Banco de Dados com Java

Programação Orientada a Objetos - Aula 08

Professor: Hamilton Machiti da Costa

JDBC

- Significa Java Database Connectivity, isto é, Conectividade de Banco de Dados Java.
- É a tecnologia usada para que programas Java clientes acessem servidores de bancos de dados.
- Consiste em um conjunto de classes e interfaces (API) implementadas pelos fabricantes de banco de dados pertencentes ao pacote java.sql
- Esta implementação é reunida em um arquivo .jar pelo fabricante denominado **driver** de JDBC.

Receita de Banco de Dados com Java

“Ingredientes”

- um driver de JDBC
- uma String de Conexão
- um objeto Connection
- um objeto PreparedStatement
- um objeto ResultSet

Modo de Preparo

- Junte tudo dentro de uma classe Java
- Divida direitinho em métodos CRUD: inserir, deletar, atualizar e consultar.
- Crie uma classe só para pegar a conexão.
- Trate as transações quando necessário.
- Prepare-se para lidar com exceções a toda hora.

Tratamento de Exceções

- Problemas acontecem… e o Java tem um mecanismo para lidar com isso e evitar que um programa simplesmente caia: são os blocos **try-catch-finally** e as **exceptions**.

- Existem situações sobre as quais o programador tem total controle, como fazer um loop corretamente ou evitar divisão por zero. Neste caso, é opcional para o programador usar um try-catch.

- Existem outras situações, entretanto, sobre as quais o programador não tem nenhum controle, como por exemplo, o usuário escolhe gravar em um arquivo em um diretório no qual ele não tem permissão, ou o usuário tenta inserir um cliente no banco de dados que já existe, ou a rede caiu e o programa Java não conseguiu conexão com a internet. Nestes casos, o uso do try-catch é obrigatório.

```
try{

//coloque aqui alguma coisa que possa dar errado

} catch(Exception e){

//coloque aqui o que fazer se der errado

} finally {

//coloque aqui algo que queira fazer se der certou ou errado
//como fechar uma conexao com o banco, por exemplo

}
```

Try with resource

- A partir do java 1.7 existe um tipo de try-catch que fecha automaticamente recursos que foram abertos no try e que precisem ser fechados, com arquivos ou conexões no banco de dados.

- A diferença do anterior é que você abra e fecha parênteses depois do try e, dentro destes parênteses, faz a abertura dos recursos questão automaticamente fechados.

```
try(/*abra aqui recursos que irao precisar ser fechados,
      como conexoes*/){

   //aqui coloque comandos que possam gerar exceções

} catch (Exception e){

   //faça o tratamento da exceção aqui

}
```

Exceções

- A classe Exception é superclasse de todas as exceptions. Qualquer exceção que aconteça é pega por ela.
- Porém, existem exceções mais específicas, usadas para dar tratamento específico para o problema.
- Neste caso, podemos simplesmente repetir o bloco **catch** várias vezes, por ordem de especificidade. As mais específicas antes, as mais genéricas depois, terminando com a Exception.
- Podemos também usar um **try-multicatch** se o tratamento dado a diversas exceções for o mesmo.
- Vale lembrar que o **finally** não é obrigatório, apenas um **try** e um ou mais **catch**s.
- Se você não quiser tratar a exceção você pode lançá-la por meio do comando **throws**. (Exemplo mais adiante).

```java
try{

//coloque aqui alguma coisa que possa gerar uma SQLException e
//uma IOException, senao nao compila

} catch(SQLException e){

//coloque aqui o que fazer se der uma SQLException

} catch(IOException e){

//coloque aqui o que fazer se der uma IOException

} catch(Exception e){

//coloque aqui o que fazer se der outro erro

}
```

Obs: nunca irá acontecer mais de uma exception ao mesmo tempo, porque quando uma acontece a execução do bloco try é imediatamente interrompida e passa para o bloco catch correspondente.

