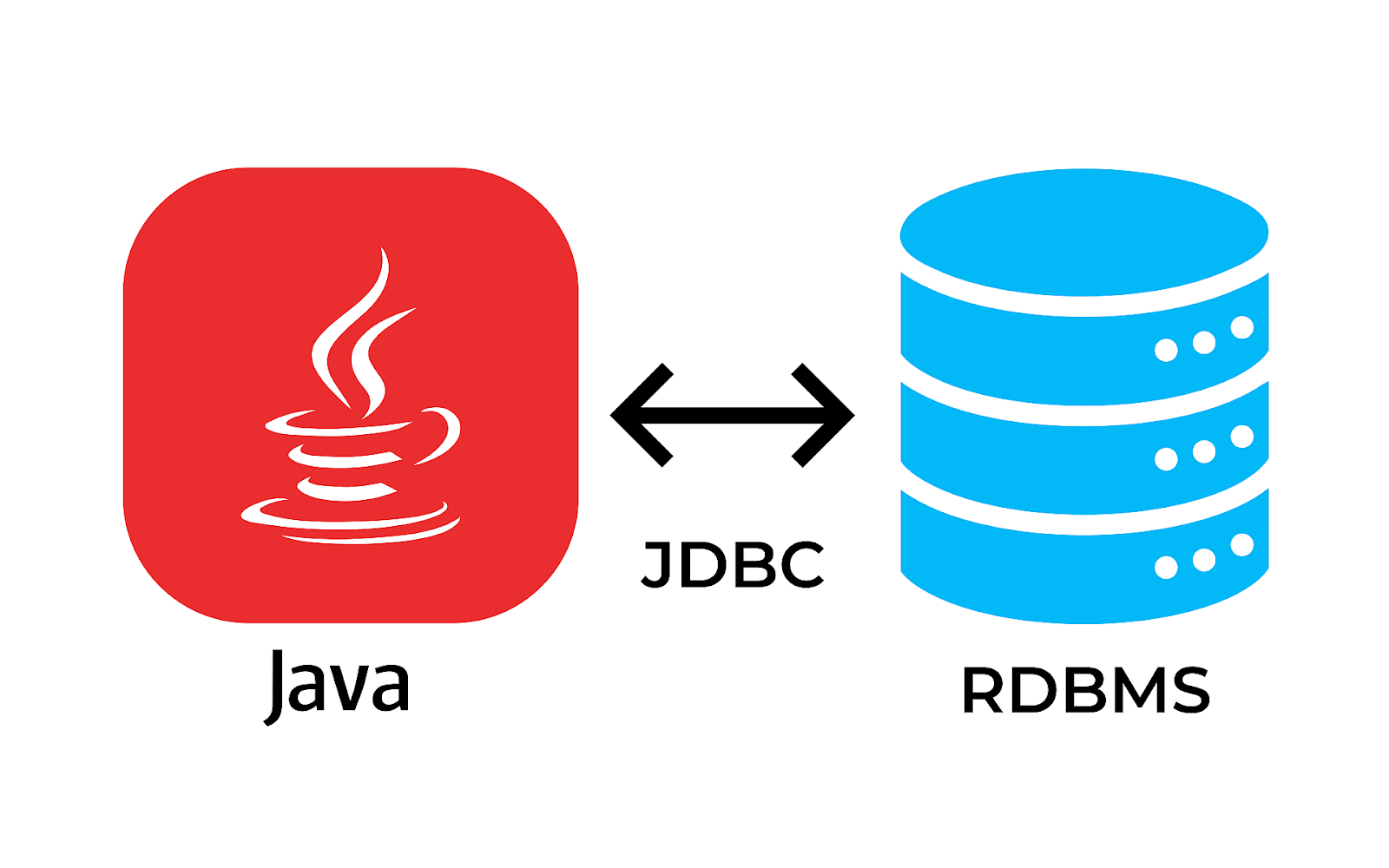
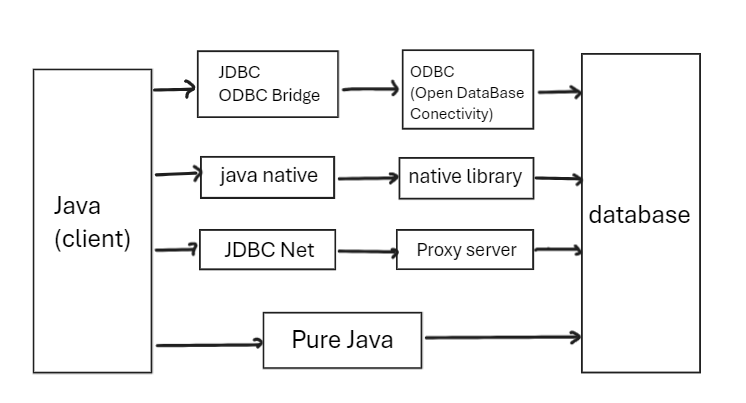
# Kết nối cơ sở dữ liệu với chương trình bằng JDBC



# **Các khái niệm cơ bản**

* JDBC (Java Database Connectivity) là một API trong Java cho phép chương trình java (phía client) kết nối và tương tác (trực tiếp) với các cơ sở dữ liệu.
* Nếu sử dụng JDBC như thế này, client sẽ không thông qua server mà tương tác trực tiếp với database. Nếu sử dụng framework như Spring thì sẽ phải thông qua bước kiểm duyệt ở máy chủ đối với một vài request cần kiểm duyệt (GPT bảo thế).
* Các cách để kết nối java client với database bằng JDBC:

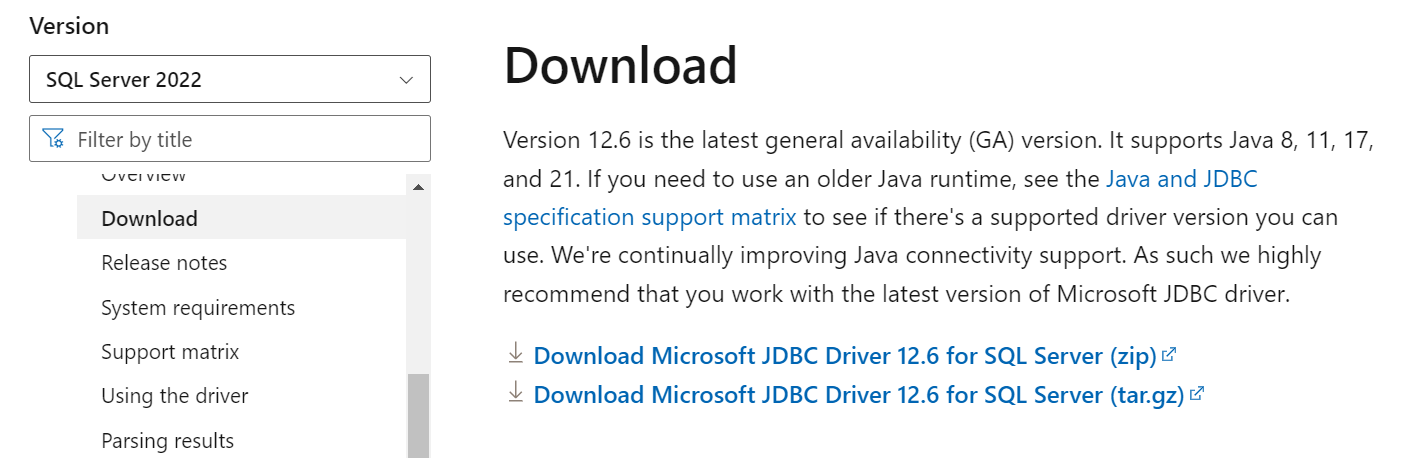


Trong khuôn khổ bài này chỉ nói về cách sử dụng pure java.

