

UC Programação de Soluções Computacionais (2024.1)

Exemplo de um projeto prático - Tema: Agenda de Contatos

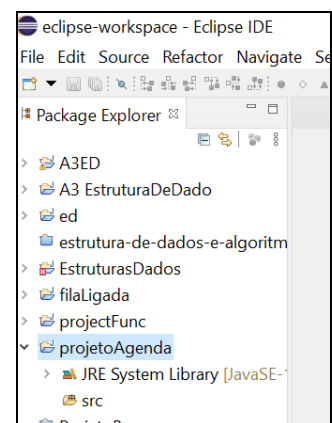
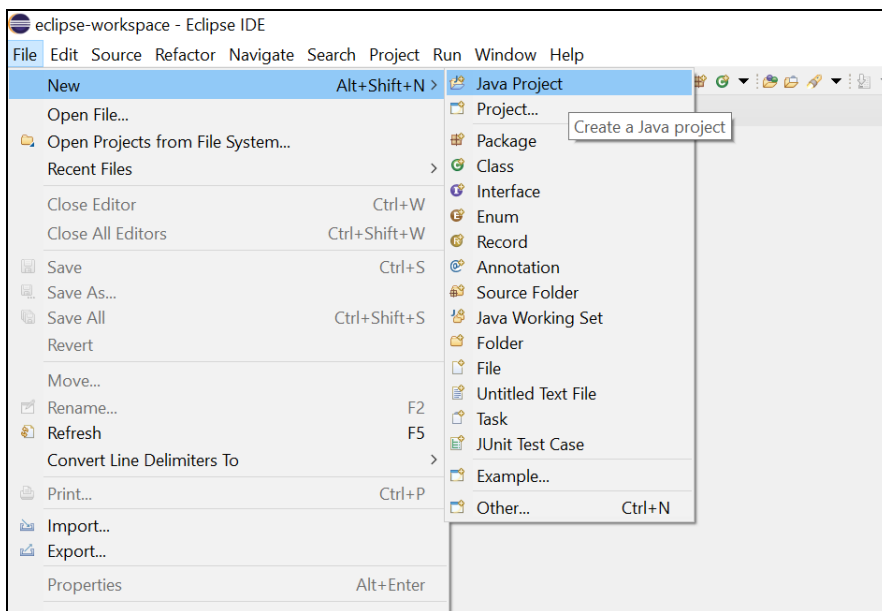
OBS.: Seja criativo na sua implementação. NÃO COPIE, POR COPIAR.

Objetivo: desenvolver uma aplicação desktop com banco de dados para sua Agenda de Contatos.

1ª. ETAPA

PASSO 1) CRIAÇÃO DO PROJETO

Abrir o Eclipse e informar o Nome do Projeto para criação. Neste exemplo será uma Agenda de Contatos. Em seguida, informe a Localização do Projeto, podendo escolher o diretório que desejar.