Obs2: Se você colocar o catch da Exception primeiro, não compila porque as outras duas exceptions nunca irão ser alcançadas, pois a Exception irá pegá-las antes.

```
import javax.swing.JOptionPane;
public class Excecoes{
   public static void main(String[] args){
      String s = JOptionPane.showInputDialog("Valor");

      try{

         int x = Integer.parseInt(s.substring(2,3));

      } catch (NullPointerException | NumberFormatException | StringIndexOutOfBoundsException e){

         throw new RuntimeException(e);

      }
   }
}
```

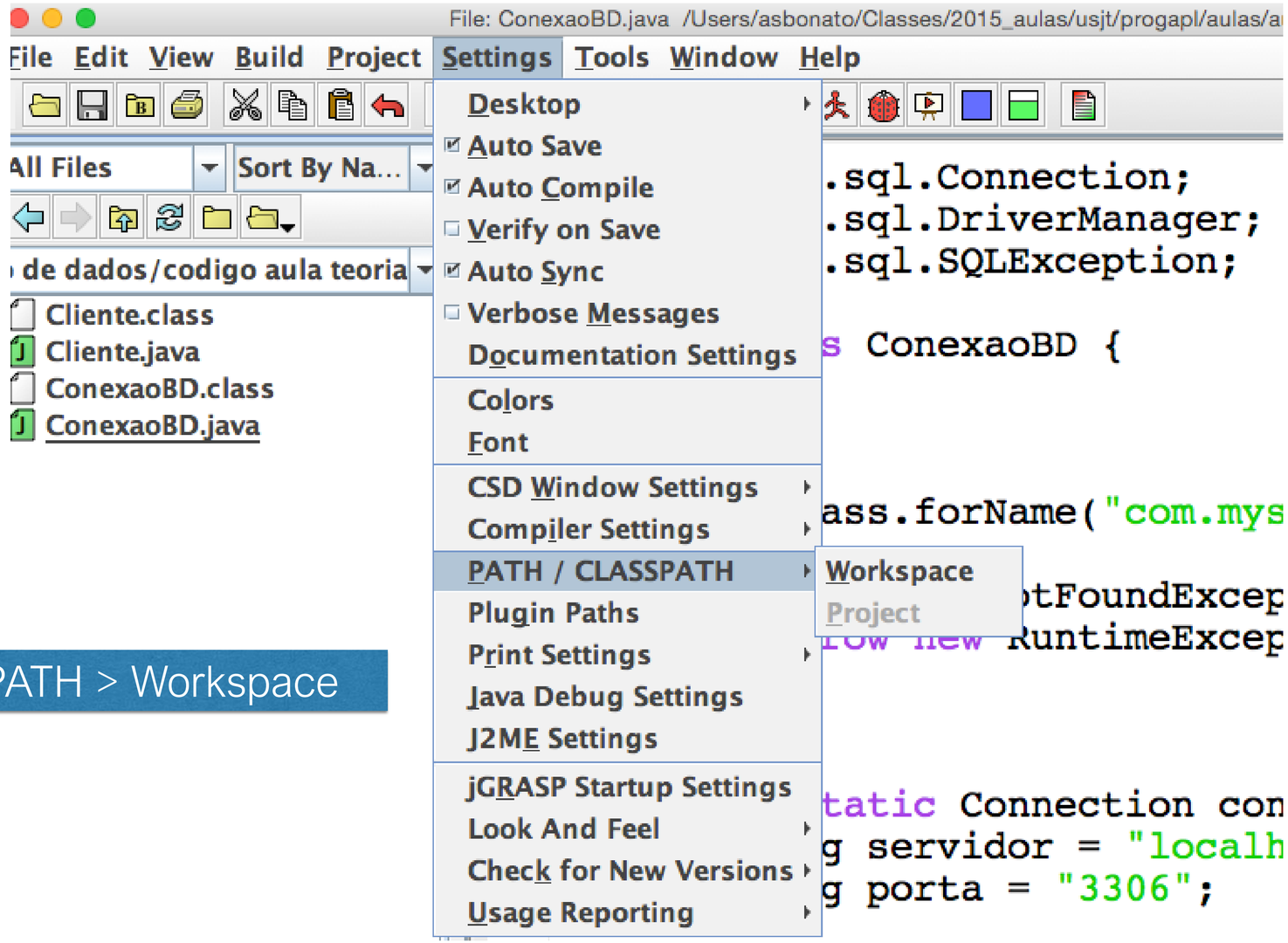
Exemplo de try-multicatch – usa-se o operador ou | para unir logicamente as exceções.

Carregando um driver de JDBC

- Use um bloco estático.
- Carregue o driver usando o método Class.forName.
- A string para pegar o driver depende de cada fabricante. No nosso caso, como trabalhamos com o MySQL, a string é com.mysql.jdbc.Driver.
- Você deve tratar a exceção ClassNotFoundException que acontece quando a classe não é encontrada.
- Não se esqueça de colocar o caminho do driver no CLASSPATH.