# **Cài đặt**

## Download driver

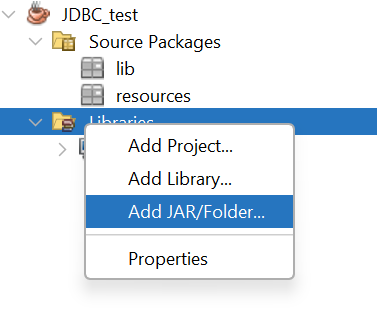
* Đầu tiên download driver của JDBC với MSSQL.
* Đây là một phần mềm trung gian đóng vai trò quan trọng trong việc giúp ứng dụng Java giao tiếp với cơ sở dữ liệu. Nó chịu trách nhiệm chuyển đổi các lệnh JDBC API của Java thành các lệnh cụ thể mà cơ sở dữ liệu có thể hiểu và thực thi.
* Các chức năng chính của driver JDVC:
  + Chuyển đổi lệnh (trong trường hợp ODBC hay JDBC Net, khi mà câu lệnh thực thi không phải câu truy vấn mà CSDL thực thi ngay được. nếu không thì công việc của Driver chỉ là chuyển tiếp câu lệnh từ client sang DBMS).
  + Quản lý kết nối (Quản lý việc mở và đóng kết nối đến cơ sở dữ liệu).
  + Thưc hiện truy vấn (Gửi các truy vấn từ ứng dụng Java đến cơ sở dữ liệu và nhận kết quả trả về).
  + Xử lý kết quả (Nhận kết quả từ cơ sở dữ liệu, chuyển thành dạng mà chương trình java có thể sử dụng (ResultSet)).
* Link download: <https://learn.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16>



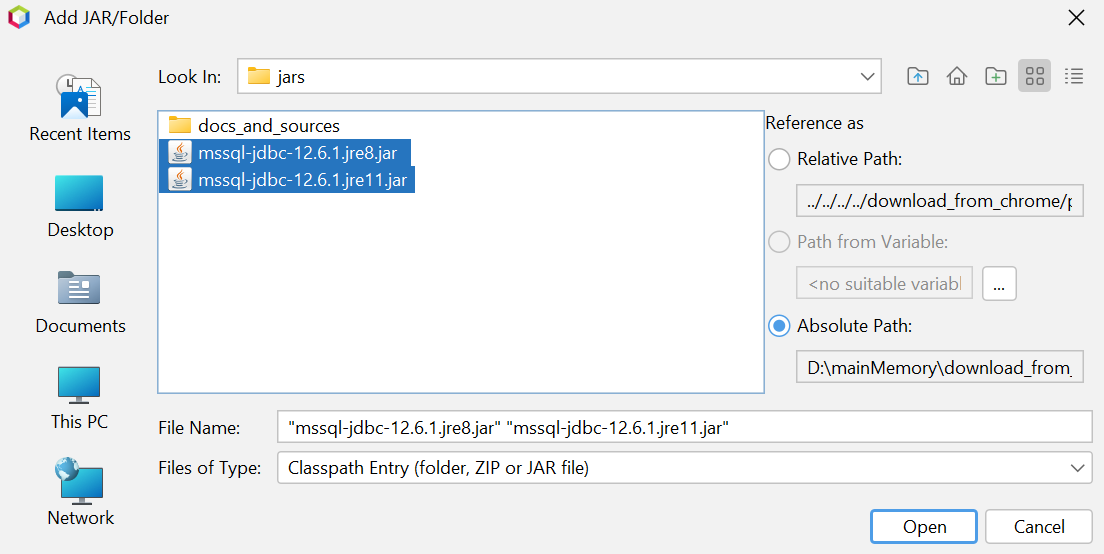
Tải một trong 2 file này về và giải nén. Bên trong đó có chứa file jar chứa các driver.

## Add driver vào thư viện (với Netbeans)

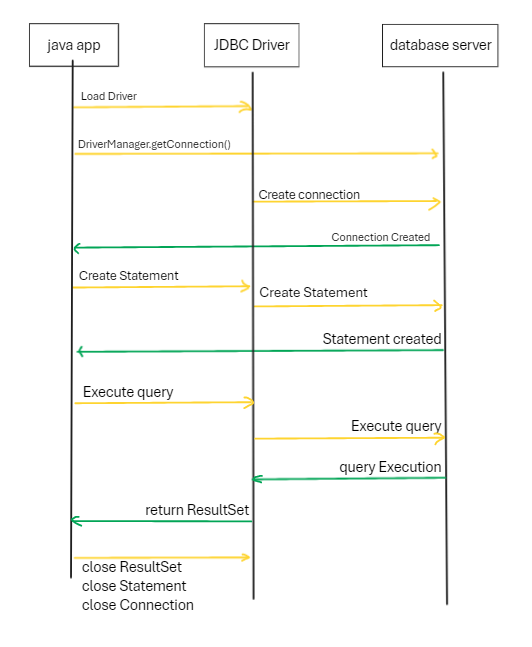
* Chuột phải vào thư mục ‘thư viện’ và chọn ‘add JAR/Folder’



* Tại đây chọn 2 file jar chứa driver và ‘open’.



## Viết code pure java để connect



Giải thích các bước

1. Load Driver: Ứng dụng Java tải driver JDBC vào bộ nhớ.

2. DriverManager.getConnection(): Ứng dụng Java gọi phương thức để tạo kết nối đến cơ sở dữ liệu thông qua DriverManager.

3. Create Connection: JDBC driver thực hiện việc tạo kết nối đến cơ sở dữ liệu.

4. Connection Created: Kết nối được tạo thành công và trả về cho ứng dụng Java.

5. Create Statement: Ứng dụng Java tạo một đối tượng Statement để gửi các lệnh SQL.

6. Create Statement: JDBC driver tạo đối tượng Statement trong cơ sở dữ liệu.

7. Statement Created: Đối tượng Statement được tạo thành công và trả về cho ứng dụng Java.

8. Execute Query: Ứng dụng Java gửi lệnh SQL thông qua đối tượng Statement.

9. Execute Query: JDBC driver truyền lệnh SQL đến cơ sở dữ liệu để thực thi.

10. Query Execution: Cơ sở dữ liệu thực thi lệnh SQL và tạo kết quả.

11. Return ResultSet: Kết quả của truy vấn được trả về cho JDBC driver dưới dạng ResultSet.

12. <Process ResultSet: Ứng dụng Java xử lý dữ liệu từ ResultSet.>

13. Close ResultSet: Ứng dụng Java đóng ResultSet sau khi xử lý xong.

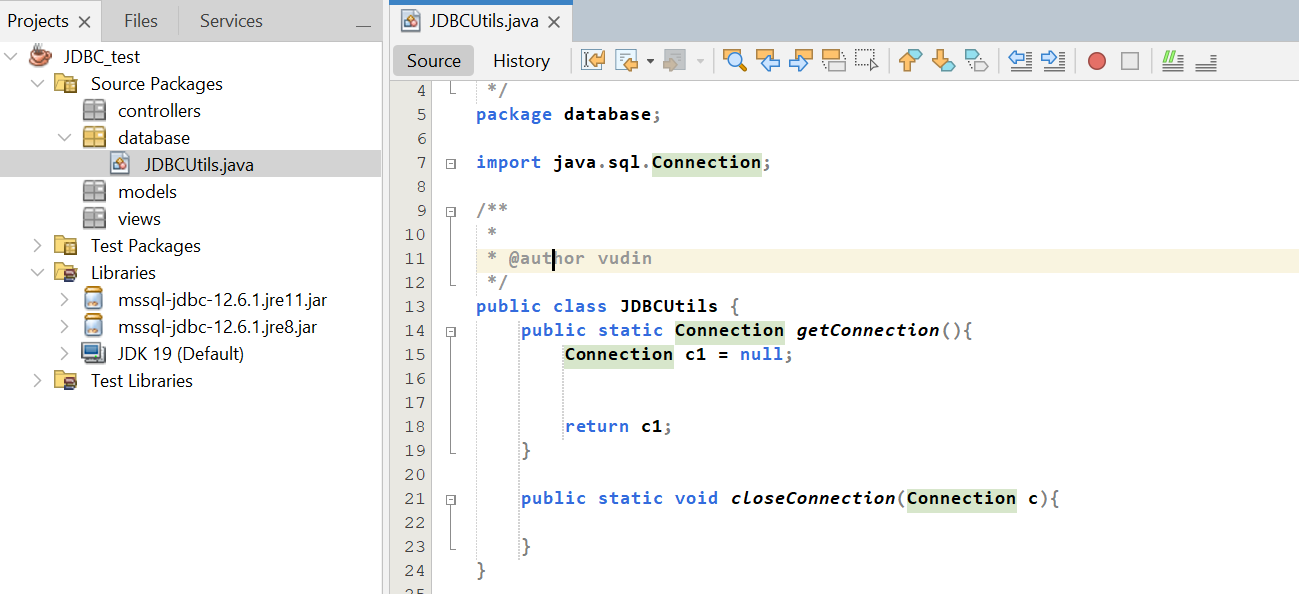
14. Close Statement: Ứng dụng Java đóng đối tượng Statement.

15. Close Connection: Ứng dụng Java đóng kết nối đến cơ sở dữ liệu để giải phóng tài nguyên.

Sau đây sẽ cụ thể hóa phần này bằng code và giải thích rõ hơn từng bước.

### **Các bước thực hiện cụ thể:**

1. Đầu tiên tạo một Project mới theo dạng MVC, có thêm package ‘database’ để chứa các class kết nối.
2. Trong package ‘database’, tạo ra file ‘JDBCUtils.java’ để chứa các phương thức kết nối: trước hết có getConnection() và closeConnection().