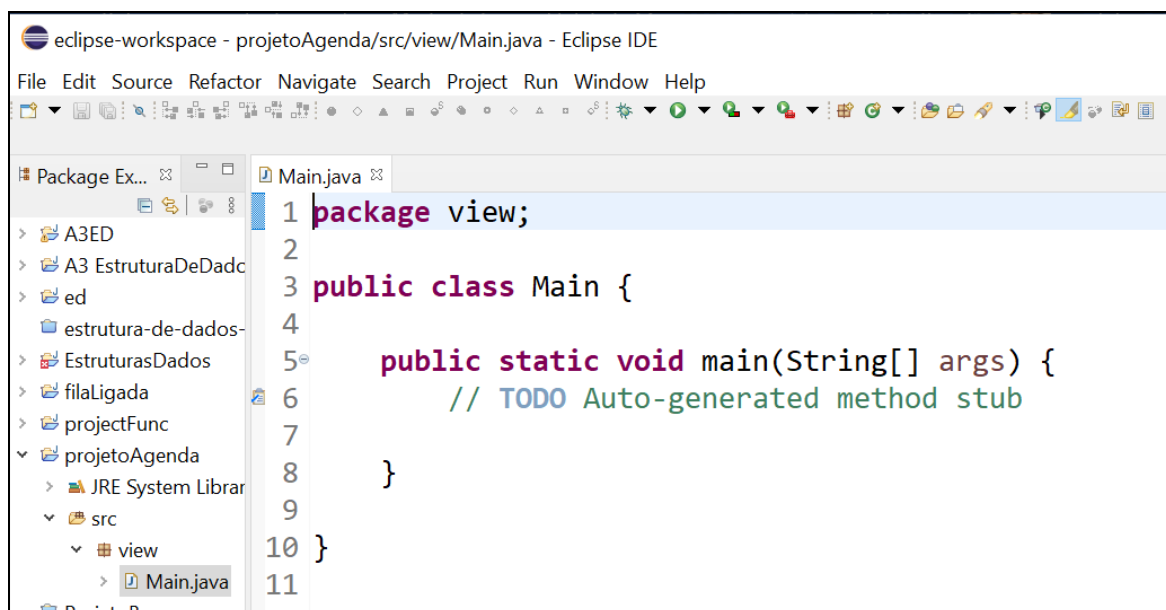
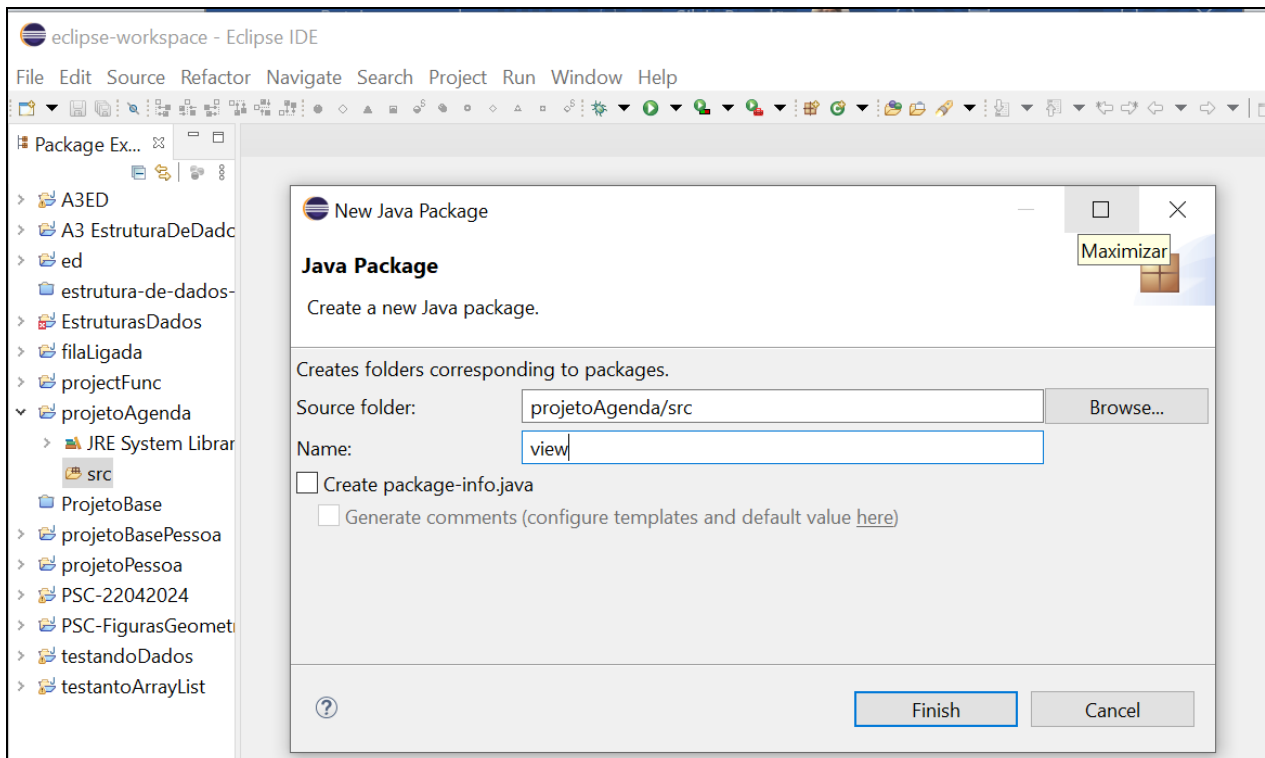


Após ter concluído a criação do Projeto Agenda, **projetoAgenda**, que deverá ser semelhante a imagem das telas acima, serão criados arquivos e pastas do projeto, os quais serão exemplificados a seguir:

- **projetoAgenda:** pasta raiz do projeto Agenda de Contatos
- **Referenced Libraries:** todas as bibliotecas utilizadas ficam nesta pasta (exemplo: o Conector/J)
- **src:** local onde irá conter todos os códigos fontes (sources) do projeto
 - **view:** conterá os arquivos de inicialização e teste da aplicação
 - **model:** conterá os arquivos com as classes da aplicação
 - **controller:** conterá o arquivo com as opções do menu para cadastrar, remover, alterar, atualizar, listar, ...(regras de negócio); assim como os serviços realizados no banco, como cadastrar, remover, alterar, atualizar, listar, ...
 - **connection:** conterá o arquivo Java para conexão com o banco de dados
 -
 - **service:** conterá os arquivos Java para os serviços realizados no banco, como cadastrar, remover, alterar, atualizar, listar, **Não teremos este pacote.**
 - **persistence:** conterá o arquivo Java para conexão com o banco de dados. **Não teremos este pacote.**

PASSO 2) CRIAÇÃO DO PACOTE VIEW

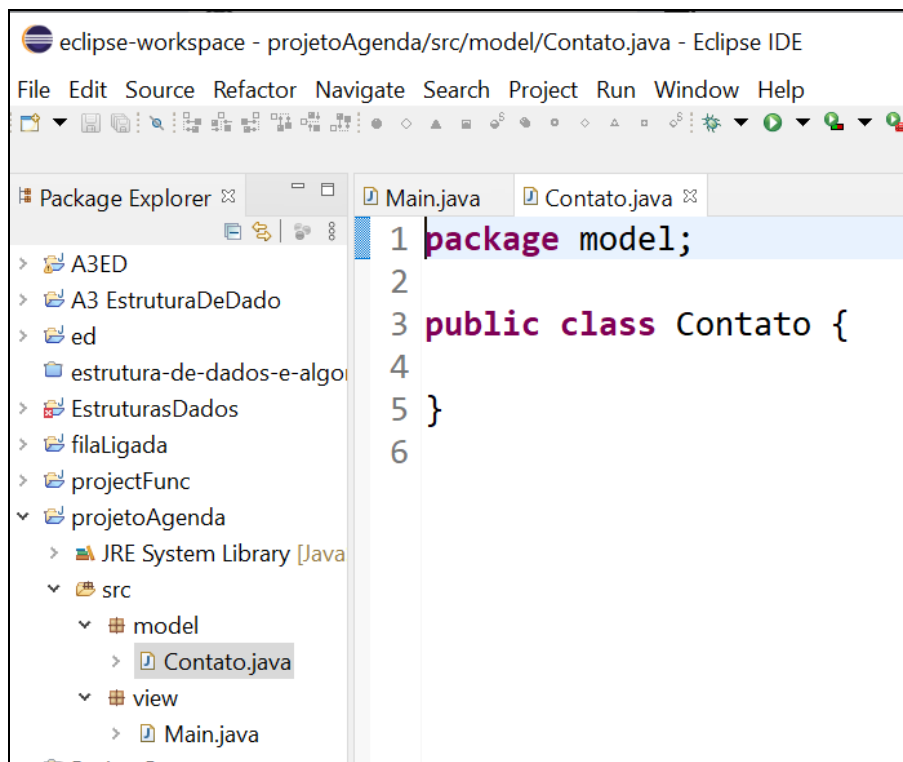
Na pasta padrão, **src**, crie o pacote **view**, em seguida crie dentro do pacote **view** a classe **Main.java** com o método **main()** que irá iniciar a execução do seu código.



A classe **Main.java** deverá ser criada por você, conforme seu entendimento do projeto.

