

Linguagem de Programação Java



CONEXÃO AO BANCO DE DADOS

JDBC

Java Database Connectivity

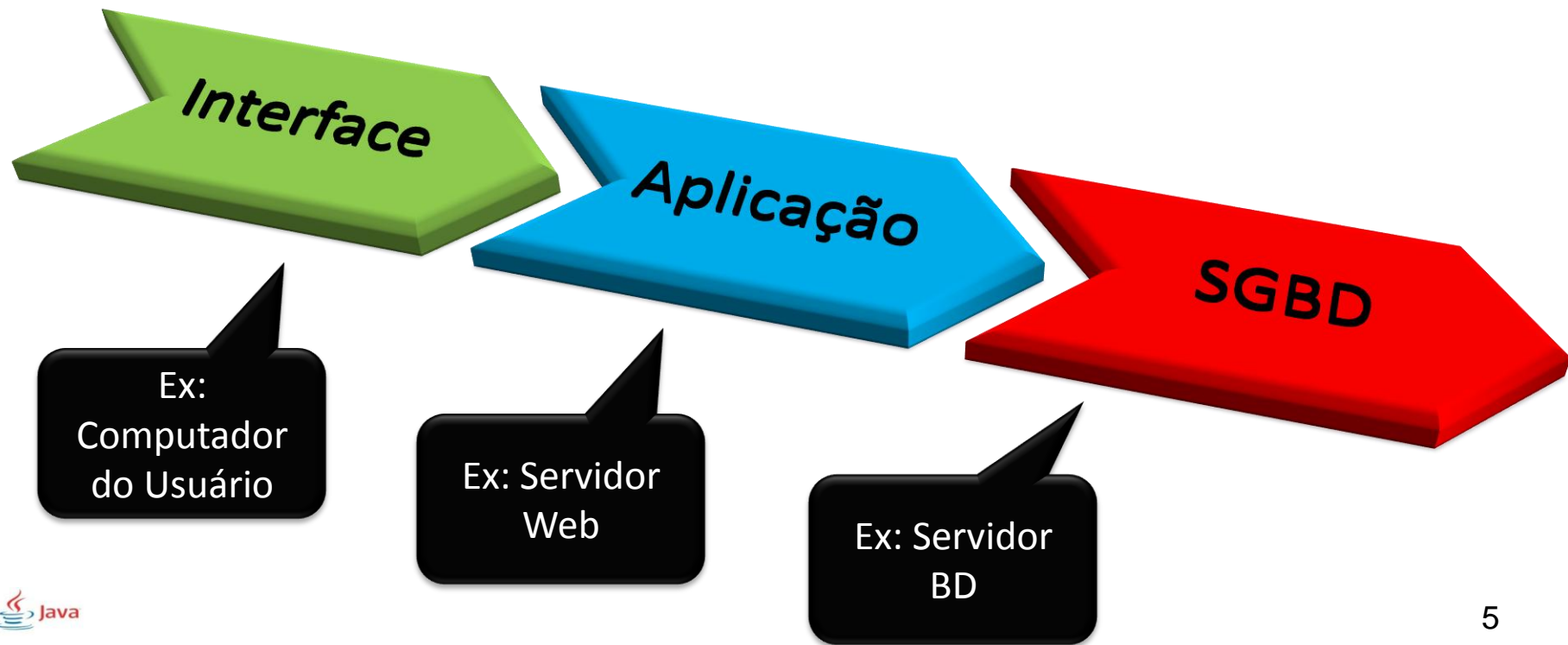
É um conjunto de classes e interfaces escritas em Java que fazem o envio de instruções SQL para o banco de dados.

JDBC x ODBC (Open Database Connectivity)

JDBC não é necessário configurar localmente. ODBC é.

JDBC possui benefícios como compatibilidade, segurança e portabilidade.

