Sistemas de Bancos de Dados
Aula 12 – Conectando sem as classes do tutorial JDBC

O tutorial do JDBC possui diversas classes e arquivos que exploram os recursos da biblioteca de conexão a banco de dados do java de forma sistemática. Para uma conexão simples (em apenas um banco de dados, um único usuário, uma consulta simples) não há necessidade em utilizar-se de todas as classes desse tutorial. De fato veremos que uma única classe, com quatro métodos, atende a esta necessidade simples, ao custo da perda de flexibilidade na passagem de parâmetros e recursos mais avançados no manuseio do banco de dados. Entretanto, para aplicações simples e específicas essa alegada "perda" não chega a ser significativa, situação na qual você pode preferir por deixar o código de conexão ao banco de dados do mesmo modo da aplicação: simples. Nesta aula você criará um código mínimo para a conexão a um banco de dados gerenciado pelo PostgreSQL. Você utilizará o banco de dados IB2 criado e modificado em aulas anteriores;

1) Inicialmente, a quantidade de bibliotecas a ser importada será menor quando comparada com as bibliotecas da classe *MyQueries*:

```
import java.util.Properties; //Objeto genérico que armazena propriedades com usuário e senha
import java.sql.DriverManager; //Objeto que criará a conexão do sistema de banco de dados
import java.sql.Connection;  //Objeto que armazenará o objeto de conexão ao banco de dados
import java.sql.Statement; //Objeto para disparar um comando para o SGBD
import java.sql.ResultSet;  //Objeto que armazenará as tuplas resultantes de um comando SQL
import java.sql.SQLException; //Objeto para capturar eventos de erro no acesso ao banco de dados
```

As bibliotecas acima estão listadas na ordem em que geralmente são utilizadas em um sistema de banco de dados;

2) Para implementação do código você precisa de, pelo menos, uma classe e quatro métodos:

```
public class StandAloneJDBCCode {

	public static Connection getConnection(){
	...
	}

	public static void myquery(Connection con) throws SQLException {
	...
	}

	public static void closeConnection(Connection con) {
	...
	}

	public static void main(String[] args) {
	...
	}
}
```

Neste exemplo o nome da classe é *StandAloneJDBCCode*. O método main não é obrigatório porque nem sempre uma classe é acionada pelo sistema operacional para execução (assim com demonstrado), mas pode ser chamado apenas um método de sua classe. Como exemplo, *Integer.parseInt("1.000")* chama o método *parseInt* da classe para converter uma string para um número inteiro e não possui um método *main*. Os outros três métodos (*getConnection, closeConnection, myquery*) poderiam todos serem codificados dentro do método *main*, ou qualquer outro, sem necessariamente estarem contidos em métodos próprios. Porém, para uma melhor

organização do código optei por ao menos esta estrutura mínima de métodos e uma classe.

Crie um arquivo denominado *StandAloneJDBCCode.java* para armazenar o código fonte desta atividade, assim como sugerido pelo item 2..

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.

```
public static Connection getConnection(){
        Connection con = null;
        String currentUrlString = null;
        Properties connectionProps = new Properties();

        connectionProps.put("user", "usuário");
        connectionProps.put("password", "senha");

        currentUrlString = "jdbc:dbms://servidor:porta/nome do banco de dados";
        //atenção para os símbolos de barra "/" na linha acima, não confunda com a letra l
        try {
                con = DriverManager.getConnection(currentUrlString, connectionProps);
        } catch (SQLException e) {
                e.printStackTrace();
        }
        return con;
}
```

O nome do método *getConnection* é utilizado em outras classes do JDBC, assim como neste próprio código de exemplo. *GetConnection* também é o nome do método chamado da classe *DriverManager*. O método *getConnection* de nossa classe em desenvolvimento faz uso de um objeto criado a partir da classe *Properties*, de uso genérico na linguagem java, para armazenar os dados de usuário e senha de conexão ao banco de dados. Este objeto de propriedades foi denominado *connectionProps* e será passado para o método getConnection da classe *DriverManager*, acompanhado do endereço de conexão ao banco de dados: *currentUrlString*. Esta última variável específica o tipo de *driver (dbms)*, o nome/número do servidor, a porta e o banco de dados usado na conexão. O tipo de *driver* não está sendo importado pela definição do programa, ao início do código. O JDBC vai procurar pela existência de um *driver* para o *dbms* com este nome. A busca ocorrerá entre os caminhos passados no momento da compilação e execução do programa java via a variável *classpath* (*-cp*) valorada pelo programa de compilação denominado *comp*. Se tudo estiver devidamente configurado a conexão será realizada e um objeto, aqui denominado *con*, receberá a conexão devolvida como parâmetro de retorno da função *getConnection*.

