

T08 – Conectando sem as classes do tutorial JDBC

Igor Augusto Reis Gomes – 12011BSI290 – igor.augusto@ufu.br

Heitor Guimarães Da Fonseca Filho – 12011BSI203 – heitor.filho@ufu.br

Obs.: o resumo mencionado no item 7 está como comentários ao longo do próprio código.

1. Inicialmente, a quantidade de bibliotecas a ser importada será menor quando comparada com as bibliotecas da classe MyQueries:
2. Para implementação do código você precisa de, pelo menos, uma classe e quatro métodos:

Tanto as bibliotecas foram importadas, quanto a estrutura dos quatro métodos foram inseridos:

```
src > StandAloneJDBCCode.java > StandAloneJDBCCode > main(String[])
1 import java.util.Properties; //Objeto genérico que armazena propriedades com usuário e senha
2 import java.sql.DriverManager; //Objeto que criará a conexão do sistema de banco de dados
3 import java.sql.Connection; //Objeto que armazenará o objeto de conexão ao banco de dados
4 import java.sql.Statement; //Objeto para disparar um comando para o SGBD
5 import java.sql.ResultSet; //Objeto que armazenará as tuplas resultantes de um comando SQL
6 import java.sql.SQLException; //Objeto para capturar eventos de erro no acesso ao banco de dados
7
8 public class StandAloneJDBCCode {
9 >
12 > public static Connection getConnection() { ...
15 > public static void myquery(Connection con) throws SQLException { ...
18 > public static void closeConnection(Connection con) { ...
20 }
21 }
```

3. O código abaixo mostra um esboço do método criador do objeto armazenador da conexão ao banco de dados. Perceba que este método retorna um objeto de conexão do tipo Connection, cuja biblioteca da classe foi incorporada no início do código.

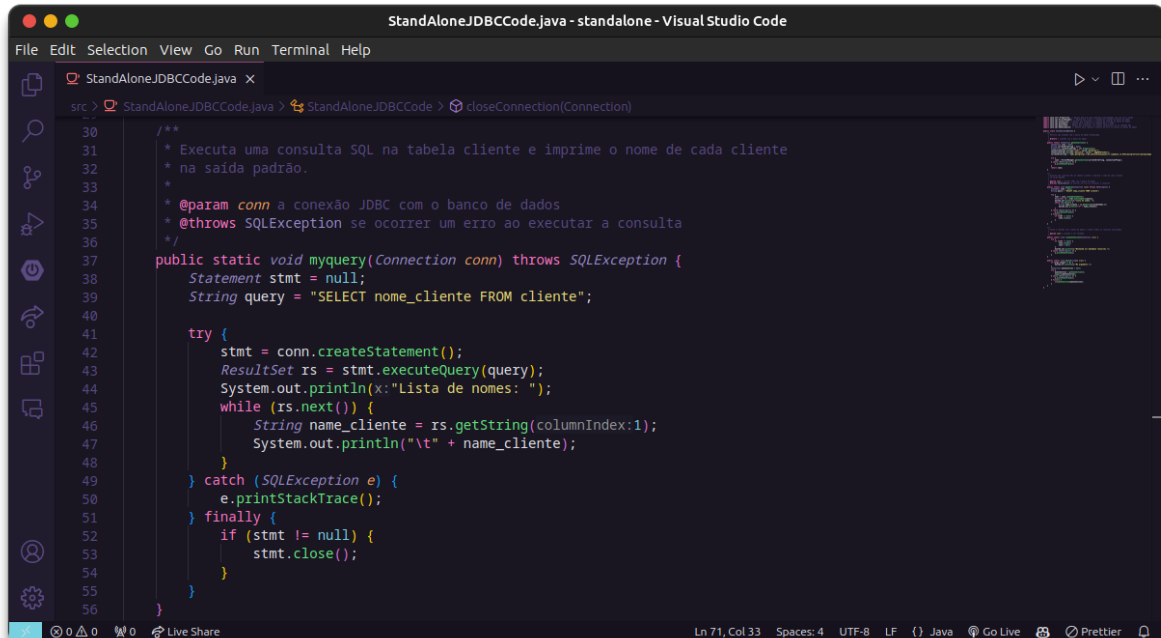
Termine de implementar este código exemplo na sua classe de consulta StandAloneJDBCCode, dentro do arquivo java de mesmo nome.

