

---

①DBConnector を作成する。

Java とデータベースを繋げる為に自分達で作るファイルです。

今回ですと Mysql を使っていますので、Mysql 用のドライバーである

mysql-connector-java-5.1.46-bin.jar

使って Java と繋げます。この繋げる道の事をコネクションといいます。

---

#### 事前の準備

Java と DB 接続の 3、4 限目で作成したデータベースを使いますので、無い方はもう一度作成をしておいて下さい。

Java プロジェクト「TestDB」を選択する。

src フォルダを右クリックします。

新規→クラスを選択します。

「新規 Java クラス」画面が表示されます。

名前：DBConnector

「完了」ボタンを押下します。

---

②以下をプログラミングしましょう。

---

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
import java.sql.Connection;
```

```
public class DBConnector {
```

```
    private static String driverName = "com.mysql.jdbc.Driver";
```

```
    private static String url =
```

```
"jdbc:mysql://localhost/testdb?autoReconnect=true&useSSL=false";
```

```
    private static String user = "root";
```

```
    private static String password = "mysql";
```

```
    public Connection getConnection() {
```

```
        Connection con = null;
```

```
        try{
```

```
            Class.forName(driverName);
```

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

```
        } catch (ClassNotFoundException e) {
```

```
            e.printStackTrace() ;
```

```
        } catch (SQLException e) {
```

```
e.printStackTrace() ;  
}  
return con ;  
}  
}
```

## 解説

```
private static String driverName = "com.mysql.jdbc.Driver";
```

JDBC のドライバーの名前を変数に代入しています。ちなみにドライバーとは Java とデータベースを繋げる工具箱の様な物です。

```
private static String url =  
"jdbc:mysql://localhost/testdb?autoReconnect=true&useSSL=false";
```

今回は mysql を使っていますので、mysql 用の URL の指定の仕方になります。  
localhost (自分の使っている PC) testdb (データベース名) を使います。？以降はオプションなので必須ではありません。

```
private static String user = "root";
```

上で指定した root アカウントに対するパスワードを "mysql" に指定しています。

```
private static String password = "mysql";  
mysql にログインする為の値の準備をしています。
```

```
public Connection getConnection() {  
接続状態にする  
Connection con = null;  
一度状態を初期化にしています。  
この二つは接続しかないクラス。公式として覚えておくと良いです。
```

```
try{
```

```

Class.forName(driverName);
con = DriverManager.getConnection(url, user, password);
}
catch (ClassNotFoundException e) {
e.printStackTrace() ;
}
catch (SQLException e) {
e.printStackTrace() ;
}
return con ;
}
}

```

まず、try〜catch は Java の例外処理の為の構文です。

try の中にはエラーが発生しそうな処理を書きます。

```
Class.forName(driverName);
```

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

ここでは簡単に言うと、ドライバーがロードされ使えるような状態にしています。これは覚えて下さい。

try の中でエラーが発生した場合に、catch が受け取り、printStackTrace でエラーに至る履歴を表示してくれます。間違っただけの赤い文字です。今回だと 2 つのエラーが表示されます。

ClassNotFoundException (クラスが見つからない場合の例外) と SQLException (データベース処理に関する例外) です。

---

### ③TestUserDAO を作成する

ここでは何をしているのかを簡単に言うと、DB と会話を出来るクラスを作成しています。まずは、写経をしてみましょう。

---

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TestUserDAO {

    String name = "";
    String password = "";
    public void select(String name,String password) {
        DBConnector db = new DBConnector();

```

```

Connection con = db.getConnection();

String sql ="select * from test_table where user_name=? and password=?";
try {
    PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1, name);
    ps.setString (2, password);
    ResultSet rs=ps.executeQuery();
    if (rs.next()) {
        System.out.println(rs.getString("user_name"));
        System.out.println(rs.getString("password"));
    }
} catch (SQLException e ) {
    e.printStackTrace();
}

try {
    con.close() ;
} catch (SQLException e ) {
    e.printStackTrace();
}
}
}

```

---

## 解説

DBConnector **db** = **new** DBConnector();  
 Connection **con** = db.getConnection();  
 DB への接続の準備。DB と会話する為のコードと思って覚えて下さい。  
 これで mysql にログイン出来ました。

String **sql** ="select \* from test\_table where user\_name=? and password=?";  
 test\_table に入っているデータ user\_name=? password=?に入る 2 つの条件を満たしたデータが sql に代入されます。?はプレースホルダと言ってその都度違うデータを入れていきたい使用します。例えば user\_name=" taro" and password=" 123" とした場合は太郎と123しかデータを抽出することが出来なくなります。

```

PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1, name);
    ps.setString (2, password);
    ResultSet rs=ps.executeQuery();

```

PreparedStatement とは DB まで運んでくれる箱です。

executeQuery(); は実行メソッドで、必ず ResultSet が返ってきます。これは覚えて下さい。

**ResultSetは型の名前:表を出したい時に使う**

user\_name=? and password=?

