



Algoritmos e Programação II

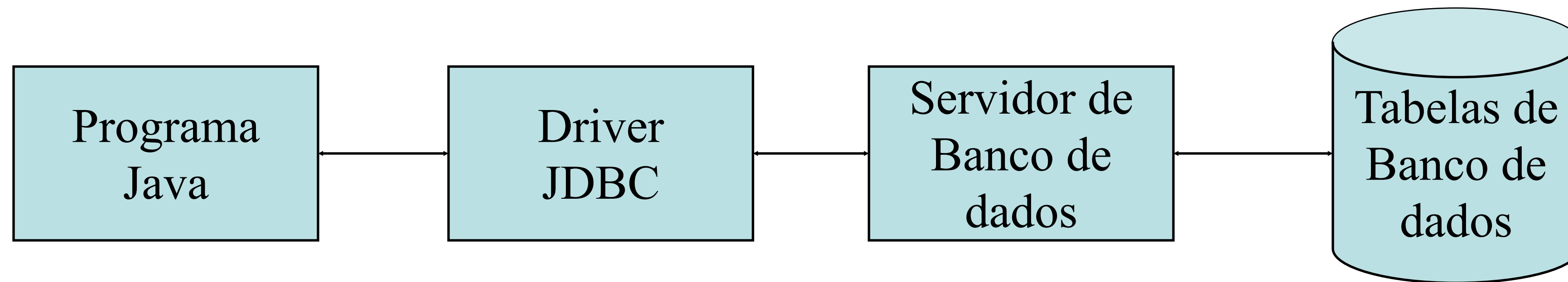
Rafael Vieira Coelho

Roteiro

- Conexão com BD em Java
- Statements, ResultSet, etc.
- Implementando o padrão DAO



Acesso ao Banco de Dados em Java



<http://dev.mysql.com/downloads/connector/j/>

TEMOS DUAS OPÇÕES

- Adicionar o arquivo jar (mysql-connector-java-8.0.29) às bibliotecas do projeto.
- Adicionar a dependência no arquivo pom.xml, caso seja um projeto maven (<https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-installing-maven.html>).

JDBC

- O que é a JDBC?
 - consiste em uma biblioteca;
 - implementada em Java;
 - disponibiliza classes e interfaces para o acesso ao banco de dados;
- Para cada banco de dados existe uma implementação.

Pacote java.sql

- Fornece a API para acesso e processamento de dados;
 - **DriverManager**, responsável por criar uma conexão com o banco de dados;
 - **Connection**, classe responsável por manter uma conexão aberta com o banco;
 - **Statement**, gerencia e executa instruções SQL;
 - **PreparedStatement**, gerencia e executa instruções SQL, permitindo também a passagem de parâmetros em uma instrução;
 - **ResultSet**, responsável por receber os dados obtidos em uma pesquisa ao banco.

DriverManager

- Responsável pelo gerenciamento de drivers JDBC;
- Estabelece conexões a bancos de dados;

// Tentando estabelecer conexão com o Banco de Dados

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://localhost/turma",  
    "login",  
    "senha");
```

Connection

- Representa a conexão com o banco de dados;
- Proporcionar informações sobre as tabelas do banco através de transações;
- Métodos:
 - ***commit()***, executa todas as alterações feitas com o banco de dados pela atual transação.
 - ***rollback()***, desfaz qualquer alteração feita com o banco de dados pela atual transação.
 - ***close()***, libera o recurso que estava sendo utilizado pelo objeto.

Statement

- Implementação de uma Interface que fornece métodos para executar uma instrução SQL;
- Não aceita a passagem de parâmetros;
- principais métodos são:
 - executeUpdate(), executa instruções SQL do tipo: INSERT, UPDATE e DELETE;
 - execute(), executa instruções SQL de busca de dados do tipo SELECT;
 - close(), libera o recurso que estava sendo utilizado pelo objeto.

Statement

Exemplo:

```
// Instanciando o objeto statement (stmt)
Statement stmt = conn.createStatement();

// Executando uma instrução SQL.
stmt.executeUpdate("INSERT INTO ALUNO VALUES (1,
'Pedro da Silva')");
```


Exemplos de uso de Statement