*Đọc thêm về class Connection:*

[*https://chatgpt.com/share/57189851-b54e-416f-a288-0a21c643f42d*](https://chatgpt.com/share/57189851-b54e-416f-a288-0a21c643f42d)

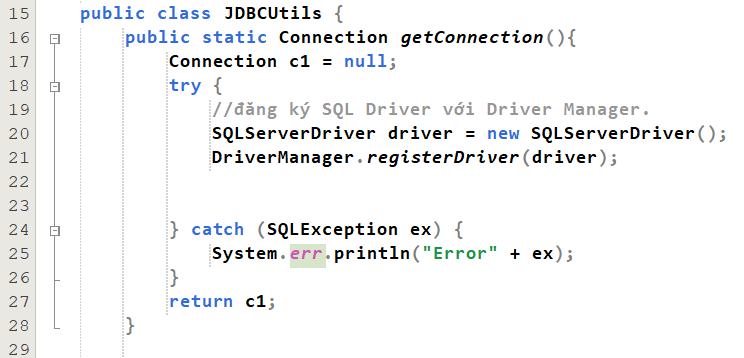
[*https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html*](https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html)

*Nói qua về class ‘Connection’ trong java: Lớp ‘Connection’ trong Java là một phần của gói java.sql và được sử dụng để thiết lập và quản lý kết nối với cơ sở dữ liệu. Một đối tượng ‘Connection’ là đại diện cho một kết nối duy nhất đến cơ sở dữ liệu.*

*Trong ‘Connection’ có một vài method cơ bản như sau:*

* + *createStatement(): Tạo một đối tượng Statement để thực thi các câu lệnh SQL.*
  + *prepareStatement(String sql): Tạo một đối tượng PreparedStatement để thực thi các câu lệnh SQL có tham số.*
  + *commit(): Xác nhận giao dịch hiện tại.*
  + *rollback(): Hoàn tác giao dịch hiện tại.*
  + *close(): Đóng kết nối với cơ sở dữ liệu.*
  + *setAutoCommit(boolean autoCommit): Bật hoặc tắt chế độ tự động xác nhận giao dịch.*

1. Theo ảnh trên thì bước đầu tiên là LoadDriver:
   * Bao gồm các bước: khai báo và đăng ký driver với DriverManager – bộ quản lý các driver.
   * Nhớ bọc nó với try-catch vì nó có thể gây lỗi SQLException, không load đc database.



1. Tiếp theo là bước ‘DriverManager.getConnection()’.
   * Với bước này cần một đường link dẫn tới database của mình. Cụ thể link đó là :

"jdbc:sqlserver://<server>:<port>;encrypt=<true/false>;trustServerCertificate=<true/false>;databaseName=<database name>;user=<user>;password=<password>"

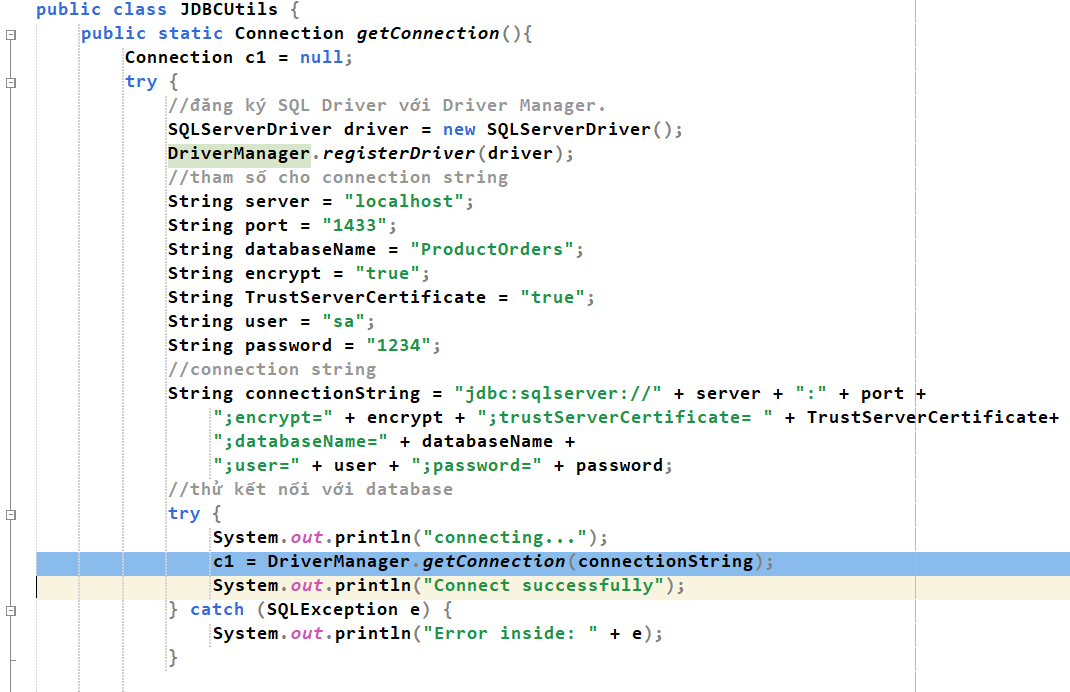
* + Các tham số cần có trong link:
    - <server> : Đây là tên hoặc địa chỉ IP của máy chủ SQL Server mà muốn kết nối đến.
    - <port>: Đây là cổng mà SQL Server đang lắng nghe, mặc định là 1433.
    - encrypt: Tham số này cho biết có muốn kết nối sử dụng mã hóa khi truyền tải giữa ứng dụng và SQL Server không.
    - trustServerCertificate: giúp xác định xem có tin tưởng vào chứng chỉ máy chủ SQL Server hay không khi sử dụng kết nối mã hóa.
    - databaseName: tên database mà đang muốn kết nối tới trong server.
    - user và password: user và password trong server của mình.

*Ngoài ra còn một số thuộc tính nữa, như: integratedSecurity, applicationName, connectTimeout, failoverPartner, …*

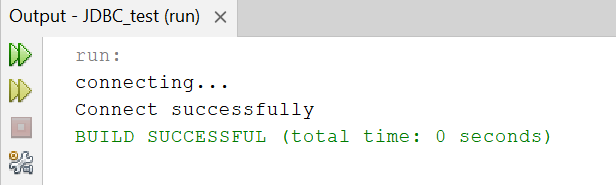
Trong chuỗi trên, để ý tới 2 thuộc tính: encrypt và trustServerCertificate. Khi đặt ‘encrypt’ là true, nghĩa là phải dùng phần mềm bên thứ 3 để mã hóa dữ liệu cho kết nối. Nếu như dùng nó thì có 2 lựa chọn: tin tưởng nó (trustServerCertificate= true), hoặc không tin nó và kiểm tra chứng chỉ máy chủ - cái cung cấp phần mềm thứ 3 để mã hóa kết nối. Trong trường hợp làm việc với localhost, nếu dùng ‘trustServerCertificate= false’ thì máy xảy ra lỗi không thể thiết lập kết nối an toàn (chưa hiểu vì sao). Nên nếu dùng ‘encrypt= true’ thì cứ tin tưởng và chọn ‘trustServerCertificate= true’.

* + Sau khi có connection string, tiếp tục thực hiện ‘DriverManager.getConnection()’.

Trong DriverManager có phương thức ‘getConnection(String connectionString)’, nên cứ thế gọi. Code sẽ trông như thế này:

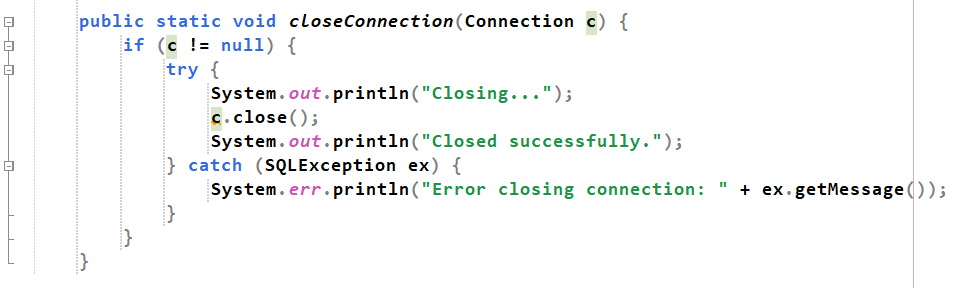


* + Nếu như kết nối thành công thì sẽ hiển thị “Connect successfully”. (tự tạo class khác có main mà test).