Modelo de 3 Camadas



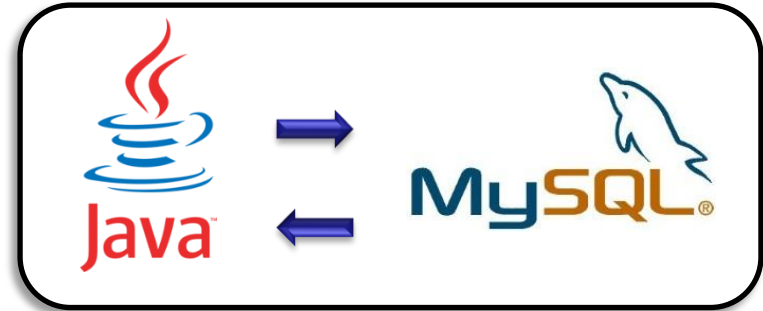
Classe ***SQLException***

É a Classe que trata os erros que acontecem na camada SGBD.

Seus métodos mais utilizados são:

- **getMessage ()** – Retorna uma string descrevendo o erro ocorrido;
- **getErrorCode ()** – Retorna um código específico do banco de dados que está sendo utilizado;
- **getSQLState ()** – Retorna a instrução SQL que deu erro.

Conexão com SGBD



É necessário fazer o download do Framework que possui o driver do respectivo SGBD.

Endereço web:

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

Conexão com SGBD (Cont.)

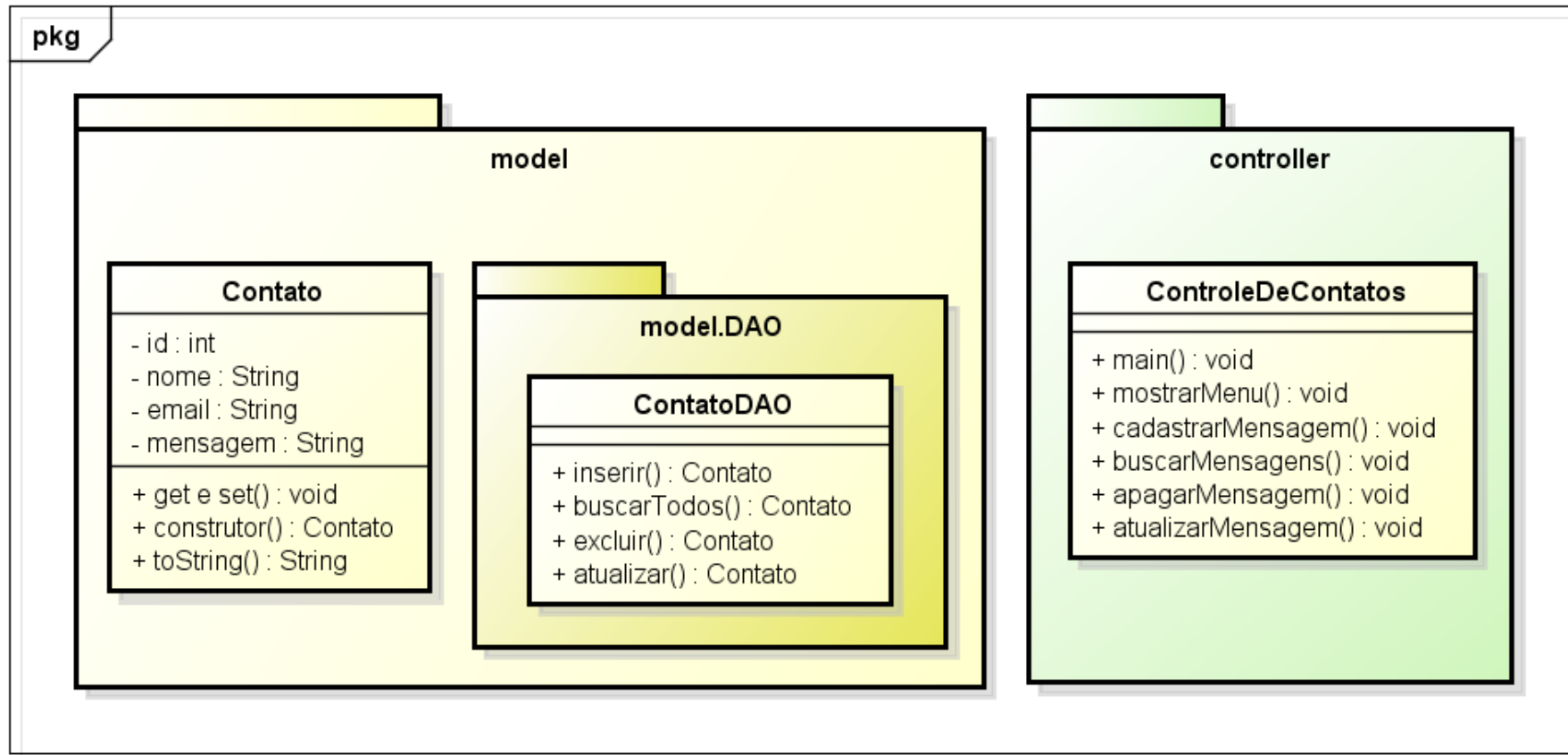
- Abrir o navegador no site
<http://www.mysql.com/downloads/connector/j/>
- Fazer o download do **Connector/J**
- Descompactar
- Copiar o arquivo **mysql-connector-java-...-bin.jar** e colar dentro da pasta da Workspace.