```
stmt.execute("CREATE TABLE filme "  
            + "(codigo INT PRIMARY KEY, "  
            + "genero CHAR(20), "  
            + "titulo CHAR(20));");
```

```
stmt.executeUpdate("INSERT INTO filme "  
                  + "(codigo, genero, titulo) VALUES "  
                  + "(499, 'Aventura', 'Frio de Lages')");
```

```
ResultSet rs = stmt.executeQuery(  
    "SELECT genero, titulo " +  
    " FROM filme " +  
    " WHERE codigo = 499");
```


PreparedStatement

- A interface PreparedStatement possui todos os recursos da interface Statement;
- Acrescentando a utilização de parâmetros em uma instrução SQL;
- métodos da interface PreparedStatement são:
 - ***execute()***, consolida a instrução SQL informada;
 - ***setDate()***, método utilizado para atribuir um valor do tipo Data;
 - ***setInt()***, utilizado para atribuir valores do tipo inteiro;
 - ***setString()***, método utilizado para atribuir valores do tipo Alfa Numéricos.

PreparedStatement

```
// Instanciando o objeto preparedStatement (pstmt)
PreparedStatement pstmt =
    conn.prepareStatement("UPDATE ALUNO
SET NOME = ?");

// Setando o valor ao parâmetro
pstmt.setString(1, "MARIA RITA");
```

ResultSet

- Esta interface permite o recebimento e gerenciamento do conjunto de dados resultante de uma consulta SQL;
- métodos capazes de acessar os dados:
 - ***next()***, move o cursor para a próxima linha de dados, já que o conjunto de dados retornados pela consulta SQL é armazenado como em uma lista.
 - ***close()***, libera o recurso que estava sendo utilizado pelo objeto.
 - ***getString(String columnName)***, recupera o valor da coluna informada como parâmetro, da linha atual do conjunto de dados recebidos pelo objeto ResultSet.

ResultSet x Tipos de dados

<i>Método de ResultSet</i>	<i>Tipo de dados SQL92</i>
<code>getInt()</code>	INTEGER
<code>getLong()</code>	BIG INT
<code>getFloat()</code>	REAL
<code>getDouble()</code>	FLOAT
<code>getBignum()</code>	DECIMAL
<code>getBoolean()</code>	BIT
<code>getString()</code>	CHAR, VARCHAR
<code>getDate()</code>	DATE
<code>getTime()</code>	TIME
<code>getTimestamp()</code>	TIME STAMP
<code>getObject()</code>	<i>Qualquer tipo (Blob)</i>

ResultSet

//Recebendo o conjunto de dados da consulta SQL

```
ResultSet rs = stmt.executeQuery("SELECT id, nome FROM ALUNO");
```

// Se houver resultados, posiciona-se o cursor na próxima linha de dados