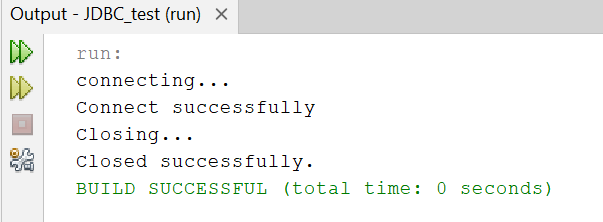


Mở được thì đóng được: viết hàm cuối – closeConnection.

* + Đóng thì chỉ có mỗi method close() của kết nối.



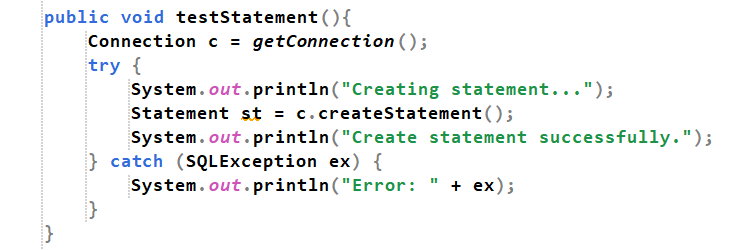
Kết quả nếu đóng thành công:



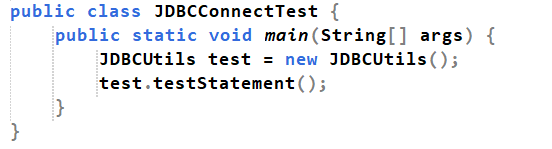
1. Sau khi hết bước 4, đến bước tiếp theo: làm việc với Statement.
   * Statement là một interface đại diện cho một câu lệnh SQL mà ứng dụng Java có thể gửi đến cơ sở dữ liệu để thực thi. Đối tượng Statement được sử dụng để thực hiện các truy vấn SQL, cập nhật dữ liệu, và các thao tác khác trên cơ sở dữ liệu.
   * *Trong interface ‘Statement’ có một vài câu lệnh quan trọng sau:* 
     + *boolean* ***executeQuery****(String sql) : thực hiện các câu lệnh SQL dạng ‘SELECT’ và trả về một đối tượng ResultSet chứa kết quả truy vấn.*
     + *int* ***executeUpdate****(String sql) : thực hiện các câu lệnh SQL dạng INSERT, UPDATE, DELETE hoặc các câu lệnh SQL khác không trả về kết quả. Phương thức này trả về số lượng hàng bị ảnh hưởng bởi câu lệnh SQL.*
     + *boolean* ***execute****(String sql) : thực hiện những câu lệnh phức tạp, những câu lệnh ghép hay StoreProcedures. Trả về true nếu kết quả đầu tiên là một ResultSet, false nếu là một số nguyên hoặc không có kết quả.*
     + *ResultSet* ***getResultSet****() : dùng để lấy ResultSet chứa kết quả truy vấn cuối cùng nếu câu lệnh SQL trả về một ResultSet.*
     + *int* ***getUpdateCount****() : dùng để lấy số lượng hàng bị ảnh hưởng bởi câu lệnh SQL cuối cùng nếu câu lệnh không trả về ResultSet.*
     + *void* ***close****() : đóng đối tượng Statement và giải phóng các tài nguyên liên quan.*

*Còn nhiều method lắm nhưng trong khuôn khổ bài giới thiệu chỉ có thế này. Đọc thêm:* [*https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html*](https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html)

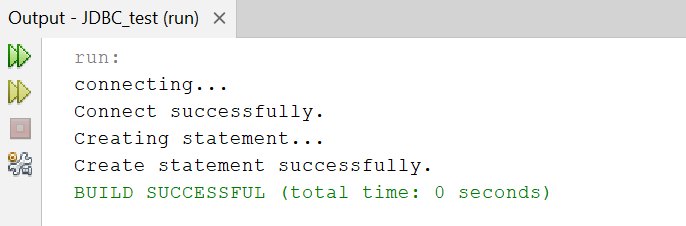
* + Đầu tiên tạo một Statement đã. Muốn có một Statement – đại diện cho câu lệnh thì đầu tiên phải có kết nối từ chương trình tới DB để truyền lệnh trước. Trong Connection có một method để tạo ra Statement từ nó: createStatement().
  + Như vậy code demo sẽ như sau:



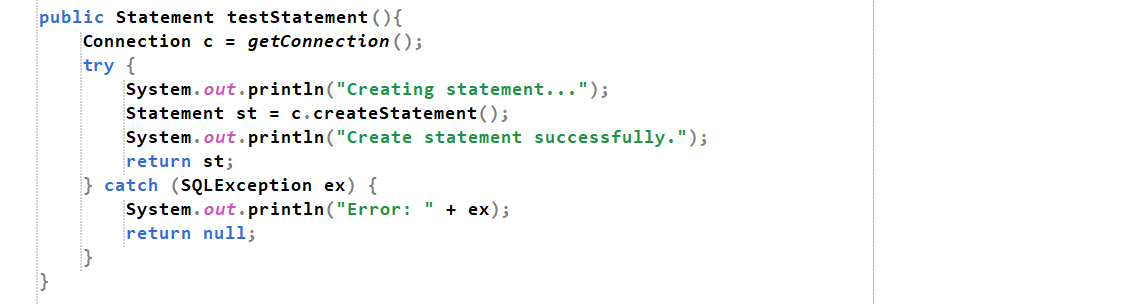
Bây giờ sẽ thử test nó:

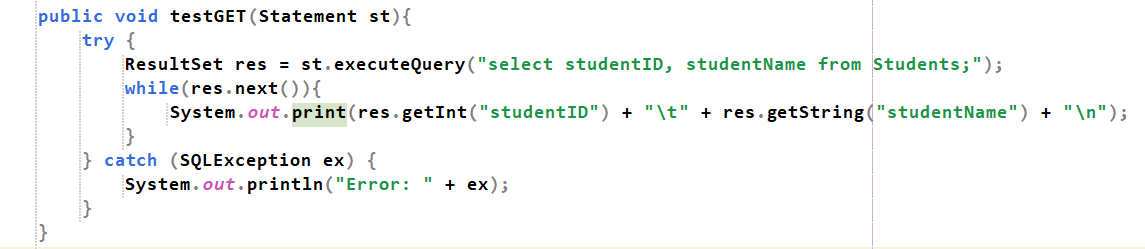


Nếu tất cả đúng như dự đoán thì kết quả sẽ như sau:

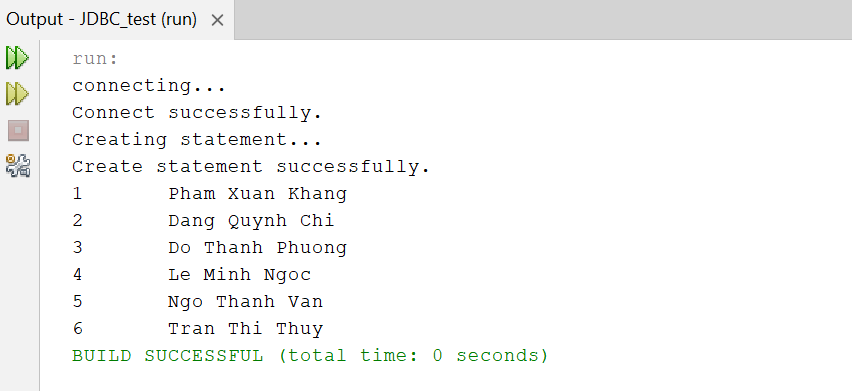


* + Tiếp theo sẽ demo vào phần thực hiện code SQL và thực hiện nó từ trong chương trình java. Từ method testStatement vừa rồi, chỉnh lại hàm để nó trả về một Statement, sau đó viết một method khác để thực hiện lệnh SQL. (đoạn này đổi database cho dễ làm, không làm với ‘ProductOrders’ nữa)





Nếu may mắn thì kết quả sẽ như sau:

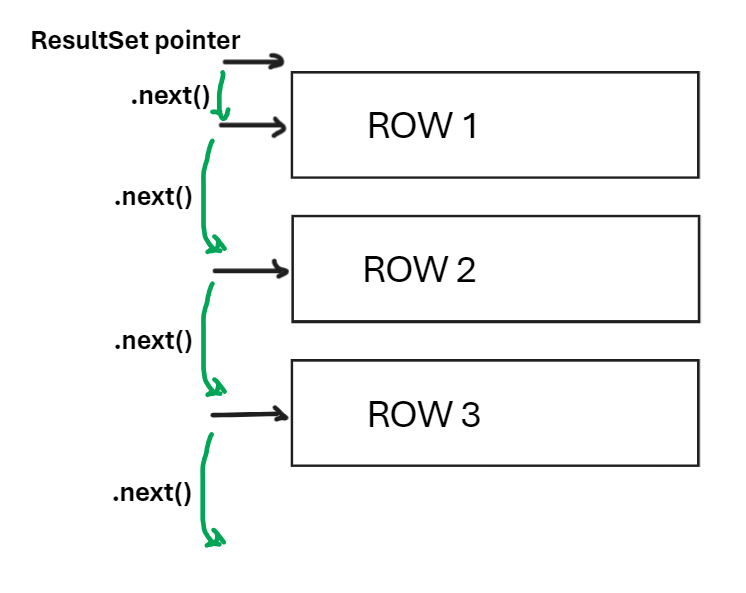


*(demo về phần executeQuery của Statement, còn phần về ResultSet nói sau)*

1. Như vậy là hết phần 11 của sơ đồ. Tiếp theo là bước 12: Process ResultSet - ứng dụng Java xử lý dữ liệu từ ResultSet.

**Trước khi làm về DAO, bây giờ sẽ nói về “ResultSet’.**

* + 1. Định nghĩa: là một interface trong Java, được sử dụng để lưu trữ và quản lý kết quả truy vấn từ cơ sở dữ liệu (database) khi thực hiện các lệnh SELECT thông qua JDBC.
    2. Hàng trong một ResultSet hoạt động tương tự như con trỏ xen hàng:



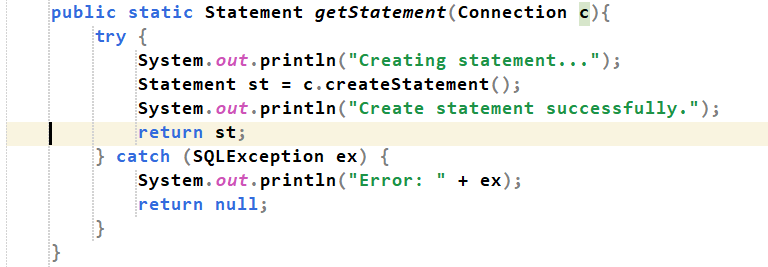
* + 1. Các method cơ bản:
       1. **next**(): Di chuyển con trỏ đến hàng tiếp theo trong ResultSet. Trả về true nếu có hàng tiếp theo, ngược lại trả về false. Tương tự, có **previous**().
       2. **getString**(int columnIndex) hoặc **getString**(String columnLabel): Lấy giá trị của cột chỉ định dưới dạng String. Nhưng lấy theo Label (tên cột) tốt hơn là lấy theo ID (vì tiện ích). Ngoài ra cũng có **getNString**().
       3. **getInt**(int columnIndex) hoặc **getInt**(String columnLabel): Lấy giá trị của cột chỉ định dưới dạng int. Tương tự thì Double, Float, Long cũng có.
       4. **getDate**(int columnIndex) hoặc **getDate**(String columnLabel): lấy giá trị của cột chỉ định dưới dạng **java.sql.Date**. Ngoài ra cũng có **getTime**.
       5. **absolute**(int row): Di chuyển con trỏ đến hàng xác định.
       6. **beforeFirst**(): Đặt con trỏ trước hàng đầu tiên.
       7. **afterLast**(): Đặt con trỏ sau hàng cuối cùng.
       8. **clearWarnings**(): xóa cảnh báo (không tác dụng mấy nhưng đẹp).
       9. **close**(): đóng kết nối, giải phóng bộ nhớ.
       10. **Ngoài ra có các method cập nhật.**
    2. Một ResultSet được trả về nhờ câu lệnh được thực hiện từ Statement. Sau khi lấy được ResultSet, chúng ta bắt đầu tương tác với tập đó qua code Java.

Vì mỗi lần mở Statement, từ Statement đó trả về một ResultSet nên ResultSet chỉ tồn tại trong thời gian Statement sinh ra nó chưa đóng. Vì vậy ResultSet có tính chất tạm thời và tham chiếu trực tiếp tới DB. **Mọi thay đổi với ResultSet trên code java bằng method update hay delete ResultSet sẽ tạo ra thay đổi trực tiếp lên DB**.

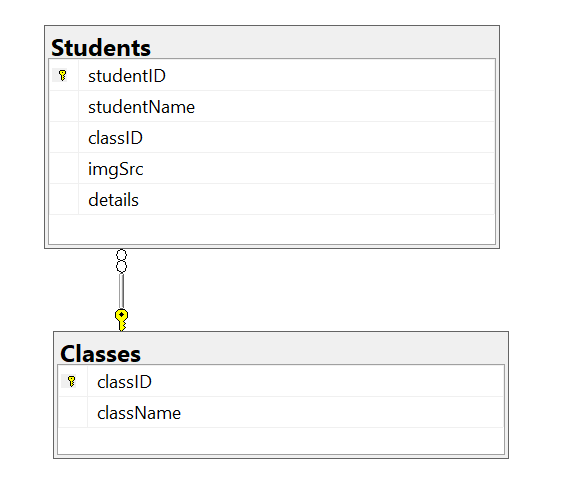
*Đọc thêm:* [*https://chatgpt.com/share/d41ca2c8-6a20-4726-997d-ac8fbcf79c4d*](https://chatgpt.com/share/d41ca2c8-6a20-4726-997d-ac8fbcf79c4d)

**Sau đây sẽ là code cụ thể.**

* + Đầu tiên sửa lại hàm ‘getStatement’ để nó return về một Statement:



* + Sau đó bắt đầu đi xây dựng mô hình MVC cho chương trình. Bắt đầu bằng việc xây dựng hệ thống Data Access Object (DAO) và hệ thống Models.
    - Nhìn vào database thì có 2 bảng:

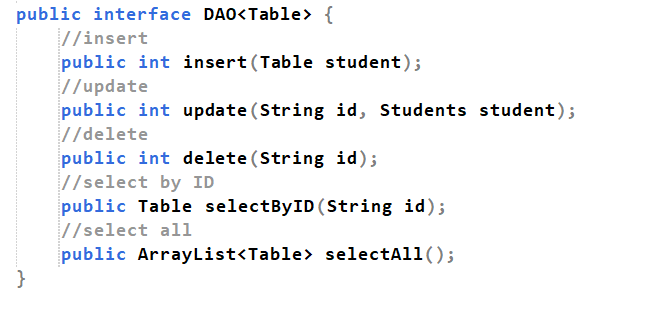


Như vậy sẽ có 2 class Students và Classes. Vì mỗi lần tương tác với database đều phải tương tác với một/nhiều instance của một/nhiều class, nên mỗi class sẽ có các phương thức để lấy/tương tác với instance (get, post, update, delete). Như thế thì cách tốt nhất là tạo interface DAO để có method tổng quát tương tác với các class, rồi sau đó vào từng class sẽ custom nó sau.

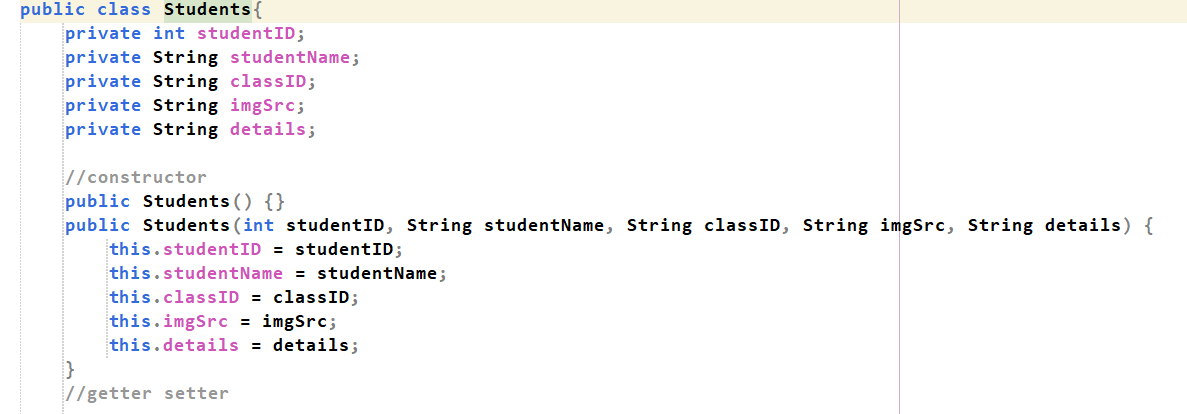
*(nên nhớ phần model của MVC chứa các class tượng trưng cho thực thể, còn trong package ‘database’ chứa các class DAO – để tương tác với database.*

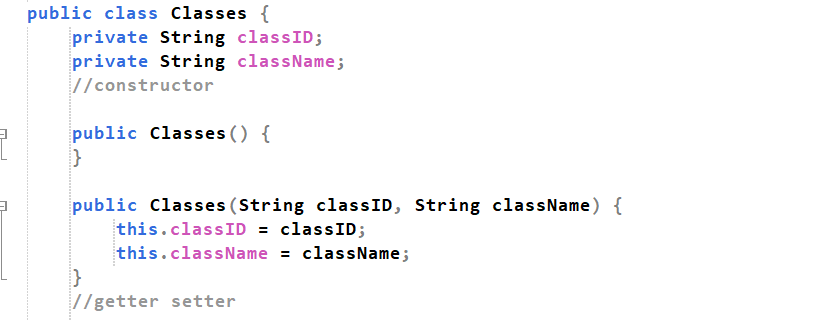
*Mỗi class DAO ứng với một class trong model - ứng với một thực thể).*

* + Dựng code của interface DAO sẽ phải có những phương thức cơ bản mà tất cả các bảng đều phải có. Cụ thể là 4 lệnh cơ bản select, insert, update, delete. Nên để kiểu dữ liệu truyền vào cho Interface là Generics để làm việc được với nhiều thực thể khác nhau.