Adicionei o corpo do método e as propriedades do banco:

```
src > StandAloneJDBCCode.java > StandAloneJDBCCode > getConnection()
8 public class StandAloneJDBCCode {
9 /**
10 * Retorna uma conexão com o banco de dados PostgreSQL.
11 *
12 * @return a conexão com o banco de dados
13 */
14 public static Connection getConnection() {
15 Connection conn = null;
16 String currentUrlString = null;
17 Properties connectionProps = new Properties();
18 connectionProps.put(key:"user", value:"postgres");
19 connectionProps.put(key:"password", value:".kB0e6PAT%JS3j");
20 currentUrlString = "jdbc:postgresql://db.umsozekvkyejwaoazrf1.supabase.co:5432/postgres";
21
22 try {
23 conn = DriverManager.getConnection(currentUrlString, connectionProps);
24 } catch (SQLException e) {
25 e.printStackTrace();
26 }
27 return conn;
28 }
29 }
```

- Termine de implementar este código exemplo na sua classe de consulta StandAloneJDBCCode, dentro do arquivo java de mesmo nome, provendo um nome correto de uma tabela do seu banco de dados, bem como tratando os dados retornados no objeto do tipo ResultSet.

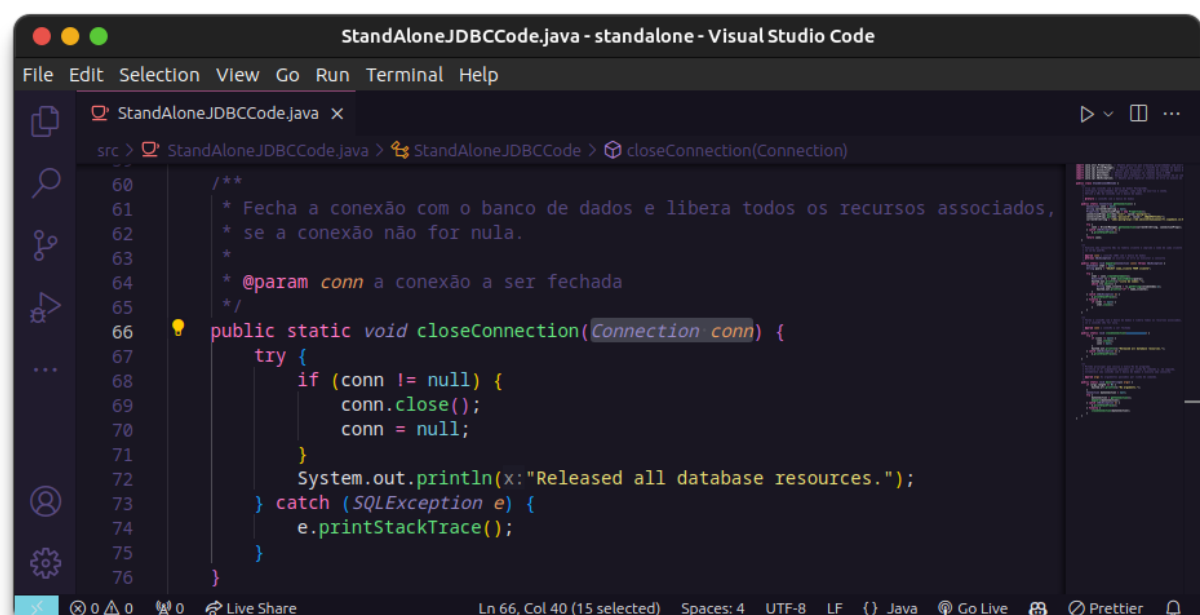
Decidi por fazer uma consulta simples na tabela cliente, que retorna todas as linhas de nome_cliente:



```
src > StandAloneJDBCCode.java > StandAloneJDBCCode > closeConnection(Connection)
30  /**
31   * Executa uma consulta SQL na tabela cliente e imprime o nome de cada cliente
32   * na saída padrão.
33   *
34   * @param conn a conexão JDBC com o banco de dados
35   * @throws SQLException se ocorrer um erro ao executar a consulta
36   */
37  public static void myquery(Connection conn) throws SQLException {
38      Statement stmt = null;
39      String query = "SELECT nome_cliente FROM cliente";
40
41      try {
42          stmt = conn.createStatement();
43          ResultSet rs = stmt.executeQuery(query);
44          System.out.println(x:"Lista de nomes: ");
45          while (rs.next()) {
46              String name_cliente = rs.getString(columnIndex:1);
47              System.out.println("\t" + name_cliente);
48          }
49      } catch (SQLException e) {
50          e.printStackTrace();
51      } finally {
52          if (stmt != null) {
53              stmt.close();
54          }
55      }
56  }
```