Exemplo ***18_ConexaoBD***

Trabalharemos neste projeto, que será o nosso exemplo de utilização do SGBD MySQL para conexão com Java

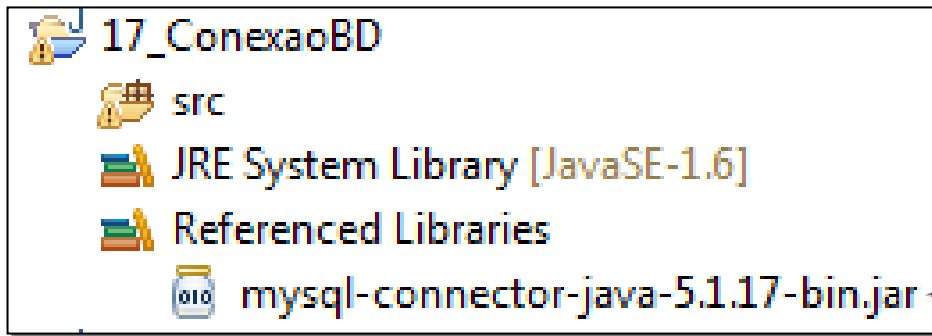
- Crie no Eclipse o projeto ***18_ConexaoBD***
- Nos próximos slides trabalharemos na criação de diversas Classes e pacotes dentro deste projeto, para exemplificarmos a utilização de conexão ao banco de dados.

Exemplo *18_ConexaoBD* (Cont.)



Criando a referência ao conector:

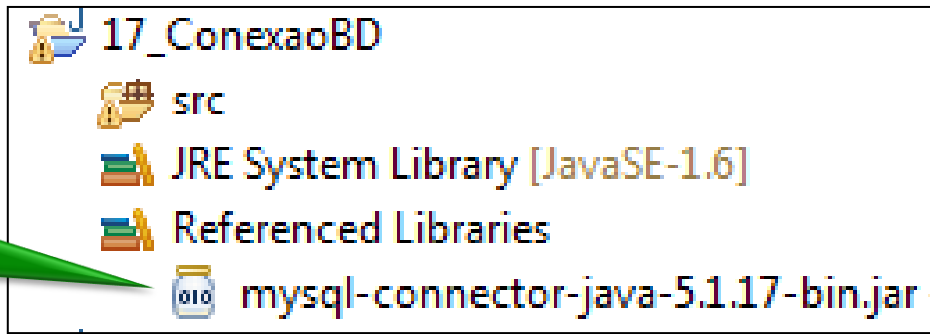
1. No Eclipse, clicar com o **botão direito** do mouse sobre o projeto.
2. Selecionar a opção **Build Path / Configure Build Path...**
3. Clicar na aba **Libraries**, e depois no botão **Add External JARs...**
4. Localizar a pasta da Workspace e **selecionar** o arquivo do conector que você já salvou dentro desta pasta.
5. Clicar em **OK**.