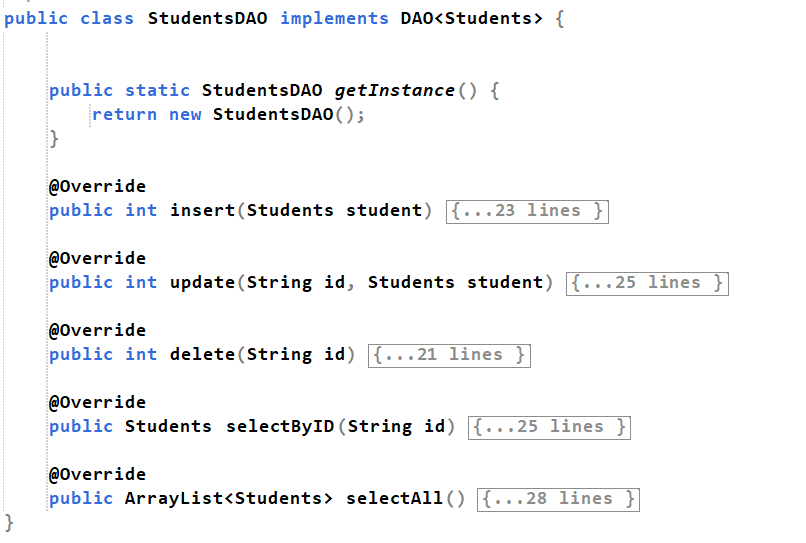
* + Như vậy đã có interface rồi, tiếp theo cần Table để truyền vào cho nó. Trong trường hợp này có 2 table như đã nói ở trên: Students và Classes. Các class này viết đơn giản, hiện tại chưa có chức năng gì thêm, chỉ có thuộc tính và getter setter constructor.





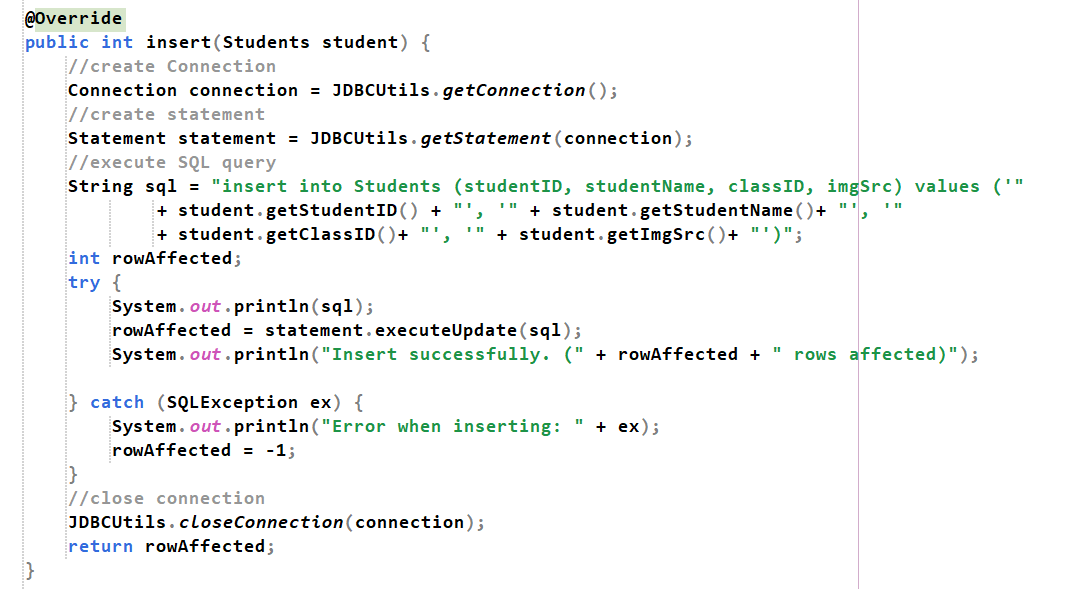
*Đoạn thiếu trong ảnh code mẫu chỉ là getter setter. Hai class này đều nằm trong models.*

* + Sau khi có cả interface và tham số cụ thể, bây giờ đi xây dựng các lớp DAO.
  + Đầu tiên là với Students, lớp StudentsDAO chuyên dùng để tương tác với database để thao tác với bảng Students trong Database.

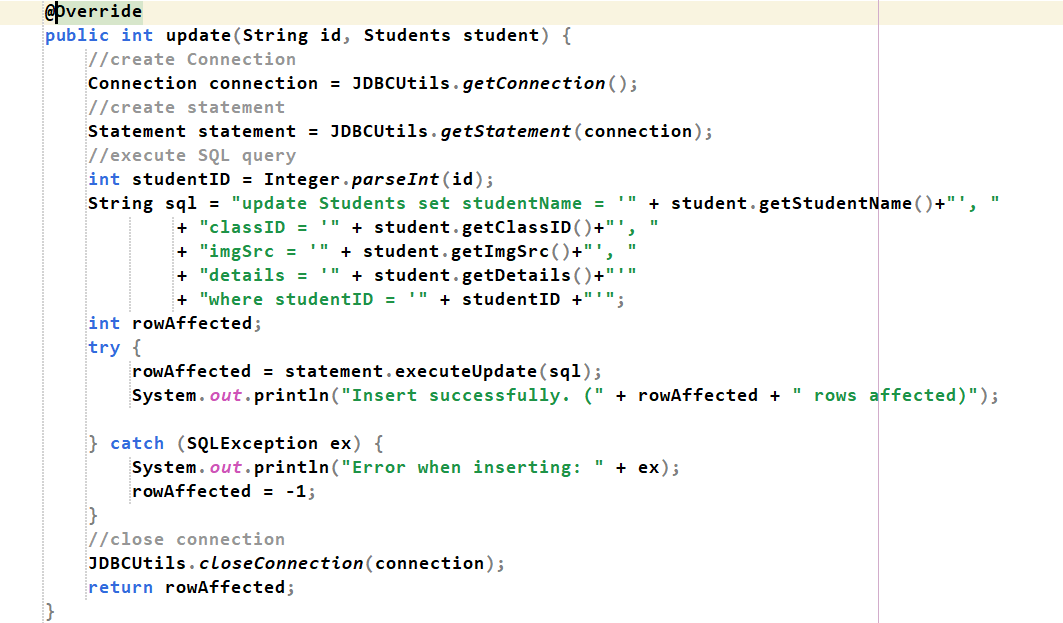


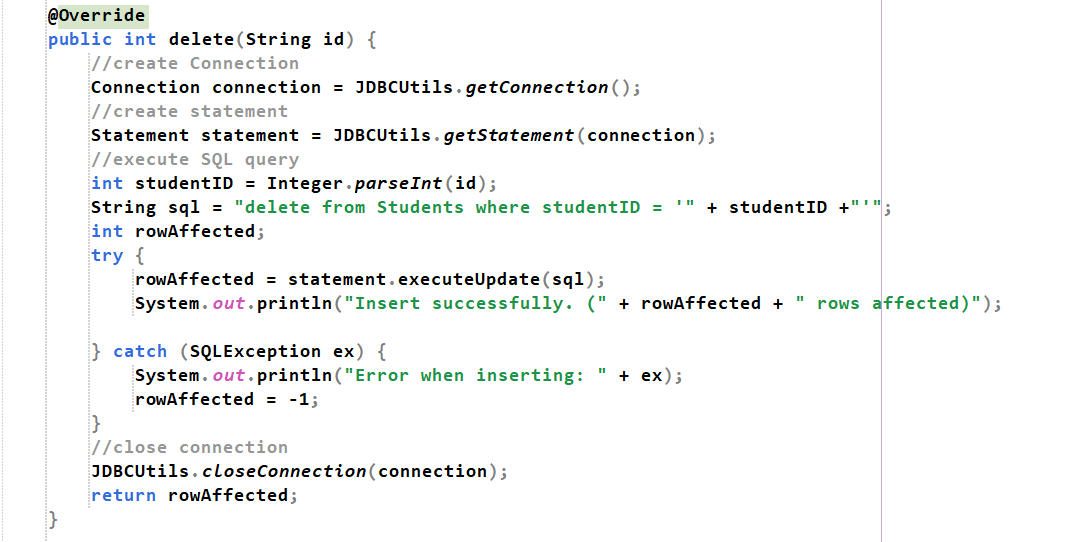
*+1 mẹo vặt: lấy instance của một method bằng cách* ***return new ClassName().***  *Nếu như không muốn viết method static hoặc không được phép dùng Static thì có cách này thay thế.*