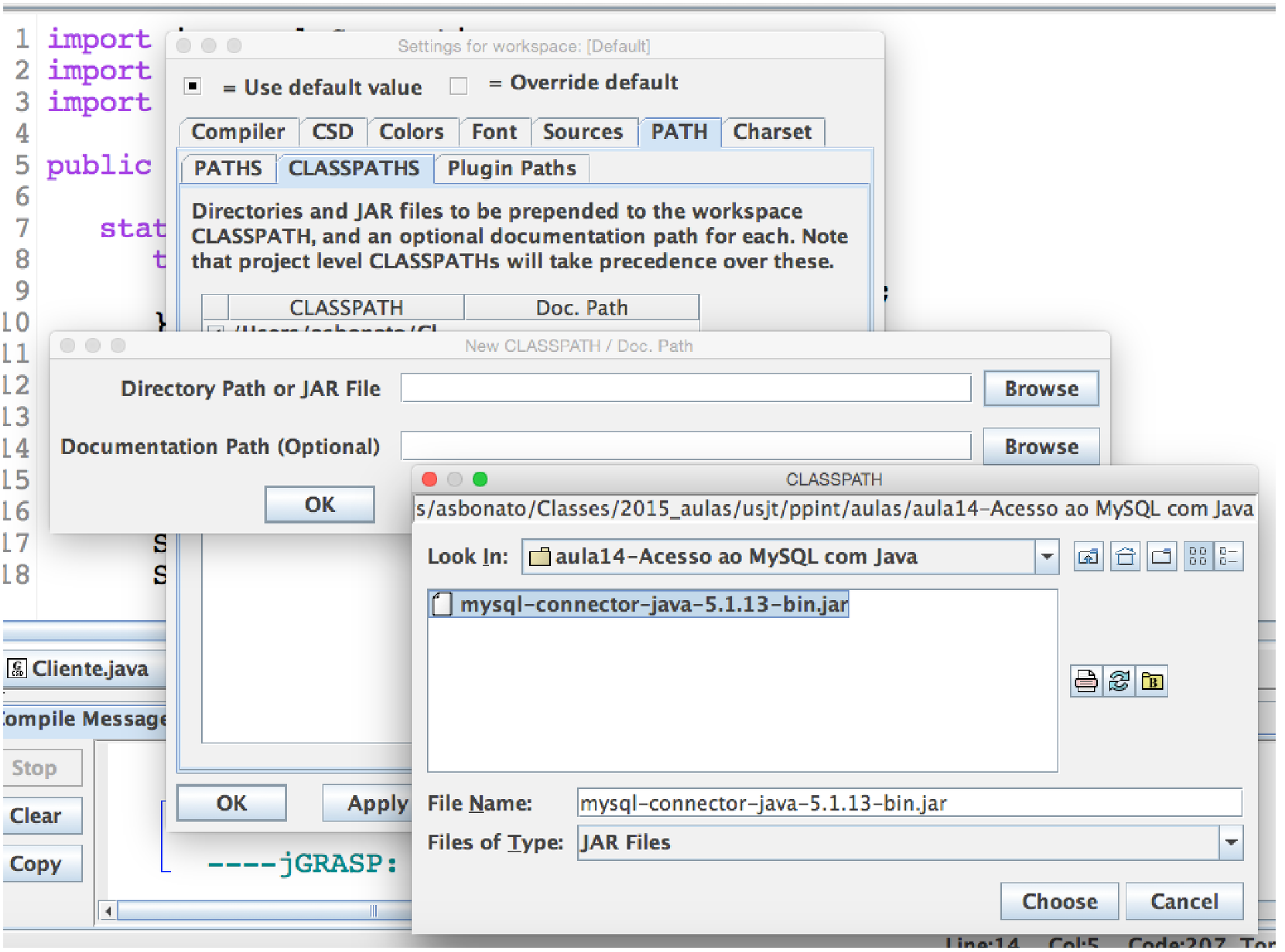
```
public class ConexaoBD {

    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        }
        catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
```

Configuração do CLASSPATH no JGrasp



Acesse o menu Settings > PATH / CLASSPATH > Workspace

- Selecione a aba CLASSPATH e clique em New.
- Clique em Browse na linha Directory Path or JAR File
- Navegue até a pasta onde está o JAR do driver de JDBC
- Selecione o Jar, clique em Chose, Ok, Ok.