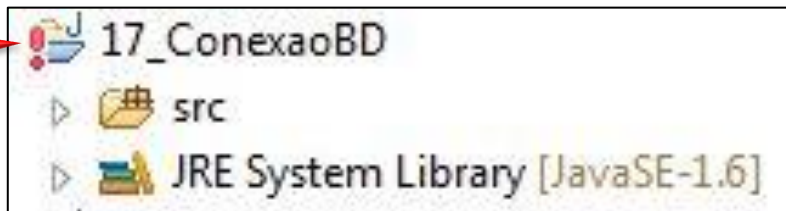
Indicação do conector no projeto

ATENÇÃO:

**Indicação de que a IDE
localizou corretamente o
conector**



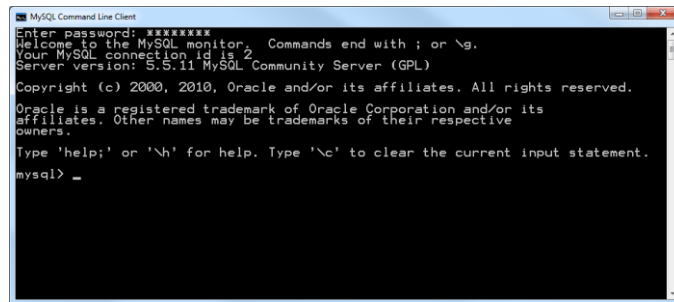
**Indicação de que a IDE
NÃO localizou
corretamente o conector**



Criando o Banco de Dados

Objetivos:

- Criar um novo Banco de Dados no MySQL
- Criar uma tabela com as colunas referentes às informações que serão salvas via conexão com Java.



```
MySQL Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.5.11 MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

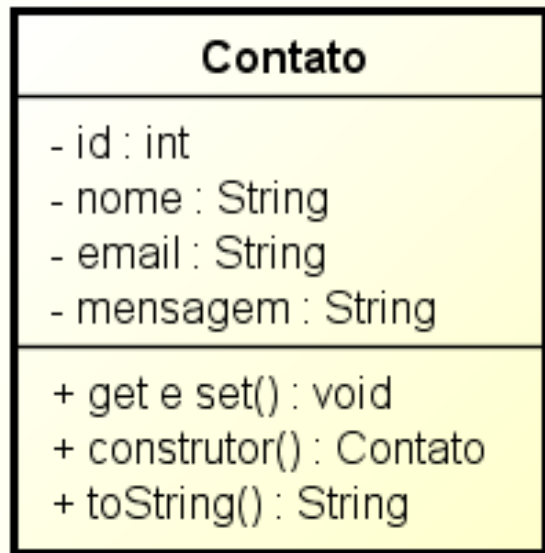
Criando o Banco de Dados (*Cont.*)

1. Abrir o ***MySQL Command Line Client*** e se logar.
2. Criar e ativar o Banco de Dados com o nome ***18_conexaobd***
3. Criar a tabela **Contato** com a seguinte estrutura:

Colunas	Domínios
id	int primary key auto_increment
nome	varchar(255)
email	varchar(100)
mensagem	varchar(255)

Estamos criando um BD com o mesmo nome do projeto Java apenas para facilitar o entendimento. Este BD poderia ter qualquer outro nome.

Criando a Classe *Contato*



Esta classe possui os atributos correspondentes à tabela ***contato*** do Banco de Dados.

Crie o pacote ***model*** e, dentro deste, a Classe ***Contato***, conforme o diagrama ao lado.

Criando a Classe *Contato*

Parte 1

```
package model;  
  
public class Contato {  
  
    private int id;  
    private String nome;  
    private String email;  
    private String mensagem;
```

Atributos

Criar também os
Métodos get e set, e
construtor básico.

Criando a Classe *Contato*

Parte 2

```
public Contato(int id, String nome, String email, String mensagem) {  
    super();  
    this.id = id;  
    this.nome = nome;  
    this.email = email;  
    this.mensagem = mensagem;  
}
```

Construtor para **visualização**
de dados a partir de um campo
id auto_increment

Criando a Classe *Contato*

Parte 3

```
public Contato(String nome, String email, String mensagem) {  
    super();  
    this.nome = nome;  
    this.email = email;  
    this.mensagem = mensagem;  
}
```

Construtor para **gravação**
de dados no banco de
dados a partir de um campo
id auto_increment

Criando a Classe *Contato*

Parte 4 - Fim

@Override

```
public String toString(){  
    final String ENTER = "\n";  
    String retValue = "";  
  
    retValue = "Mensagem Enviada Com Sucesso:" + ENTER +  
    "Nome: " + nome + ENTER +  
    "E-mail: " + email + ENTER +  
    "Mensagem: " + mensagem;  
  
    return retValue;  
}
```

Método
toString()

Criando a Classe *Conexão*

Objetivos:

Determinar elementos necessários para a conexão do banco de dados junto à Classe de drivers baixada da internet.

São eles:

- Endereço do Banco de Dados
- Classe Java que é o driver do framework baixado na web.
- Login do usuário no SGBD
- Senha do usuário no SGBD

Criando a Classe *Conexão*

Parte 1