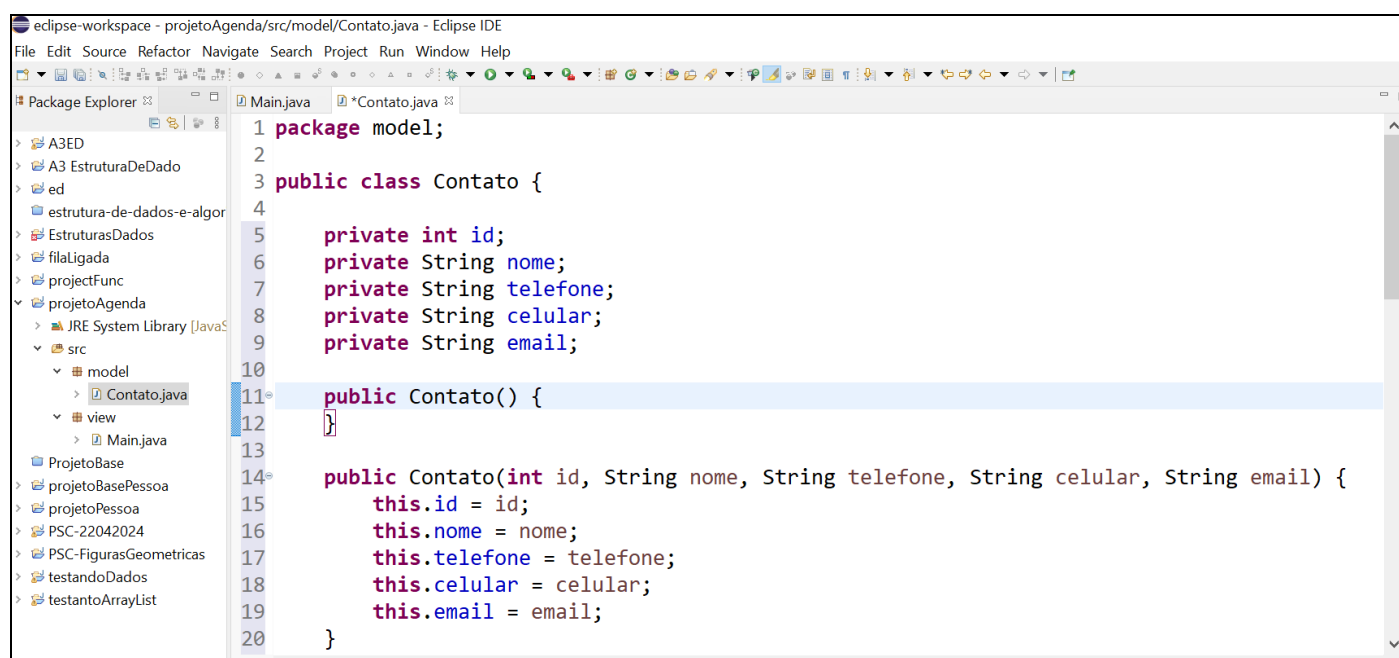
PASSO 3) CRIAÇÃO DO PACOTE MODEL OU BEANS

Na pasta padrão, **src**, crie o pacote **model**, em seguida crie dentro do pacote **model** a classe Java denominada de **Contato**.



PASSO 3.1) CRIAÇÃO DA CLASSE CONTATO

A classe Contato deverá ter os atributos privados e, os construtores listados como na figura abaixo:



Pronto! Nossa classe **Contato.java** já possui seus atributos e os métodos construtores da classe sem parâmetros e com parâmetros.

PASSO 3.2) CRIAÇÃO DOS MÉTODOS SETTERS E GETTERS

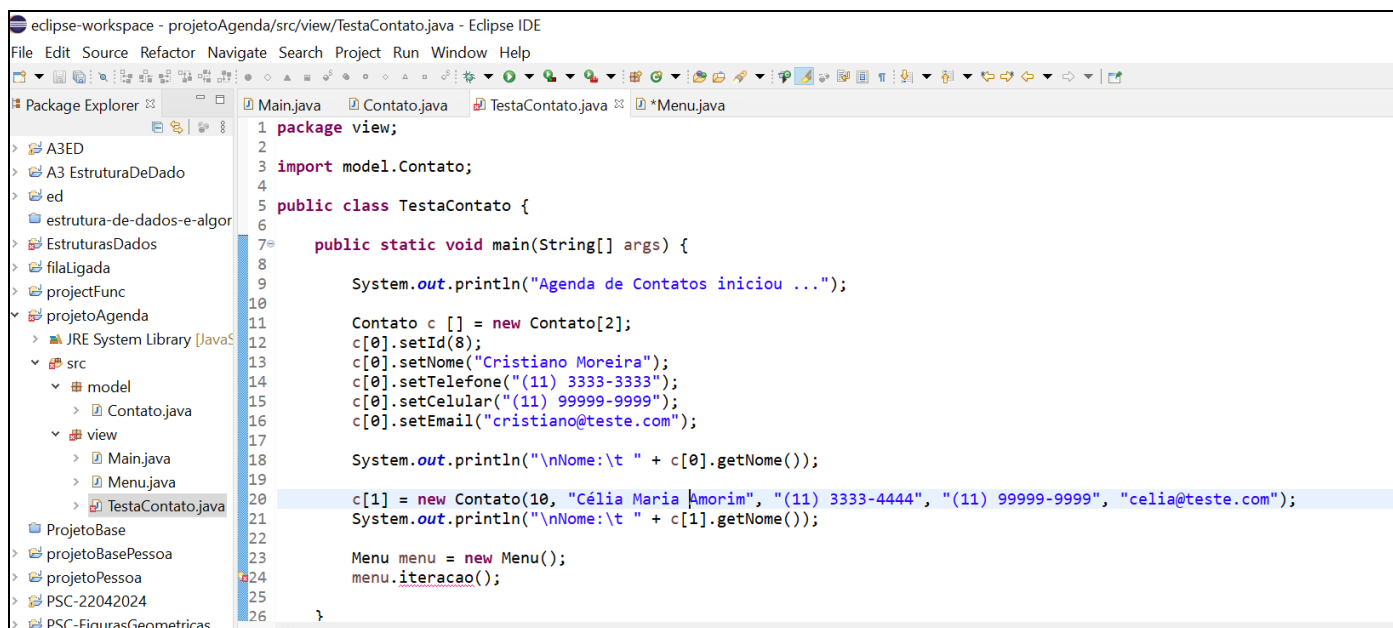
Para criar os métodos **getters** e **setters**, clique na barra de ferramentas em **Source**, depois clique em **Generate Getters and Setters**. Escolha todos os atributos como privados, o acesso público aos métodos e clique no botão para confirmar a geração.