Pegando a conexão

- Use o DriverManager.getConnection para pegar uma conexão com o banco.
- Neste método você tem que passar como parâmetro uma string de conexão, parecida como a URL (endereço) de um site, mas que em vez de usar http usa jdbc.
- Na string você passa o fabricante do banco, o servidor, a porta, o database, o usuário e a senha.
- Veja no exemplo a seguir que o método está lançando uma SQLException, pois várias coisas podem dar errado nesta operação: rede fora, banco fora, usuário e senha inválidos, acesso negado, etc.
- Para desconectar basta dar um close na conexão.

```
public static Connection conectar() throws SQLException {
    String servidor = "localhost";
    String porta = "3306";
    String database = "tutorial";
    String usuario = "aluno";
    String senha = "alunos";
    return DriverManager
           .getConnection("jdbc:mysql://"+servidor+":"+porta+
           "/"+database+"?user="+usuario+"&password="+senha);
}

public static void desconectar(Connection conn) throws SQLException {
    conn.close();
}
```

Inserir / Deletar / Atualizar

- Nestes casos os métodos são bem parecidos:

1. Escreva em uma string o comando SQL com pontos de interrogação (?) nos campos parametrizáveis.

2. Pegue uma conexão.

3. Peça um PreparedStatement para a conexão passando a string como parâmetro.

4. Atribua valores via set para os pontos de interrogação pelo seu número de ordem e pelo tipo de dado. O primeiro ponto de interrogação é o 1, o segundo é o 2, e assim por diante. Se for configurar um inteiro, é setInt, se for varchar, é set String.

Inserir / Deletar / Atualizar

5. Chame o método execute() do PreparedStatement.

6. Dê commit (se não estiver em auto commit).

7. Feche o preparedStatement (se não houver usado try with resources).

8. Feche a conexão (somente depois que encerrar a transação)

9. Trate as exceções e dando rollback se der errado.

Para os exemplos a seguir considere

- Uma classe Cliente com os atributos idCliente(int), nome(String), telefone(String).
- Uma banco tutorial com uma tabela Cliente com os campos id(smallint, pk), nome(varchar 60), fone (char 10).
- O código e o script de criação do banco serão fornecidos.

```
public void incluir(Connection conn) {
    String sqlInsert =
        "INSERT INTO cliente(id, nome, fone) VALUES (?, ?, ?)";

    try (PreparedStatement stm = conn.prepareStatement(sqlInsert);) {
        stm.setInt(1, getIdCliente());
        stm.setString(2, getNome());
        stm.setString(3, getFone());
        stm.execute();
    }
    catch (Exception e) {
        e.printStackTrace();
        try {
            conn.rollback();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
```

```
public void excluir(Connection conn) {
    String sqlDelete = "DELETE FROM cliente WHERE id = ?";
    try (PreparedStatement stm = conn.prepareStatement(sqlDelete);) {
        stm.setInt(1, getIdCliente());

        stm.execute();
    }
    catch (Exception e) {
        e.printStackTrace();
        try {
            conn.rollback();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
```

```
public void atualizarTelefone(Connection conn, String novoFone) {
    String sqlUpdate = "UPDATE CLIENTE SET FONE = ? WHERE ID = ?";

    try (PreparedStatement stm = conn.prepareStatement(sqlUpdate);){
        stm.setString(1, novoFone);
        stm.setInt(2, getIdCliente());

        stm.execute();
    }
    catch (Exception e) {
        e.printStackTrace();
        try {
            conn.rollback();
        }
        catch (SQLException e1) {
            System.out.print(e1.getStackTrace());
        }
    }
}
```

Consultar

- Quase igual ao anterior, mas com a diferença de que estou buscando dados, e não enviando.

1. Escreva em uma string o comando SELECT com pontos de interrogação (?) nos campos parametrizáveis.

2. Pegue uma conexão.

3. Peça um PreparedStatement para a conexão passando a string como parâmetro.

4. Atribua valores via set para os pontos de interrogação pelo seu número de ordem e pelo tipo de dado. O primeiro ponto de interrogação é o 1, o segundo é o 2, e assim por diante. Se for configurar um inteiro, é setInt, se for varchar, é set String.

Consultar

5. Chame o método executeQuery() do PreparedStatement.

6. Pegue o ResultSet

7. Navegue no ResultSet usando o método next() e pegando as informações via get e passando o número de ordem da coluna dentro do select. Pegar um inteiro na primeira coluna é getInt(1).