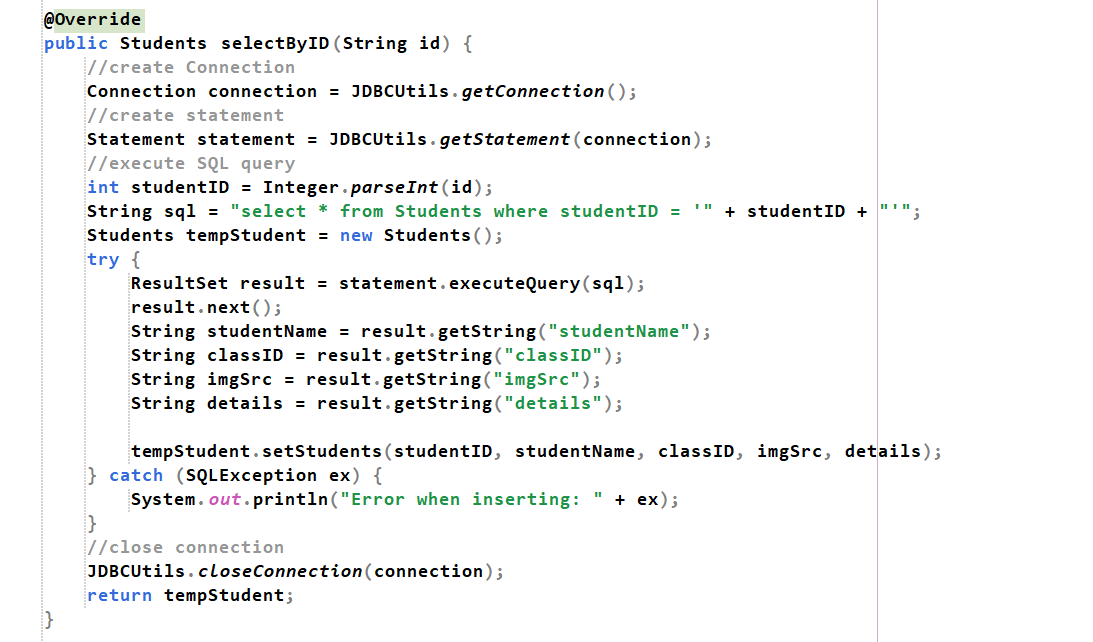
* + Chi tiết từng method trong class StudentsDAO:
    - Luôn luôn gồm 4 bước: lấy Statement, tạo query từ Statement, thực thi kết quả được trả về từ query đó, sau đó đóng kết nối.

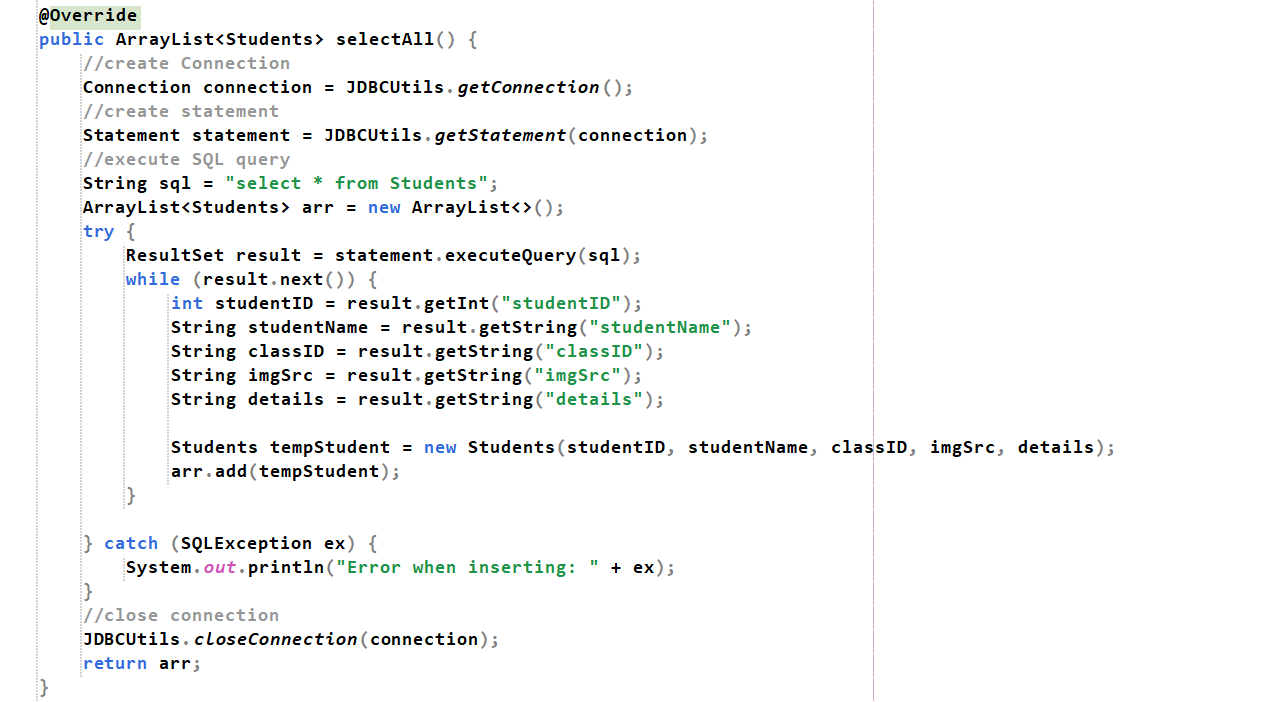


Tương tự cho các hàm khác:



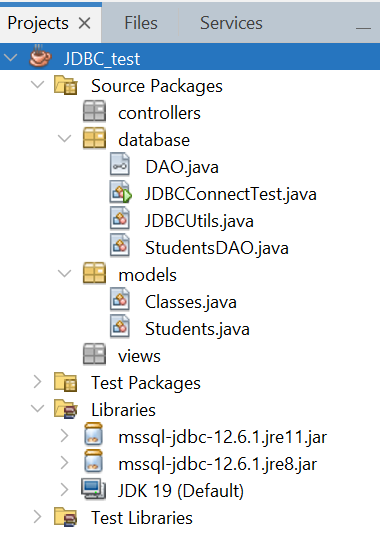




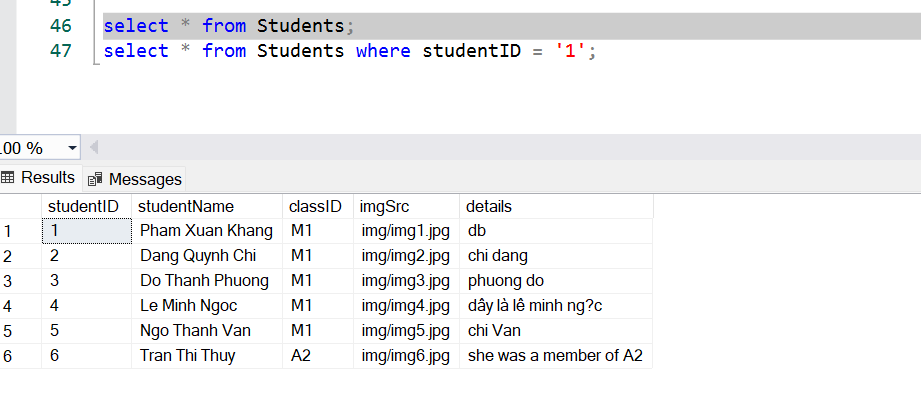


Đó là tất cả method của class DAO với thực thể Students. Với Classes thì cũng tương tự thế.