```
while (rs.next()) {
```

```
    // Recuperando os dados retornados pela consulta SQL
```

```
    int id = rs.getInt("id");
```

```
    String nome = rs.getString("nome");
```

```
}
```

- métodos como o `getInt()`, `getString()` para recuperar os valores;

Resumindo: Acesso ao Banco de Dados em Java

- 1) Escolher um banco de dados (MySQL, Oracle, PostgreSQL,...)
- 2) Criar as tabelas do banco de dados relacional.
- 3) Baixar o driver do banco de dados e associá-lo no projeto na IDE
- 4) Importar o pacote de classes e interfaces para o acesso ao banco de dados
- 5) Estabelecer uma conexão
- 5) Executar comandos SQL
- 6) Encerrando o `statement`
- 7) Encerrando a conexão

EXEMPLO

Cadastro de Alunos

INSERÇÃO DE NOVOS ELEMENTOS

Insere Remove Alterar Pesquisa

Nome

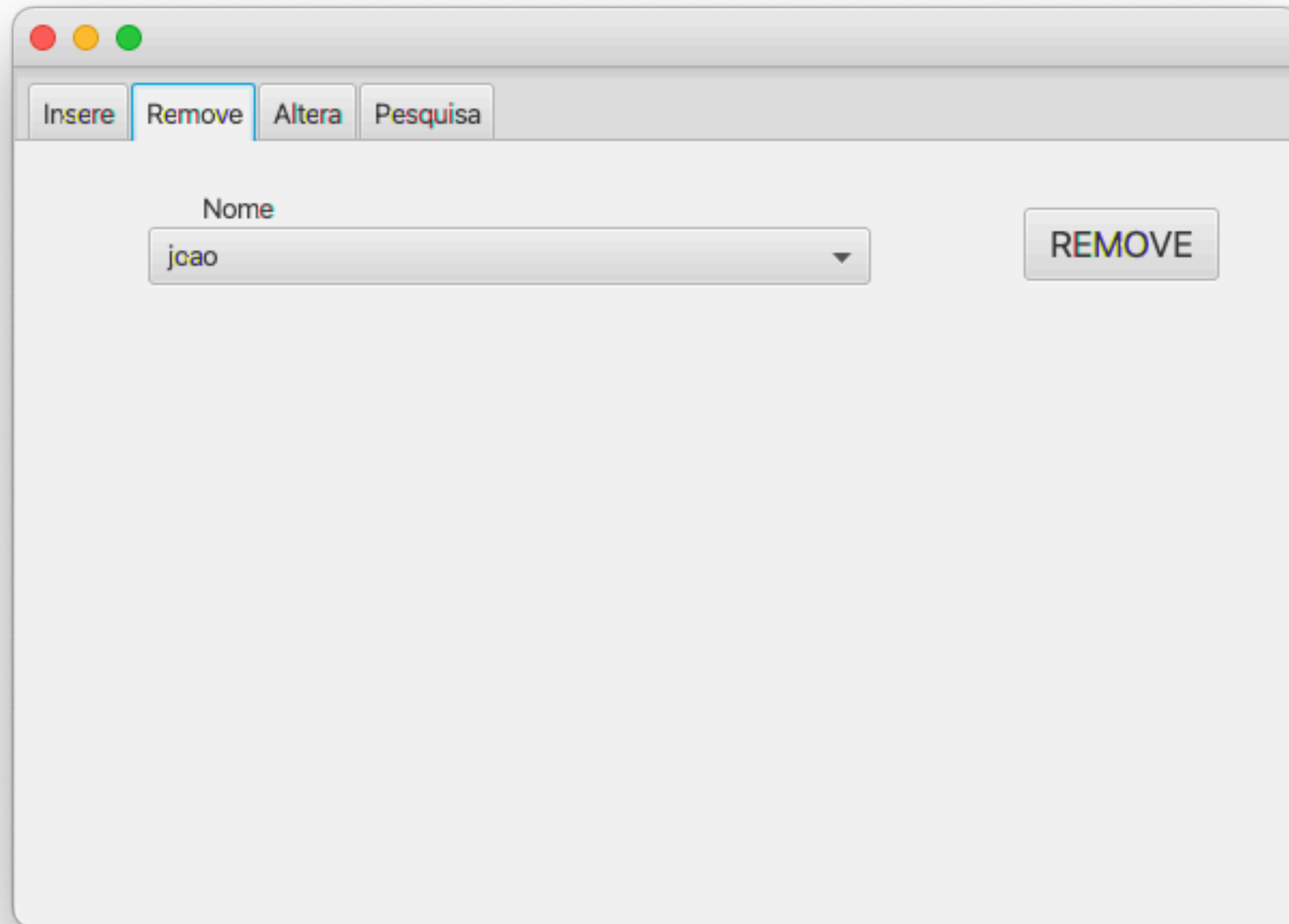
Nota 1

Nota 2

Nota 3

ADD

REMOÇÃO DE ELEMENTOS CADASTRADOS



Insere Remove Alterar Pesquisa

Nome

jcao

REMOVE

ALTERAÇÃO DE DADOS PREVIAMENTE CADASTRADOS

Insere Remove **Alterar** Pesquisa

▼ Seleção

joao ▼

▼ Alteração

Nome joao

Nota 1 10.0

Nota 2 0.0

Nota 3 10.0

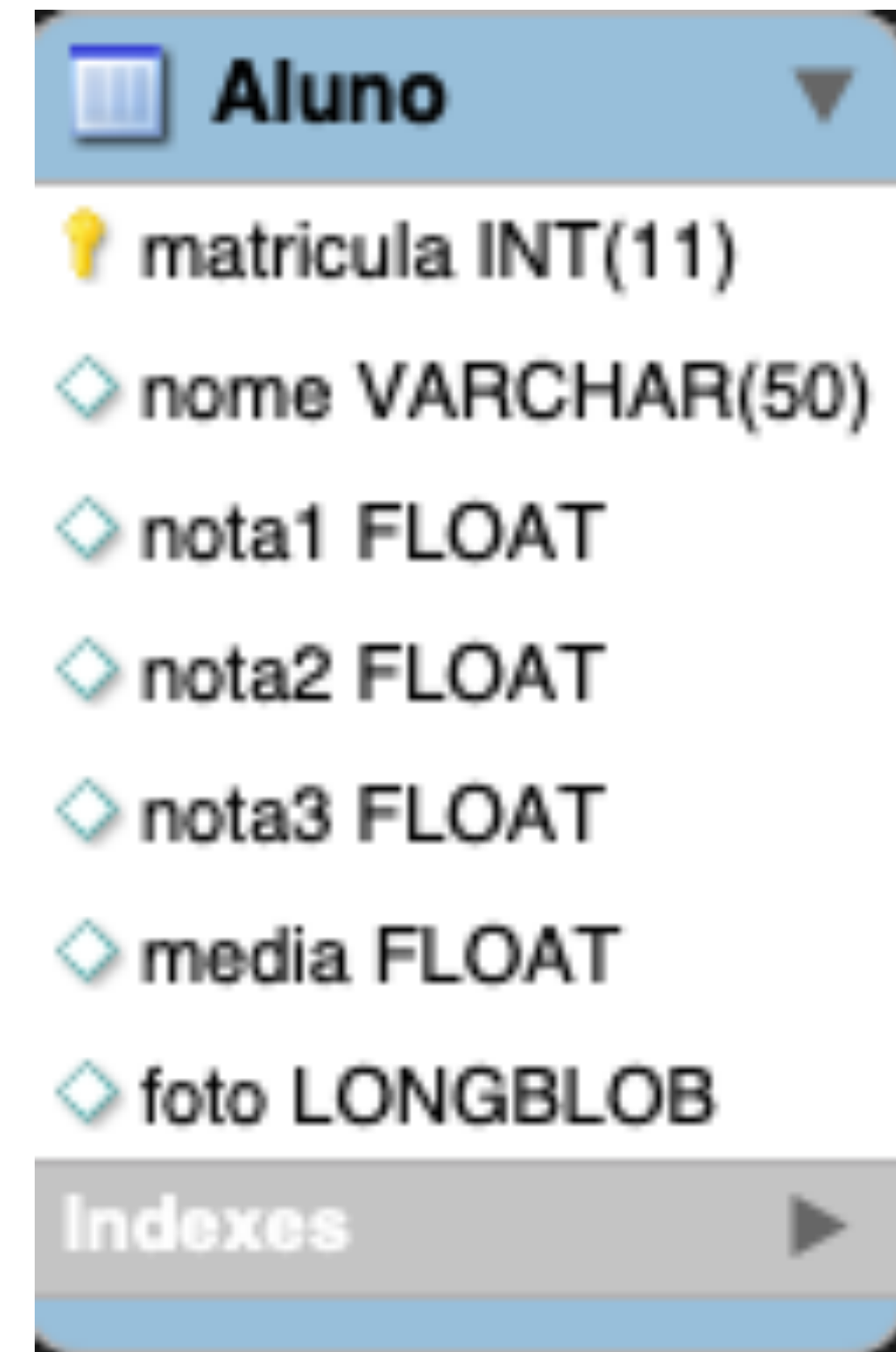
UPDATE

Criando o Banco de Dados "banco"