- O código acima possui um erro. Transcreva o código para o programa StandAloneJDBCCode.java e conserte este erro. Se tiver dificuldade em encontrar o erro então espere até a compilação do código completo do programa StandAloneJDBCCode.java e o compilador do java pode fornecer alguma informação adicional para você corrigir o erro:

O erro está na falta do argumento/parâmetro de conexão que deveria ser passado na assinatura do método: `closeConnection(Connection conn)`:



```
src > StandAloneJDBCCode.java > StandAloneJDBCCode > closeConnection(Connection)
60  /**
61   * Fecha a conexão com o banco de dados e libera todos os recursos associados,
62   * se a conexão não for nula.
63   *
64   * @param conn a conexão a ser fechada
65   */
66  public static void closeConnection(Connection conn) {
67      try {
68          if (conn != null) {
69              conn.close();
70              conn = null;
71          }
72          System.out.println(x:"Released all database resources.");
73      } catch (SQLException e) {
74          e.printStackTrace();
75      }
76  }
```

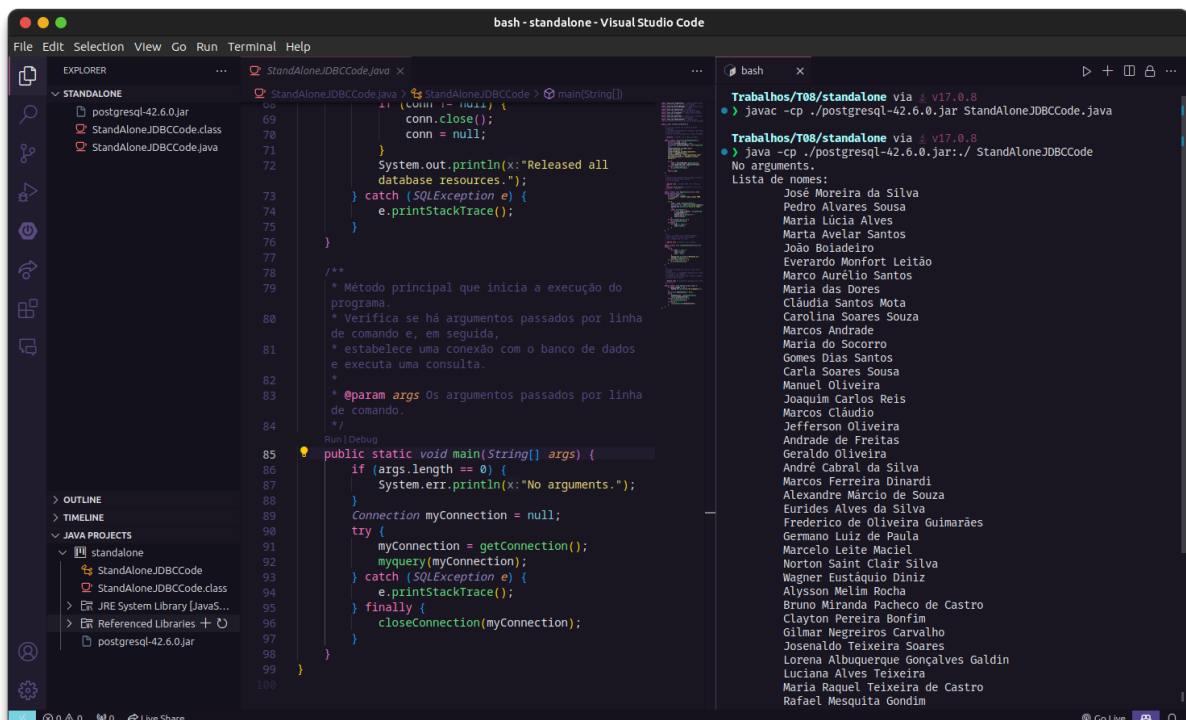
6. Na sequência, haverá a instânciação de um objeto do tipo `Connection` que inicialmente está nulo. O método `StandAloneJDBCCode.getConnection()` fará o trabalho de criar o objeto de conexão ao banco de dados. Por fim, o objeto de conexão é passado para o método deste programa que cumprirá o objetivo maior, realizar a consulta ao banco e tratar os dados como for necessário. Passe este código para nosso programa e execute-o. Ah, tem um erro neste código também, encontre-o.