```
package util;
```

Criar o pacote *util*, e dentro deste, a Classe *Conexao*

Todos os *imports* desta classe são de *java.sql*

```
public class Conexao {
```

```
    private String url; // Local do Banco de Dados.
```

```
    private String driver; // Classe Java do Framework que foi baixado na web.
```

```
    private String login; // Login do usuário no SGBD.
```

```
    private String senha; // Senha do usuário no SGBD.
```

Aqui estão os 4 atributos necessários para fazer a conexão.

Criando a Classe *Conexão*

Parte 2

```
public Conexao(String url, String driver, String login, String senha){  
    try{
```

```
        this.url = url;
```

```
        this.driver = driver;
```

```
        this.login = login;
```

```
        this.senha = senha;
```

```
        Class.forName(driver);
```

```
        // registro da Classe de driver na conexão através de JDBC.
```

```
    } catch (ClassNotFoundException e) {
```

```
        System.out.println(e.getMessage());
```

```
    }
```

```
}  Java
```

É por este método que o *JDBC* é informado sobre qual é o driver.

Criando a Classe *Conexão*

Parte 3

```
public Connection obterConexao() {  
    Connection con = null;  
    try{  
        con = DriverManager.getConnection(url, login, senha);  
    } catch (SQLException e) {  
        System.out.println(e.getMessage());  
    }  
    return con;  
}
```

Este método informa ao *DriverManager* a *url* do BD, o *login* e *senha* do usuário.

Criando a Classe *Conexão*

Parte 4 - Fim

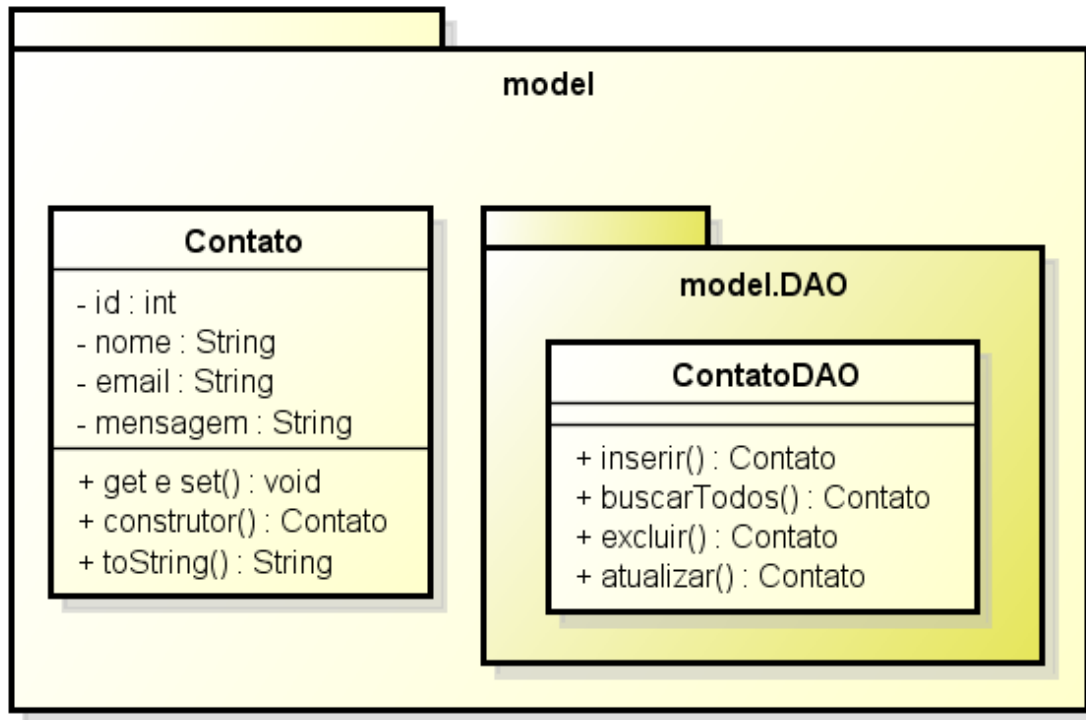
```
public String getDriver(){  
    return driver;  
}
```