データベースのカラム名

ps.setString(1, name);

ps.setString (2, password);

その中に入るデータ

user_name	password
taro	123
jiro	456
hanako	789

```
if (rs.next()) {  
    System.out.println(rs.getString("user_name"));  
    System.out.println(rs.getString("password"));  
}
```

ここでは二つの事を行っています。

②□を下に一行ずらすこと

②データが存在していれば戻り値を true で返す。存在しなければ false で返す

表で説明しますと、

user_name	password
taro	123
jiro	456
hanako	789

→の初期位置はカラム名つまり 0 行目からスタートします。今回は if (rs.next()) が実行されたので、一行下にずれます。そうすると

user_name	password
taro	123
jiro	456
hanako	789

この位置に→が移動して、中のデータを抽出してくれます。今回は if (rs.next()) なので一行で終了しましたが、この後に出てくる while (rs.next()) はデータが存在する限り、→を一行ずつ移動するという意味になります。

```
con.close()
```

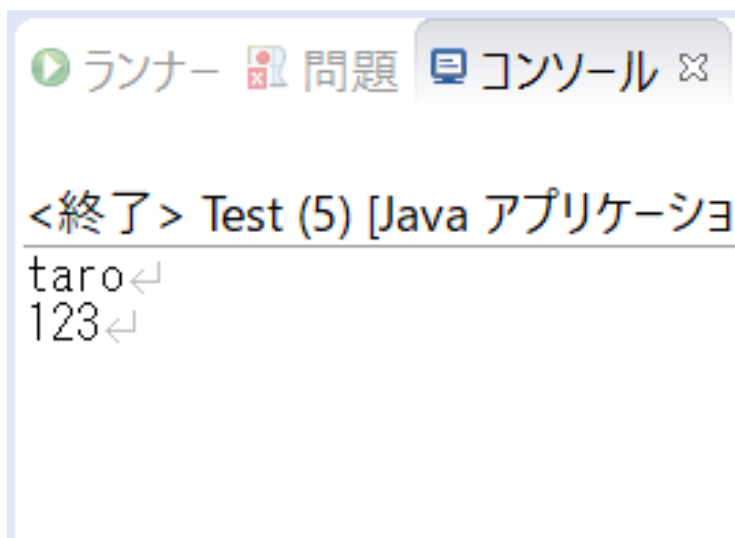
データベースとの接続を終了させるという意味です。これをしないとデータベースを接続したまま次の作業が実行されてしまい、メモリに負荷がかかりますので、終わりには必ず終了をするようにして下さい。

ここまで、色々と説明をしてきましたが、DAO の流れについては基本的に決まっています。DAO は DBConnector からインスタンス化をして、getConnection を呼びだして、mysql にログインをします。その後は SQL 文を書いて PreparedStatement の中にデータを入れて executeQuery もしくは updateQuery で実行して con.close をして接続を切ります。この先もほとんど構文は一緒です。まずはこの全体の流れを把握するところから覚えるところからはじめてみてください。

---

④Test クラスを作成して DAO クラスをインスタンス化およびメソッドを実行する。

```
public class Test {  
    public static void main(String[] args) {  
        TestUserDAO dao = new TestUserDAO();  
        dao.select("taro", "123");  
    }  
}
```



---

⑤TestUserDAO に以下のプログラムを追加してみましょう。

```
public void selectAll() {
    DBConnector db = new DBConnector();
    Connection con = db.getConnection();

    String sql ="select * from test_table";
    try {
        PreparedStatement ps = con.prepareStatement(sql);
        ResultSet rs=ps.executeQuery();
        while (rs.next()) {
            System.out.println(rs.getString("user_name"));
            System.out.println(rs.getString("password"));
        }
    } catch (SQLException e ) {
        e.printStackTrace();
    }
    try {
        con.close() ;
    } catch (SQLException e ) {
        e.printStackTrace();
    }
}
```

### 解説

while (rs.next())とはカーソルを1行ずつ実行していき、データがなくなったら実行を終了して下さいという意味。

表で表してみると・・・

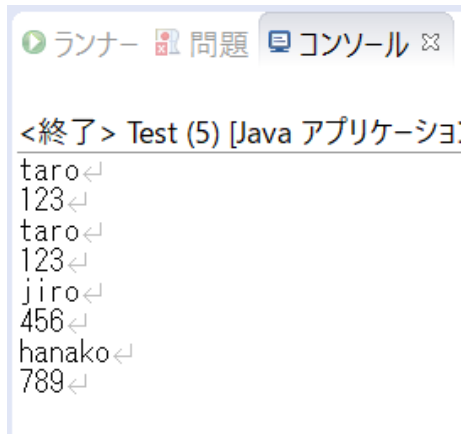
この行から1行ずつ入っているデータをみていく。  
データが入っていない所まできたら動作終了。