PASSO 3.3) TESTE DA CLASSE CONTATO

Dentro do pacote **view**, criar a classe **testeContato** com o método **main()**. A classe deverá apresentar:

- criação de um vetor com 2 contatos;
- cadastro de contatos nesse vetor;
- busca sequencial de um dado contato utilizando o e-mail;
- emissão de um relatório de todas os contatos cadastrados;
- acesso ao menu de opções;
- além de outros procedimentos/métodos necessários ao teste do menu de opções.

Veja a imagem a seguir, ela apresenta parte do que foi solicitado. Complemente o que falta.



```
1 package view;
2
3 import model.Contato;
4
5 public class TestaContato {
6
7     public static void main(String[] args) {
8
9         System.out.println("Agenda de Contatos iniciou ...");
10
11         Contato c [] = new Contato[2];
12         c[0].setId(8);
13         c[0].setNome("Cristiano Moreira");
14         c[0].setTelefone("(11) 3333-3333");
15         c[0].setCelular("(11) 99999-9999");
16         c[0].setEmail("cristiano@teste.com");
17
18         System.out.println("\nNome:\t " + c[0].getNome());
19
20         c[1] = new Contato(10, "Célia Maria Amorim", "(11) 3333-4444", "(11) 99999-9999", "celia@teste.com");
21         System.out.println("\nNome:\t " + c[1].getNome());
22
23         Menu menu = new Menu();
24         menu.iteracao();
25
26     }
```

Dentro do pacote **view**, criar a classe **Menu.java** como mostra a imagem acima. A classe **Menu.java** deverá criar com os comandos **while** ou **do-while**, e **switch** (escolha) um menu de opções. Dentre elas:

1. Adicionar um contato
2. Atualizar um contato
3. Listar todos
4. Lista por parte do nome
5. Remover um usuário
6. Fechar / Sair do programa

PASSO 4) CRIAÇÃO DO PACOTE CONNECTION

Pronto! Agora já temos o modelo de dados da nossa aplicação. Então devemos criar uma conexão com o banco de dados. Você já tem o MySQL instalado em sua máquina?

PASSO 4.1) DOWNLOAD DO BANCO MySQL

Primeiro você deve realizar o Download do MySQL Server que pode ser encontrado em:

<http://dev.mysql.com/downloads/mysql/>, nessa página, você encontrará diversas versões, porém a mais comumente usada, e que é free, é a versão MySQL Community Server. O link

<https://dev.mysql.com/downloads/workbench/> te permite fazer download de uma ferramenta SGBD para trabalhar com o banco sem scripts, caso queira.

Após baixar e instalar (não é o foco desse roteiro ensinar a instalação do banco de dados e sim conectá-lo ao Java usando uma classe de conexão), seu serviço já deve estar rodando normalmente

PASSO 4.2) DOWNLOAD DO ARQUIVO DE CONEXÃO MySQL, PARA APLICAÇÕES JAVA

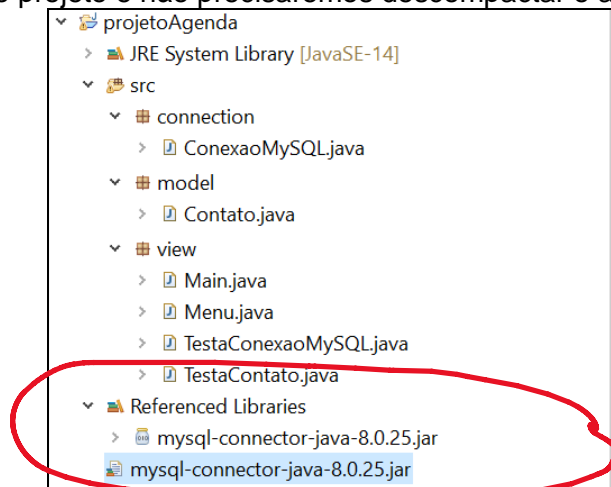
OK! Agora, precisamos de um arquivo de conexão, para aplicações Java, que no nosso caso é um driver que nos permite conectar Java + MySQL, esse driver é chamado de **mysqlconnector**. Você pode baixá-lo também do Site MySQL, <https://dev.mysql.com/downloads/connector/j/>, escolha o driver de conexão para Java, o **Connector/J**.