O erro está dentro do bloco `finally` do `try`, no qual tenta-se encerrar a conexão, porém a método `closeConnection` espera receber um parâmetro do tipo `Connection`, no qual não é feito (devido ao erro do item anterior), assim, para que funcione basta inserir a variável como argumento da função:



```
75 public static void main(String[] args) {
76     if (args.length == 0) {
77         System.err.println(x:"No arguments.");
78     }
79     Connection myConnection = null;
80     try {
81         myConnection = getConnection();
82         myquery(myConnection);
83     } catch (SQLException e) {
84         e.printStackTrace();
85     } finally {
86         closeConnection(myConnection);
87     }
88 }
```

Compilação e execução do programa como foi solicitado, no lado direito da captura:



```
Trabalhos/T08/standalone via v17.0.8
javac -cp ./postgresql-42.6.0.jar StandAloneJDBCCode.java

Trabalhos/T08/standalone via v17.0.8
java -cp ./postgresql-42.6.0.jar:./ StandAloneJDBCCode
No arguments.

Lista de nomes:
José Moreira da Silva
Pedro Alvares Sousa
Maria Lúcia Alves
Marta Avelar Santos
João Boiadeiro
Everardo Monfort Leitão
Marco Aurélio Santos
Maria das Dores
Cláudia Santos Mota
Carolina Soares Souza
Marcos Andrade
Maria do Socorro
Gomes Dias Santos
Carla Soares Sousa
Manuel Oliveira
Joaquim Carlos Reis
Marcos Cláudio
Jefferson Oliveira
Andrade de Freitas
Geraldo Oliveira
Andre Cabral da Silva
Marcos Ferreira Dinardi
Alexandre Márcio de Souza
Eurides Alves da Silva
Frederico de Oliveira Guimarães
Germano Luiz de Paula
Marcelo Leite Maciel
Norton Saint Clair Silva
Wagner Eustáquio Diniz
Alysson Malin Rocha
Bruno Miranda Pacheco de Castro
Clayton Pereira Bonfim
Gilmar Negreiros Carvalho
Josenaldo Teixeira Soares
Lorena Albuquerque Gonçalves Galdin
Luciano Alves Teixeira
Maria Raquel Teixeira de Castro
Rafael Mesquita Gondim
```

Além disso, com intuito de simplificação, descobri que é possível compilar e executar o programa sem a necessidade de digitar manualmente o caminho da classe e do driver do postgres, dado que o VSCode cria um arquivo settings.json no qual é possível colocar diretamente o caminho das bibliotecas referenciadas, neste caso, o driver que coloquei dentro da pasta “lib”. Como é possível observar abaixo, na janela do canto inferior esquerdo com o arquivo .json, e a com a execução direta na direita.