```
public String getLogin() {  
    return login;  
}
```

```
public String getSenha() {  
    return senha;  
}
```

Utilizam-se somente os métodos **get**, porque as entradas de dados são feitas pelo programador.
(url , driver , login , senha)

Criando a Classe *ContatoDAO*



DAO = Data Access Object

Objetivo:

Estabelecer o contato efetivo entre o Java e o SGBD, realizando as ações de seus métodos na tabela do Banco de Dados

Inserir Mensagens...

O objetivo agora é possibilitar a inclusão de novas mensagens no Banco de Dados.

Criando a Classe *ContatoDAO*

Parte 1

```
public class ContatoDAO {
```

Criar o pacote *model.DAO* dentro do pacote *model*.
Criar Classe *ContatoDAO* dentro do pacote *model.DAO*

```
    public static Contato inserir(String nome, String email, String mensagem){
```

```
        Contato contato = null;
```

```
        try {
```

```
            // Criação do insert
```

```
            String sql = "insert into contato (nome, email, mensagem) values (?, ?, ?)";
```

Trabalharemos por enquanto apenas com o
método *inserir*

```
            // Obter a conexão com o banco de dados
```

```
            Conexao conex = new Conexao("jdbc:mysql://localhost:3306/18_conexaobd",  
                                         "com.mysql.jdbc.Driver", "root", "alunolab");
```

```
            // Abrir a conexão
```

```
            Connection con = conex.obterConexao();
```

Continua ...



Criando a Classe *ContatoDAO*

Continuando ...

Parte 2 - Fim

```
// Preparar o comando para ser executado
```

```
PreparedStatement comando = con.prepareStatement(sql);
```

```
comando.setString(1,nome);
```

```
comando.setString(2,email);
```

```
comando.setString(3,mensagem);
```

```
// Comando executado
```

```
comando.executeUpdate();
```

```
} catch (Exception e) {
```

```
    System.out.println(e.getMessage());
```

```
}
```

```
contato = new Contato (nome, email, mensagem);
```

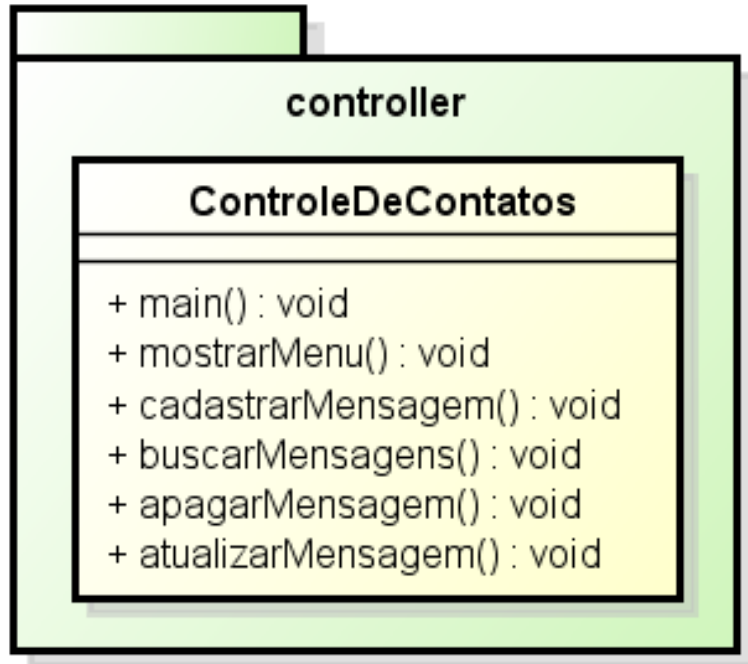
```
return contato;
```

```
}
```

```
}
```



Criando a Classe *ControleDeContatos*



Objetivo:

Implementar as ações do sistema.

OBS: Antes de criar esta Classe, copie a Classe **Teclado** do projeto **08_EntradaDeDados** e cole dentro do pacote **util** deste projeto atual.

Criando a Classe *ControleDeContatos*

Estudaremos a elaboração desta Classe em etapas, começando pela criação da Classe e implementação das constantes abaixo:

```
package controller;  
  
public class ControleDeContatos {  
  
    private static final int CADASTRAR_MENSAGEM = 1;  
    private static final int VISUALIZAR_MENSAGENS = 2;  
    private static final int APAGAR_MENSAGEM = 3;  
    private static final int ATUALIZAR_MENSAGEM = 4;  
    private static final int SAIR = 5;  
}
```

Criar pacote
controller e, dentro
deste, a Classe
ControleDeContatos

Criando a Classe *ControleDeContatos*

```
public static void main(String[] args) {  
    ControleDeContatos cdc = new ControleDeContatos();  
    int opcao = SAIR;  
    do {  
        cdc.mostrarMenu();  
        opcao = Teclado.lerInt("Digite sua opção: ");  
  
        switch (opcao) {  
            case CADASTRAR_MENSAGEM:  
                cdc.cadastrarMensagem();  
                break;  
            default:  
                System.out.println("Opção inválida!");  
        }  
        Teclado.lerTexto("Pressione uma tecla para continuar...");  
    } while (opcao != SAIR);  
}
```

Método *main()*

Leitura de dados do
Teclado

Por enquanto, apenas
o *case* para a opção de
cadastrarMensagem.

Criando a Classe *ControleDeContatos*

Método *mostrarMenu()*

```
public void mostrarMenu() {  
    System.out.println("=====");  
    System.out.println("          Cadastro de Mensagens         ");  
    System.out.println("=====");  
    System.out.println("1 - Cadastrar Mensagem");  
    System.out.println("2 - Mostrar Mensagens");  
    System.out.println("3 - Apagar Mensagem");  
    System.out.println("4 - Atualizar Mensagem");  
    System.out.println("5 - Sair");  
}
```


Criando a Classe *ControleDeContatos*

Método *cadastrarMensagem()*

```
public void cadastrarMensagem() {  
    System.out.println("=====");  
    System.out.println("          Cadastro de Mensagens         ");  
    System.out.println("=====");  
    String nome = Teclado.lerTexto("Nome: ");  
    String email = Teclado.lerTexto("E-mail: ");  
    String mensagem = Teclado.lerTexto("Mensagem: ");  
  
    Contato contato = ContatoDAO.inserir(nome, email, mensagem);  
  
    System.out.println(contato);  
}
```

Consultar Mensagens...

O objetivo agora é possibilitar a consulta de mensagens já cadastradas no Banco de Dados.

Continuando a Classe *ContatoDAO*

```
public static Contato[] buscarTodos() {  
    Contato[] contatos = null;
```

```
    try {  
        // Criação do select  
        String sql = "Select * from contato";  
  
        // Obter a conexão com o banco de dados  
        Conexao conex = new Conexao ("jdbc:mysql://localhost:3306/18_conexaobd",  
                                       "com.mysql.jdbc.Driver", "root", "alunolab");  
  
        Connection con = conex.obterConexao();
```

Método
buscarTodos()

Parte 1

Continuando a Classe *ContatoDAO*

```
/* Executa a confirmação direta de acesso ao banco
 * pois não são necessárias informações para a
 * Query (caracter curinga)
 */
```

```
Statement comando = con.createStatement();
```

```
/* ResultSet - Classe java que monta em memória uma matriz
 * com a resposta das linhas do banco de dados
 */
```

```
ResultSet rs = comando.executeQuery(sql);
```

```
// vetor de objetos
```

```
contatos = new Contato[10];
```

Método
buscarTodos()

Parte 2

Continuando a Classe *ContatoDAO*

```
/* Passagem de linha de dados do ResultSet para o vetor de objetos  
 * (uma linha de dados da matriz do ResultSet é copiada para  
 * um objeto no vetor contatos)  
 */
```

```
int i = 0;  
while (rs.next()) {  
    contatos[i++] = new Contato(  
        rs.getInt("id"),  
        rs.getString("nome"),  
        rs.getString("email"),  
        rs.getString("mensagem"));  
}
```



Método
buscarTodos()

Parte 3

Continua ...

Continuando a Classe *ContatoDAO*

```
// É necessário encerrar o acesso ao banco para liberar a conexão
rs.close();
comando.close();
con.close();
} catch (Exception e) {
    System.out.println(e.getMessage());
}
return contatos;
}
```

Método
buscarTodos()

Parte 4 - Fim

Continuando a Classe *ControleDeContatos*

Método *buscarMensagem()*