```
CREATE DATABASE turma;
```

```
USE turma;
```

```
CREATE TABLE IF NOT EXISTS turma.Aluno (  
    matricula INT(11) NOT NULL AUTO_INCREMENT,  
    nome VARCHAR(50),  
    nota1 FLOAT,  
    nota2 FLOAT,  
    nota3 FLOAT,  
    media FLOAT,  
    foto LONGBLOB,  
    PRIMARY KEY (matricula));
```



COMO SE CONECTAR AO BANCO DE DADOS

- Precisamos abrir uma conexão toda vez que formos fazer uma pesquisa ou alteração no BD.

```
7  ► public class ConnectionFactory {
8
9  @ public static Connection getConnection() {
10      try {
11          Connection con = DriverManager.getConnection(url: "jdbc:mysql://127.0.0.1:3306/turma
12              user: "root",
13              password: "");
14          return con;
15      } catch (SQLException e) {
16          System.out.println("Erro ao conectar no banco de dados!");
17          e.printStackTrace();
18      }
19      return null;
20  }
```

CLASSE QUE REPRESENTA O OBJETO SALVO NO BD

- Não esqueça de criar os get/set, construtor, toString, hash e equals.

```
Aluno.java x
1 package exemplobd.bd;
2
3 public class Aluno {
4
5     private int matricula;
6     private String nome;
7     private float nota1, nota2, nota3, media;
8
9     public String getNome() { return this.nome; }
10
11
12     public void setNome(String novo) { this.nome = novo; }
13
14
15
16
17     public int getMatricula() { return matricula; }
18
19
20
21     public void setMatricula(int matricula) { this.matricula = matricula; }
22
23
24
25
26     public float getNota1() { return nota1; }
27
28
29
30     public void setNota1(float nota1) { this.nota1 = nota1; }
```


CONSTANTES COM COMANDOS SQL

- Concentramos todos os comandos em um único arquivo para facilitar a descoberta de erros de sintaxe no SQL.

```
SQL_Constantes.java x
1  + /.../
6  package exemplobd.bd;
7
8  /**
9   *
10  * @author coelho
11  */
12  public class SQL_Constantes {
13
14      public static final String INSERT = "insert into "
15          + "aluno (matricula, nome, nota1, nota2, nota3, media) "
16          + "values (?, ?, ?, ?, ?, ?)";
17
18      public static final String UPDATE = "update aluno set "
19          + "nota1=?, nota2=?, nota3=?, media=? where matricula=?";
20
21      public static final String REMOVE = "delete from aluno where matricula=?";
22
23      public static final String SEARCH = "select * from aluno";
24  }
```

PADRÃO DAO (Data Access Object)









- Criaremos uma interface para padronizar o nome dos métodos.

```
1 package exemplobd.bd;
2
3 import java.util.List;
4
5 public interface Dao {
6
7     public boolean adiciona(Object m);
8
9     public boolean altera(Object m);
10
11     public boolean remove(Object m);
12
13     public boolean pesquisa(Object m);
14
15     public List<Object> pesquisaTodos();
16 }
```

```
1 package exemplobd.bd;
2
3 import ...
4
5
6
7
8
9
10 public class Dao_Aluno implements Dao {
11
12     @Override
13     public boolean adiciona(Object c) {
14         Aluno contato = (Aluno) c;
15         String sql = SQL_Constantes.INSERT;
16         try {
17             Connection connection = ConnectionFactory.getConnection();
18             PreparedStatement stmt = connection.prepareStatement(sql);
19             stmt.setString(parameterIndex: 1, x: null);
20             stmt.setString(parameterIndex: 2, contato.getNome());
21             stmt.setFloat(parameterIndex: 3, contato.getNota1());
22             stmt.setFloat(parameterIndex: 4, contato.getNota2());
23             stmt.setFloat(parameterIndex: 5, contato.getNota3());
24             stmt.setFloat(parameterIndex: 6, contato.getMedia());
25             stmt.execute();
26             stmt.close();
27             connection.close();
28         } catch (SQLException e) {
29             System.out.println("Erro ao inserir dados na tabela contato!");
30             return false;
31         }
32         return true;
33     }
```


CLASSE DAO_ALUNO.JAVA

- Verifique que os tipos estão corretos e na ordem certa nos métodos get.
- Este método retorna uma lista de objetos a partir do BD.

