try
        {
            Class.forName("com.mysql.cj.jdbc.Driver");
            System.out.println("Driver successfully loaded");

            Connection con = DriverManager.getConnection(url, un, pwd);
            System.out.println("Connection established");
        }
        catch (ClassNotFoundException e)
        {
            System.out.println("Driver not loaded");
        }
        catch (SQLException e)
        {
            System.out.println("Connection not established");
        }

    }
```

And when we execute the code, we can see that driver was successfully loaded and the connection was established.



```
<terminated> Demo (1) [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (
Driver successfully loaded
Connection established
```

Now that we have successfully established a connection, in the next class we try to understand how to write queries to fetch the data from the java program.