```
public void buscarMensagens () {  
    Contato[] contatos = ContatoDAO.buscarTodos();  
  
    for (int i=0; i<contatos.length; i++){  
        if (contatos[i] !=null){  
            System.out.println(  
                contatos[i].getId() + "-----" +  
                contatos[i].getNome() + "-----" +  
                contatos[i].getEmail() + "-----" +  
                contatos[i].getMensagem());  
        }  
    }  
}
```

Continuando a Classe *ControleDeContatos*

Método *main()*

Inserir após o *break* do *case CADASTRAR_MENSAGEM*:

```
case VISUALIZAR_MENSAGENS:  
    cdc.buscarMensagens ();  
    break;
```


Excluir Mensagem...

O objetivo agora é possibilitar a exclusão de uma mensagem já cadastrada no Banco de Dados.

Continuando a Classe *ContatoDAO*

Método
excluir()

Parte 1

Copiar e colar do método *inserir()*, realizando apenas as seguintes alterações:

- Declaração do método:

```
public static Contato excluir(int id) {
```

- Query e comentário da *String sql*:

```
// Criação do delete
```

```
String sql = "delete from contato where id = ?";
```

Continua...

Continuando a Classe *ContatoDAO*

Método
excluir()

Parte 2 - Fim

- Na sequência de ações do objeto *comando*, deixar exatamente assim:

```
comando.setInt(1, id);  
comando.executeUpdate();
```

- Apagar a linha:

```
contato = new Contato (nome, email, mensagem);
```

Continuando a Classe *ControleDeContatos*

Método *apagarMensagem()*

```
public void apagarMensagem() {  
    System.out.println("=====");  
    System.out.println("                Apagar Mensagem                ");  
    System.out.println("=====");  
    int id = Teclado.lerInt("Digite o número da mensagem a ser apagada:");  
  
    ContatoDAO.excluir(id);  
  
    System.out.println("Mensagem apagada com sucesso");  
}
```

OBS: Fazer o case referente a este método junto aos outros cases dentro do método *main()* desta Classe.

Atualizar Mensagem...

O objetivo agora é possibilitar a atualização (alteração) de uma mensagem já cadastrada no Banco de Dados.

Continuando a Classe *ContatoDAO*

Método
atualizar()

Parte 1

Copiar e colar do método *inserir()*, realizando apenas as seguintes alterações:

- Declaração do método:

```
public static Contato atualizar(String mensagem, int id) {
```

- Query e comentário da *String sql*:

```
// Criação do update
```

```
String sql = "update contato set mensagem = ? where id = ? ";
```

Continuando a Classe *ContatoDAO*

Método
atualizar()

Parte 2 - Fim

- Na sequência de ações do objeto *comando*, deixar exatamente assim:

```
comando.setString(1,mensagem) ;  
comando.setInt(2,id) ;
```

- Apagar a linha:

```
contato = new Contato (nome, email, mensagem) ;
```

Continuando a Classe *ControleDeContatos*

Método *atualizarMensagem()*

```
public void atualizarMensagem() {  
    System.out.println("=====");  
    System.out.println("          Atualizar Mensagem          ");  
    System.out.println("=====");  
    int id = Teclado.lerInt("Digite o número id da mensagem a ser editada:");  
    String mensagem = Teclado.lerTexto("Mensagem: ");  
  
    ContatoDAO.atualizar(mensagem, id);  
  
    System.out.println("Mensagem atualizada com sucesso");  
}
```

OBS: Fazer o case referente a este método junto aos outros cases dentro do método *main()* desta Classe.

Projeto *InfoNote_13*

- Copiar o projeto *InfoNote_12* e colar renomeando-o.
- Fazer com que todas as tarefas do projeto acessem o Banco de Dados.



Pressione o botão vermelho para abrir o documento contendo o passo a passo desta tarefa.

[JAVA2 - TI - 05.1 - Instruções Projeto InfoNote_13.pdf](#)

Dúvidas?



Bibliografia



Java Como Programar 8ª Edição
Paul Deitel e Harvey Deitel
Ed. Pearson



Java 7 Ensino Didático
Sérgio Furgeri
Ed. Érica