8. Feche o resultSet e o preparedStatement, nesta ordem (se não houver usado try with resources).

9. Feche a conexão (somente depois que encerrar alguma transação pendente)

10. Trate as exceções.

```
public void carregar(Connection conn) {
    String sqlSelect =
        "SELECT * FROM cliente WHERE cliente.id = ?";

    try (PreparedStatement stm = conn.prepareStatement(sqlSelect);){
        stm.setInt(1, getIdCliente());
        try (ResultSet rs = stm.executeQuery();){
        /*este outro try e' necessario pois nao da' para abrir o resultset
         *no anterior uma vez que antes era preciso configurar o parametro
         *via setInt; se nao fosse, poderia se fazer tudo no mesmo try
         */
            if (rs.next()) {
                this.setNome(rs.getString(2));
                this.setFone(rs.getString(3));
            }

        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
    catch (SQLException e1) {
        System.out.print(e1.getStackTrace());
    }
```

1 Cliente - busca pela PK

```java
public ArrayList<Cliente> buscarClientes(Connection conn){
    String sqlSelect = "SELECT id, nome, fone FROM CLIENTE";
    ArrayList<Cliente> lista = new ArrayList<>();

    try(PreparedStatement stm = conn.prepareStatement(sqlSelect);
        ResultSet rs = stm.executeQuery();){
        //veja que desta vez foi possivel usar o mesmo try
        while(rs.next()){
            Cliente cliente = new Cliente();
            cliente.setIdCliente(rs.getInt("id"));
            cliente.setNome(rs.getString("nome"));
            cliente.setFone(rs.getString("fone"));
            lista.add(cliente);
        }
    } catch(Exception e){
        e.printStackTrace();
    }
    return lista;
}
```

Vários clientes.
Usar uma coleção para retorná-los,

Controle de Transação

- Uma transação é um conjunto de alterações de dados no banco que acontece totalmente ou não acontece. Não pode ser parcial.
- O MySQL trabalha no modo auto commit, isto é, cada insert ou delete é efetivado no banco. Não dá para desfazer.
- Para controlar a transação, use o método setAutoCommit(false) da conexão.
- Para efetivar a transação, use commit() da conexão. Para desfazer, use rollback() da conexão.
- Importante: não abra ou feche a conexão no meio, ou irá perder a transação.

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

public class Teste {

   public static void main(String[] args) {
      Connection conn = null;
      Cliente cl;
      Vendedor vd;

      try {
         // obtem conexao com o banco
         ConexaoBD bd = new ConexaoBD();
         conn = bd.conectar();
         vd = new Vendedor();

         // *** IMPORTANTE ***: Força o uso de transação.
         conn.setAutoCommit(false);
         // *** Inclusao do Primeiro Cliente ***
         cl = new Cliente(1001, "Zé das Couves", "1127991999");
         cl.incluir(conn);

         // *** Inclusao do Segundo Cliente ***
         cl = new Cliente();
         cl.setIdCliente(1002);
         cl.setNome("João das Couves");
         cl.setFone("1160606161");
         cl.incluir(conn);

         // *** Inclusao do Terceiro Cliente ***
         cl = new Cliente(1003, "Maria das Couves", "1121212121");
         cl.incluir(conn);

         // *** IMPORTANTE ***: Efetiva inclusões
         conn.commit();

         // *** Lista todos os clientes
         System.out.println("\nLista todos os clientes");
         vd.listarClientes(conn);

         // *** Carregar o cliente 1003 a partir do bd ***
         cl = new Cliente(1003);
         System.out.println("\nLista o 1003 antes de carregar os dados");
         System.out.println(cl);
         cl.carregar(conn);
         System.out.println("\nLista o 1003 depois de carregar os dados");
         System.out.println(cl);
         // *** Excluir o cliente 1003 (carregado em cl) do bd
         cl.excluir(conn);

         // *** IMPORTANTE ***: Efetiva exclusão
         conn.commit();

         // *** Lista novamente todos os clientes
         System.out.println("\nLista todos os clientes depois de apagar o 1003");
         vd.listarClientes(conn);
      }
      catch (Exception e) {
         e.printStackTrace();
         if (conn != null) {
            try {
               conn.rollback();
            }
            catch (SQLException e1) {
               System.out.print(e1.getStackTrace());
            }
         }
      }
      finally {
         if (conn != null) {
            try {
               conn.close();
            }
            catch (SQLException e1) {
               System.out.print(e1.getStackTrace());
            }
         }
      }
   }
```