Por exemplo, dependendo da versão, você terá o arquivo **mysql-connector-java-8.0.25.jar**.

Agora vamos adicionar essa biblioteca, do Driver de conexão com o banco na pasta do projeto.

Localize o arquivo **.jar** do conector, onde você baixou, e direcione para sua aplicação arrastando-o. Sendo assim, arraste o arquivo para dentro do **projetoAgenda** e escolha **Copy Files, ok**. Somente assim você poderá usar o MySQL na sua aplicação.

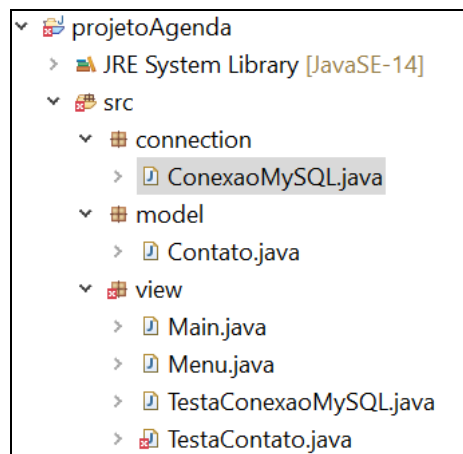
Em seguida, clique com o botão direito no arquivo **.jar**, **Buid Path** e **Add to Build Path**. Agora o conector MySQL já foi incorporado ao projeto e não precisaremos descompactar o arquivo.



PASSO 4.3) CRIAÇÃO O PACOTE CONNECTION E A CLASSE CONEXAOMYSQL

Agora que o gerenciador de banco de dados, MySQL, já está instalado e o **Conector/J** está na biblioteca do projeto, criaremos um pacote chamado **CONNECTION** e uma classe chamada **ConexãoMySQL.java** no **src** da aplicação.

Será nessa classe, **ConexãoMySQL.java**, que faremos a nossa magia de conexão com o banco de dados.



Agora vamos entender o código necessário para a conexão com banco. No código a seguir, os comentários, explicam melhor o que cada linha da classe faz.

```

package connection;

//Classes necessárias para uso de Banco de dados
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class ConexaoMySQL {

    public static String status = "Não conectou...";

    //Método Construtor da Classe
    public ConexaoMySQL() {
    }

    //Método de Conexão
    public static java.sql.Connection getInstance() {
        Connection connection = null; //atributo do tipo Connection

        try {
            //Carregando o JDBC Driver padrão
            String driverName = "com.mysql.jdbc.Driver";

            Class.forName(driverName);

            //Configurando a nossa conexão com um banco de dados
            String serverName = "localhost"; //caminho do servidor do BD
            String mydatabase = "agendaContatos"; //nome do seu banco de dados
            String url = "jdbc:mysql://" + serverName + "/" + mydatabase;
            String username = "root"; //nome de um usuário de seu BD
            String password = "123456"; //sua senha de acesso

            connection = DriverManager.getConnection(url, username, password);

            //Testa sua conexão
            if (connection != null) {
                status = ("STATUS--->Conectado com sucesso!");
            } else {
                status = ("STATUS--->Não foi possível realizar conexão");
            }
            return connection;
        } catch (ClassNotFoundException e) { //Driver não encontrado
            System.out.println("O driver especificado não foi encontrado.");

            return null;
        } catch (SQLException e) {
            //Não conseguindo se conectar ao banco
            System.out.println("Nao foi possivel conectar ao Banco de Dados.");

            return null;
        }
    }

    //Método que retorna o status da sua conexão//
    public static String statusConection() {

        return status;
    }

    //Método que fecha sua conexão
    public static boolean FecharConexao() {