User_name	Password
123	taro
456	jiro
789	Hanako

⑥Test に以下のプログラムを追加して実行してみましょう。

```
dao.selectAll();
```



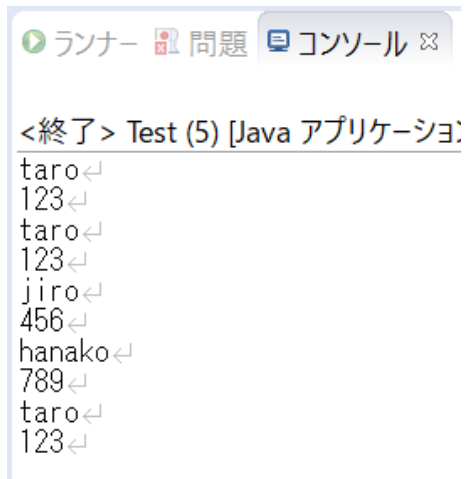
⑦TestUserDAO に以下のプログラムを追加してみましょう。

```
public void selectByName(String name) {  
    DBConnector db = new DBConnector();  
    Connection con = db.getConnection();  
  
    String sql ="select * from test_table where user_name=?";  
    try {  
        PreparedStatement ps = con.prepareStatement(sql);  
        ps.setString(1, name);  
        ResultSet rs=ps.executeQuery();  
        while (rs.next()) {  
            System.out.println(rs.getString("user_name"));  
            System.out.println(rs.getString("password"));  
        }  
    } catch (SQLException e ) {  
        e.printStackTrace();  
    }  
    try {  
        con.close() ;  
    } catch (SQLException e ) {  
        e.printStackTrace();  
    }  
}
```



⑧Test に以下のプログラムを追加して実行してみましょう。

```
dao.selectByName("taro");
```



```
<終了> Test (5) [Java アプリケーション:]
taro<
123<
taro<
123<
jiro<
456<
hanako<
789<
taro<
123<
```

⑨TestUserDAO に以下のプログラムを追加してみましょう。

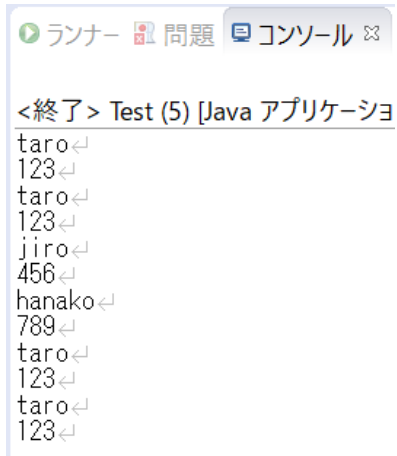
```
public void selectByPassword(String password) {
    DBConnector db = new DBConnector();
    Connection con = db.getConnection();

    String sql ="select * from test_table where password=?";
    try {
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString (1, password);
        ResultSet rs=ps.executeQuery();
        while (rs.next()) {
            System.out.println(rs.getString("user_name"));
            System.out.println(rs.getString("password"));
        }
    } catch (SQLException e ) {
        e.printStackTrace();
    }

    try {
        con.close() ;
    } catch (SQLException e ) {
        e.printStackTrace();
    }
}
```

⑩Test に以下のプログラムを追加して実行してみましょう。

```
dao.selectByPassword("123");
```



```
<終了> Test (5) [Java アプリケーショ]
taro↵
123↵
taro↵
123↵
jiro↵
456↵
hanako↵
789↵
taro↵
123↵
taro↵
123↵
```

⑪TestUserDAO に以下のプログラムを追加してみましょう。

```
public void updateUserNameByUserName(String oldName,String newName) {
    DBConnector db = new DBConnector();
    Connection con = db.getConnection();

    String sql ="update test_table set user_name=? where user_name=?";
    try {
        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, newName);
        ps.setString (2, oldName);
        int i=ps.executeUpdate();
        if (i>0) {
            System.out.println(i + "件更新されました");
        }else{
            System.out.println("該当するデータはありませんでした");
        }
    } catch (SQLException e ) {
        e.printStackTrace();
    }
    try {
        con.close() ;
    } catch (SQLException e ) {
        e.printStackTrace();
    }
}
```

```
}
```

## 解説

```
int i=ps.executeUpdate()
```

なぜ int 型なのか？executeUpdate() はデータの件数(数値) を返している。

SQL 文	メソッド	戻り値
select	executeQuery	ResultSet
Insert Delete Update	executeUpdate	int