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

4) Este código mostra o método que é o objetivo da existência deste programa, uma consulta ao banco de dados. Percebam que o método apenas recebe um objeto de conexão ao banco de dados, não há uma implementação de código para determinar os parâmetros de conexão (função do código do item anterior), mas apenas a preocupação de executar uma consulta nesta conexão e tratar os dados recebidos:

```
public static void myquery(Connection con) throws SQLException {
    Stat getConnection ement stmt = null;
    String query = "select * from sometable";
   try {
      stmt = congetConnection .createStatement();
      ResultSet rs = stmt.executeQuery(query);
      System.out.println("Table list: ");
      while ( rs.next() ) {
        String name   = rs.getString(1);
       ...
        System.out.println(name + ", " + ... );
      }
    } catch (SQLException e) { e.printStackTrace(); }
      finally { if (stmt != null) { stmt.close(); } }
}
```

O código inicia-se pela definição da variável *stmt*, que armazenará um objeto para execução de um comando na conexão aberta para o banco de dados. A variável *query* possui a definição do comando SQL. A partir do momento em que são tentados acessos ao banco de dados por meio da conexão aberta (objeto *com*), é necessário utilizar a cláusula *try* para capturar possíveis erros decorridos de acessos ao banco. A consulta é executada pelo método *executeQuery* do objeto *stmt* e o resultado da execução é armazenado no objeto *rs*, do tipo *ResultSet*. Na sequência, um *loop* é feito para percorrer todas as tuplas retornadas para *rs.* Neste *loop* os dados devem ser devidamente tratados de acordo com a finalidade do programa. No nosso caso, estamos apenas listando os dados na tela. Lembre-se que cada coluna de dados retornada via objeto do tipo *ResultSet* deve ser devidamente recuperado/indexado (por exemplo, armazenado em uma variável) para posterior tratamento.

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*.

5) O código abaixo tem o objetivo de finalizar a conexão criado com o banco de dados. Tal hábito é saudável visto que um sistema de banco de dados pode criar diversas conexões simultâneas a um banco de dados com potencial de prejudicar o desempenho de um servidor de banco de dados, bem como limitar a quantidade de conexões para outros usuários/sistemas:

```
 public static void closeConnection() {
   try {
     if (con != null) {
       con.close(); con = null;
     }
   System.out.println("Released all database resources.");
   } catch (SQLException e) { e.printStackTrace(); }
 }
```

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.

6) Por último, o código abaixo é o método executável deste programa.:

```
  public static void main(String[] args) {

    if (args.length == 0) {
       System.err.println("No arguments.");
    }

    Connection myConnection = null;
   try {
       myConnection = StandAloneJDBCCode.getConnection();
       StandAloneJDBCCode.myquery(myConnection);
    } catch (SQLException e) {
       e.printStackTrace();
    } finally {
       StandAloneJDBCCode.closeConnection();
    }
  }
}
```

Este é o ponto de entrada para a execução do programa *StandAloneJDBCCode.java*. A execução será iniciada pela verificação se houve algum parâmetro passado para o programa java no momento de seu acionamento na linha de comando. É muito comum usarmos esse tipo de verificação para identificar, por exemplo, se o usuário do programa passou a quantidade de parâmetros corretos para o programa ou ainda verificar se os valores dos parâmetros passados, ainda que na quantidade correta, estão dentro das faixas de valores que o programa foi preparado para lidar. Neste caso específico estamos apenas exibindo na tela uma mensagem que alerta para a ausência de parâmetros sem prejuízos ao nosso programa uma vez que nenhum parâmetro é exigido. Todos os parâmetros (por exemplo, de conexão) estão *hard coded* com o código fonte do programa.

Na sequência, haverá a instanciação 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.

Comando para compilar (substitua ***caminho do driver*** pelo caminho em sua máquina):

*javac -cp "**caminho do driver**/postgresql-42.2.4.jar" StandAloneJDBCCode.java*

Comando para executar:

*java -cp "**caminho do driver**/postgresql-42.2.4.jar**:./**" StandAloneJDBCCode*

Atenção para o símbolo **:./** ao final da *string* de *-cp* que significa “acrescente o caminho corrente à lista de locais onde pode existir alguma biblioteca necessária para o programa conseguir ser executado com sucesso”.

7) Entreguem o código fonte gerado, em formato java e texto puro, realizando alguma consulta ao banco IB2, além de um resumo sobre o que cada parte do código atua para a correta execução do programa.