```

```

    try {
        ConexaoMySQL.getInstance().close();

        return true;
    } catch (SQLException e) {

        return false;
    }
}

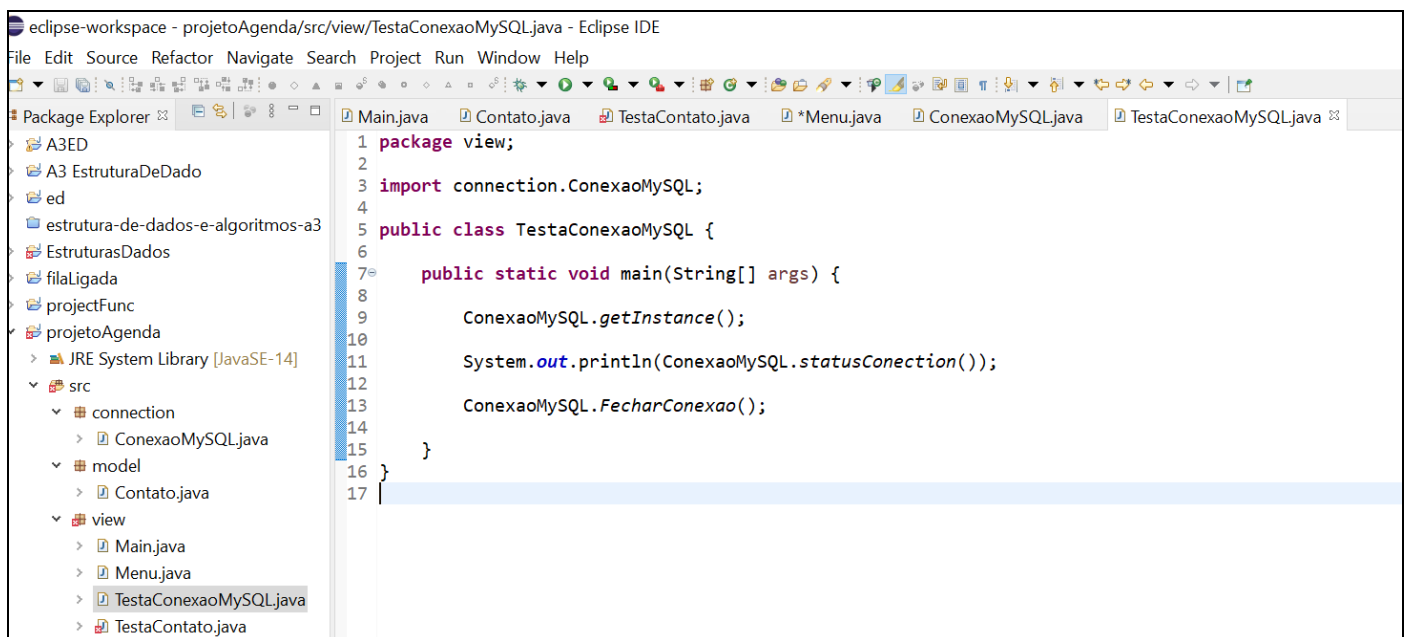
//Método que reinicia sua conexão
public static java.sql.Connection ReiniciarConexao() {
    FecharConexao();

    return ConexaoMySQL.getInstance();
}
}

```

PASSO 4.4) TESTE DA CLASSE CONEXAOMYSQL

Dentro do pacote **view**, criar a classe **testeConexaoMySQL** com o método **main()**.