こちらの表はぜひ覚えておいて下さい。

⑫Test に以下のプログラムを追加して実行してみましょう。

```
dao.updateUserNameByUserName("taro", "saburo");
```

⑬TestUserDAO に以下のプログラムを追加してみましょう。

```
public void insert(int user_id,String name, String password) {  
    DBConnector db = new DBConnector();  
    Connection con = db.getConnection();  
  
    String sql ="insert into test_table values(?, ?, ?)";  
    try {  
        PreparedStatement ps = con.prepareStatement(sql);  
        ps.setInt(1, user_id);
```

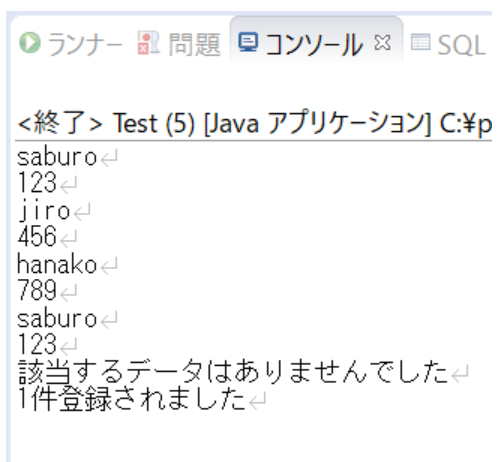
```

        ps.setString(2, name);
        ps.setString (3, password);
        int i=ps.executeUpdate();
        if (i>0) {
            System.out.println(i + "件登録されました");
        }
    } catch (SQLException e ) {
        e.printStackTrace();
    }
}
try {
    con.close() ;
} catch (SQLException e ) {
    e.printStackTrace();
}
}

```

⑭Test に以下のプログラムを追加して実行してみましょう。

```
dao.insert(4, "shiro", "012");
```



```

<終了> Test (5) [Java アプリケーション] C:¥p
saburo↵
123↵
jiro↵
456↵
hanako↵
789↵
saburo↵
123↵
該当するデータはありませんでした↵
1件登録されました↵

```

⑮TestUserDAO に以下のプログラムを追加してみましょう。

```

public void delete(String name) {
    DBConnector db = new DBConnector();
    Connection con = db.getConnection();

    String sql ="delete from test_table where user_name=?";
    try {

```

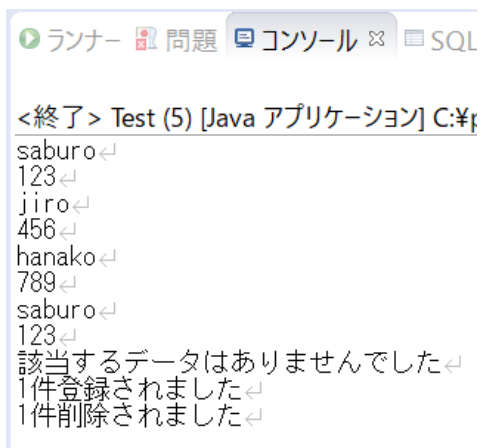
```

        PreparedStatement ps = con.prepareStatement(sql);
        ps.setString(1, name);
        int i=ps.executeUpdate();
        if (i>0) {
            System.out.println(i + "件削除されました");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
try {
    con.close() ;
} catch (SQLException e) {
    e.printStackTrace();
}
}

```

⑩Test に以下のプログラムを追加して実行してみましょう。

```
dao.delete("jiro");
```

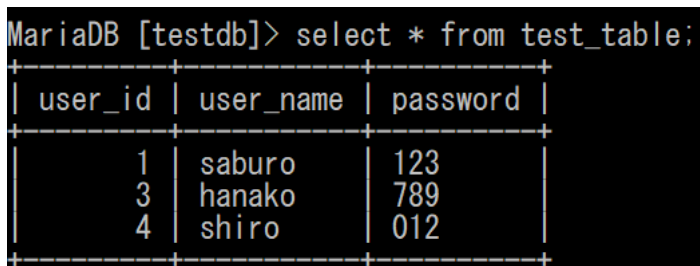


```

<終了> Test (5) [Java アプリケーション] C:\¥f
saburo↵
123↵
jiro↵
456↵
hanako↵
789↵
saburo↵
123↵
該当するデータはありませんでした↵
1件登録されました↵
1件削除されました↵

```

⑪最後にコマンドプロンプトでデータベースの中身を確認してみましょう。



```

MariaDB [testdb]> select * from test_table;
+----+-----+-----+
| user_id | user_name | password |
+----+-----+-----+
| 1 | saburo | 123 |
| 3 | hanako | 789 |
| 4 | shiro | 012 |
+----+-----+-----+

```