```
35
36  
37
38 
39
40
41
42 
43
44
45
46
47
48
49
50
51 
52
53
54 
55
56 
57
58 

@Override
public List<Object> pesquisaTodos() {
    List<Object> contatos = new ArrayList();
    try {
        Connection connection = ConnectionFactory.getConnection();
        PreparedStatement stmt = connection.prepareStatement(SQL_Constantes.SEARCH);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Aluno contato = new Aluno();
            contato.setMatricula(rs.getInt( columnLabel: "matricula"));
            contato.setNome(rs.getString( columnLabel: "nome"));
            contato.setNota1(rs.getFloat( columnLabel: "nota1"));
            contato.setNota2(rs.getFloat( columnLabel: "nota2"));
            contato.setNota3(rs.getFloat( columnLabel: "nota3"));
            contato.setMedia(rs.getFloat( columnLabel: "media"));
            contatos.add(contato);
        }
        rs.close();
        stmt.close();
    } catch (SQLException e) {
        System.out.println("Erro ao pesquisar por contatos no banco de dados!");
    }
    return contatos;
}
```


CLASSE DAO_ALUNO.JAVA

```
100      @Override
101      public boolean pesquisa(Object c) {
102          Aluno contato = (Aluno) c;
103          List<Object> todos = pesquisaTodos();
104
105          for (Object cc : todos) {
106              if (((Aluno)cc).equals(contato)) {
107                  return true;
108              }
109          }
110          return false;
111      }
112
```

```
112
113      public int buscaId(Aluno modelo) {
114          List<Object> contatos = new Dao_Aluno().pesquisaTodos();
115
116          for (Object mod : contatos) {
117              Aluno c = (Aluno) mod;
118              if (modelo.getNome().equals(c.getNome()))
119                  return c.getMatricula();
120          }
121          return -1;
122      }
123
124      public int buscaId(String nome) {
125          List<Object> contatos = new Dao_Aluno().pesquisaTodos();
126
127          for (Object mod : contatos) {
128              Aluno c = (Aluno) mod;
129              if (nome.equals(c.getNome()))
130                  return c.getMatricula();
131          }
132          return -1;
133      }
```

CLASSE DAO_ALUNO.JAVA

- Aqui retornamos um objeto a partir de um nome recebido por parâmetro.
- Caso não seja encontrado, retornamos null.

```
135 public Aluno pesquisaAluno(String nome) {  
136     List<Object> contatos = new Dao_Aluno().pesquisaTodos();  
137  
138     for (Object mod : contatos) {  
139         Aluno c = (Aluno) mod;  
140         if (c.getNome().equals(nome))  
141             return c;  
142     }  
143     return null;  
144 }
```







CLASSE DAO_ALUNO.JAVA

- Recebemos um objeto preenchido como parâmetro de entrada.
- Note que ele é genérico e precisa ser feito o cast.

60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```
@Override
public boolean altera(Object c) {
    Aluno contato = (Aluno) c;
    String sql = SQL_Constantes.UPDATE;
    try {
        Connection connection = ConnectionFactory.getConnection();
        PreparedStatement stmt = connection.prepareStatement(sql);
        stmt.setFloat( parameterIndex: 1, contato.getNota1());
        stmt.setFloat( parameterIndex: 2, contato.getNota2());
        stmt.setFloat( parameterIndex: 3, contato.getNota3());
        stmt.setFloat( parameterIndex: 4, contato.getMedia());
        stmt.setInt( parameterIndex: 5, contato.getMatricula());
        stmt.execute();
        stmt.close();
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Erro ao alterar dados do contato " + contato.getNome());
        return false;
    }
    return true;
}
```


CLASSE DAO_ALUNO.JAVA

```
82
83  
84
85 
86
87
88
89
90
91
92 
93
94
95
96 
97
98 

@Override
public boolean remove(Object c) {
    Aluno contato = (Aluno) c;
    try {
        Connection connection = ConnectionFactory.getConnection();
        PreparedStatement stmt = connection.prepareStatement(SQL_Constantes.REMOVE);
        stmt.setInt(parameterIndex: 1, contato.getMatricula());
        stmt.execute();
        stmt.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Erro ao remover contato " + contato.getNome());
        return false;
    }
    return true;
}
```

EXERCÍCIOS

- Reutilize as classes criadas previamente do exercício sobre livros e crie o BD e as classes DAO correspondentes.