PASSO 5) CLASSE MENU - OPÇÕES DO MENU: inserção e atualização

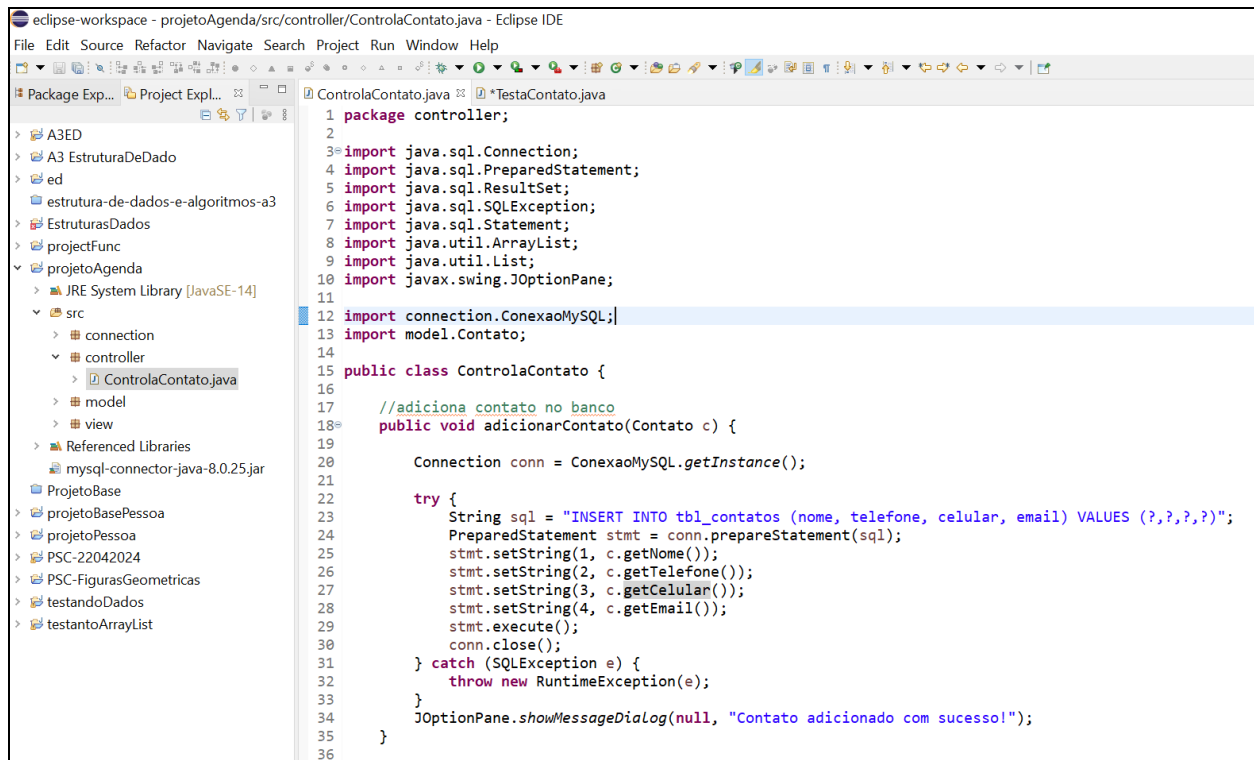
Dando continuidade a implementação da classe **Menu.java**, implemente os itens 1 e 2 do menu abaixo.

Menu:

1. Adicionar um contato
2. Atualizar um contato
3. Listar todos
4. Lista por parte do nome
5. Remover um usuário
6. Fechar / Sair do programa

Lembre-se, o cadastro deve ser realizado no banco de dados, isto é **gravado no banco**.

Na pasta padrão, **src**, crie o pacote **controller**. Em seguida, dentro do pacote **controller** crie a classe **ControlaContato.java** que irá se responsabilizar pelo CRUD no banco. Veja a imagem:



Devemos usar o **java.sql.PreparedStatement** em todos os tipos de comandos SQL (INSERT, UPDATE, DELETE, SELECT), pois é mais performático que **java.sql.Statement**.

Crie uma interface intuitiva e amigável para os usuários inserirem e visualizarem os dados.

Para as opções <1> e <2> do Menu, você deve criar os métodos adicionarContato() e atualizarContato() para respectivamente, inserir "INSERT INTO tbl_contatos (nome, telefone, celular, email) VALUES (?, ?, ?, ?)" e atualizar "UPDATE tbl_contatos SET nome = ?, telefone = ?, celular = ?, email = ? WHERE id = ?" os registros nas tabelas do banco. São eles:

```
//adiciona contato no banco
public void adicionarContato(Contato c) {

    Connection conn = ConexaoMySQL.getInstance();

    try {
        String sql = "INSERT INTO tbl_contatos (nome, telefone, celular, email)
VALUES (?, ?, ?, ?)";
        PreparedStatement stmt = conn.prepareStatement(sql);
        stmt.setString(1, c.getNome());
        stmt.setString(2, c.getTelefone());
        stmt.setString(3, c.getCelular());
        stmt.setString(4, c.getEmail());
        stmt.execute();
        conn.close();
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    JOptionPane.showMessageDialog(null, "Contato adicionado com sucesso!");
}

//atualiza contato no banco
public void atualizarContato(Contato c) {

    Connection conn = ConexaoMySQL.getInstance();
```



```

        try {
            String sql = "UPDATE tbl_contatos SET nome = ?, telefone = ?, celular = ?,
email = ? WHERE id = ?";
            PreparedStatement stmt = conn.prepareStatement(sql);
            stmt.setString(1, c.getNome());
            stmt.setString(2, c.getTelefone());
            stmt.setString(3, c.getCelular());
            stmt.setString(4, c.getEmail());
            stmt.setInt(5, c.getId());
            stmt.execute();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
    JOptionPane.showMessageDialog(null, "Contato atualizado com sucesso!");
}

```

2ª. ETAPA

No próximo roteiro, faremos as demais opções do Menu.