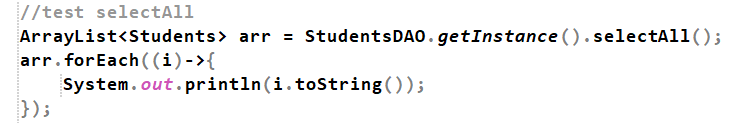
Phân bổ các file trong chương trình sẽ như sau (ngoại trừ cái ‘JDBCConnectTest’ ra thì tất cả phải ở đúng như vậy mới là MVC).



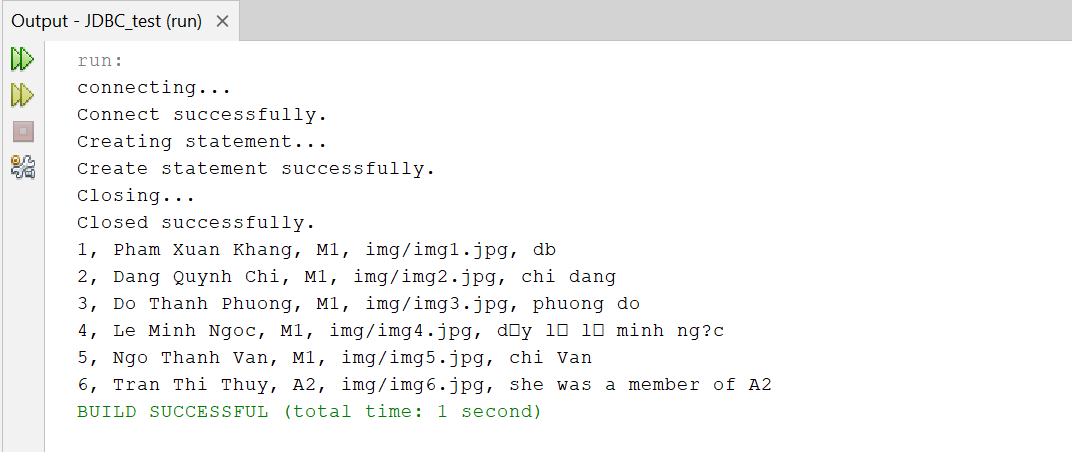
* + Kết quả:
    - Nếu chạy trong sql thì sẽ cho ra kết quả như sau:



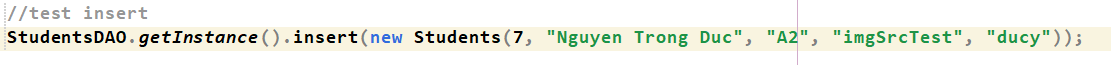
* + - Còn với chương trình, lệnh ‘selectAll’:



Nếu may mắn sẽ cho ra kết quả:

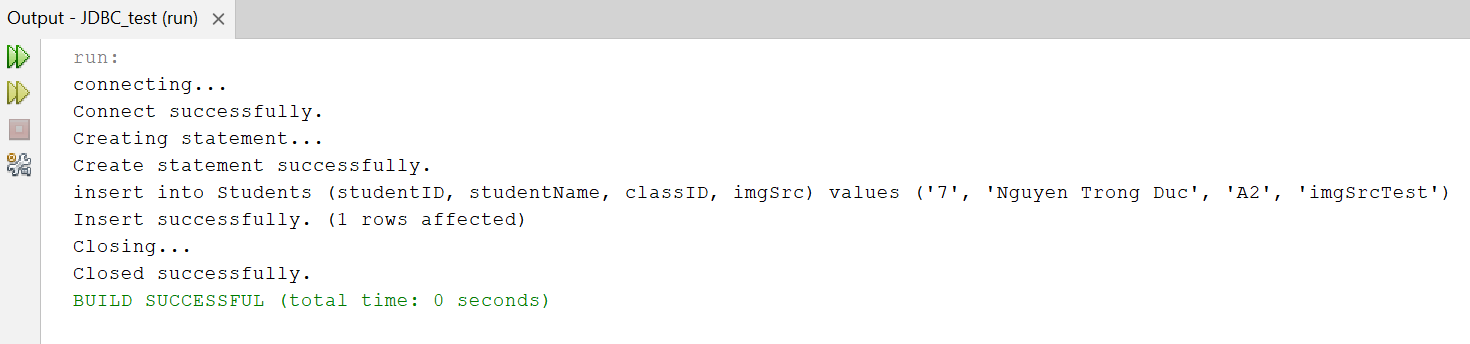


Còn đây là thử test insert (giả sử như thông tin nhập vào là hoàn toàn đúng cú pháp, ràng buộc trong DB đã tạo)

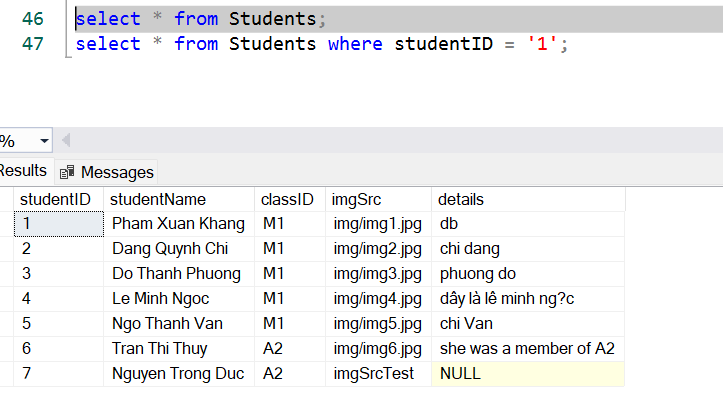


*(chỗ này hơi sai thực tế 1 chút nhưng tôi lười thêm dữ liệu vào bảng Classes để có thêm lớp Hóa)*

Nếu may mắn sẽ được kết quả sau:



Sau đó thử kiểm tra lại trong database:



(Chỗ này tôi quên mất viết thêm details cho method insert, nhưng nó vẫn hoạt động đúng như những gì có thể dự đoán từ code)

### **Các lưu ý thêm cho Statement**

* Lưu ý đầu tiên là **LUÔN ĐÓNG CÁC KẾT NỐI** sau khi thực thi xong câu lệnh cần thiết. Một database tùy vào hệ quản trị cơ sở dữ liệu mà có thể chịu được số lượng kết nối khác nhau, nhưng chắc chắn càng nhiều càng lag. Nên tối ưu hóa câu lệnh query và giảm bớt số lượng kết nối đồng thời 1 lúc với DB. (đấy là chưa tính đến Transaction – phiên làm việc).
* SQL injection:
  + Vấn đề xảy ra khi người dùng nhập một vài ký tự ‘lạ’ có chủ đích nhằm làm thay đổi kết cấu cơ bản của một câu lệnh sql.

VD: `*select username, details from users\_table where username = ‘${****người\_dùng\_nhập\_tên\_vào\_đây****}’`*

Thế nếu như người dùng nhập tên mình là: “abc’ OR 1=1; --”

* Thì câu lệnh sql sẽ thành: `*select username, details from users\_table where username = ‘abc’ OR 1=1;* --*`*
* Nhìn là thấy hậu quả: tất cả thông tin trong bảng đó được hiển thị. SQL INJECTION
  + Giải quyết: không thể để người dùng có cơ hội tương tác trực tiếp với câu lệnh SQL như trên được. Câu lệnh SQL cần được hoàn thiện trước, rồi sau đó người dùng chỉ được phép thay giá trị vào, sau khi đã qua kiểm duyệt.
  + Về vấn đề đó đã có **PreparedStatement** – giải quyết được SQLi thay cho **Statement**, vốn gây ra lỗi SQLi.
  + Cách dùng **PreparedStatement**:
    - PreparedStatement có các method để set giá trị cho các tham số được truyền vào. Rõ hơn có thể đọc docs:

<https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html>

* + - Cách dùng:

1. Viết các câu lệnh query và tham số truyền vào được thay bằng dấu “?”, và **được đánh thứ tự từ 1.**

VD: String sql = "INSERT INTO yourTable (column1, column2) VALUES (?, ?)";

1. Sau đó “thực hiện trước” – prepare cho câu lệnh đó:

VD: PreparedStatement preparedStatement = connection.preparedStatement(sql);

1. Tiếp tục set các giá trị cho các dấu hỏi chấm mình vừa viết, dựa vào thứ tự được đánh số và dữ liệu người dùng nhập vào.

VD: preparedStatement.setString(1, “value1”);

preparedStatement.setString(2, “value2”);

1. Sau cùng mới thực thi câu lệnh.

VD: int rowsAffected = preparedStatement.executeUpdate();

* + - Chi tiết hơn chút về so sánh **PreparedStatement** với **Statement**, theo docs GPT:

<https://chatgpt.com/share/6b0e984d-347b-4315-91b4-9d987fe03f85>

**HẾT**