

## Mapeamento Objeto-Relacional e DAO

Aluno: Vinícius Silva de Lima – 2023 0707 6309

Campus – Polo Vilar dos Teles – São João de Meriti – RJ

RPG0016 – Back-End sem banco não tem – 2024.3

### Objetivo da Prática

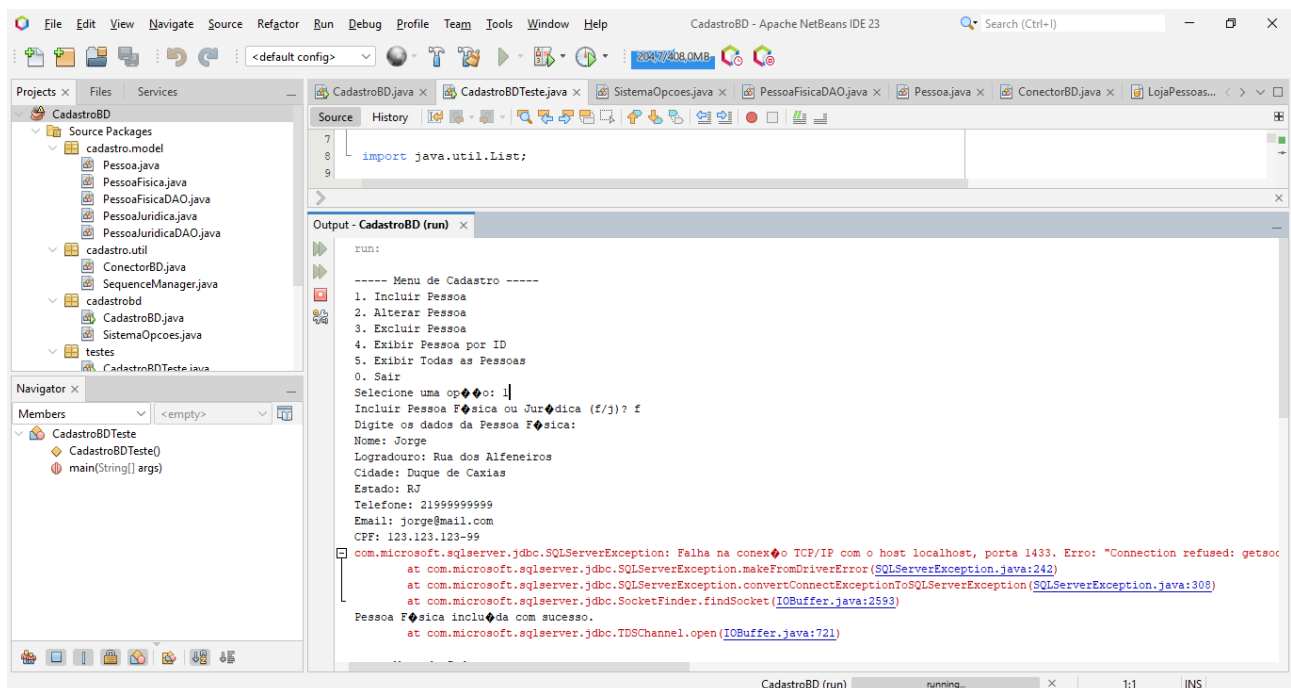
- Implementar persistência com base no middleware JDBC.
- Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
- Implementar o mapeamento objeto-relacional em sistemas Java.
- Criar sistemas cadastrais com persistência em banco relacional.
- No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

Os códigos solicitados estão no repositório:

<https://github.com/DevViniciusLima/CadastroBD>

### 1º Procedimento:

Resultados da execução dos códigos

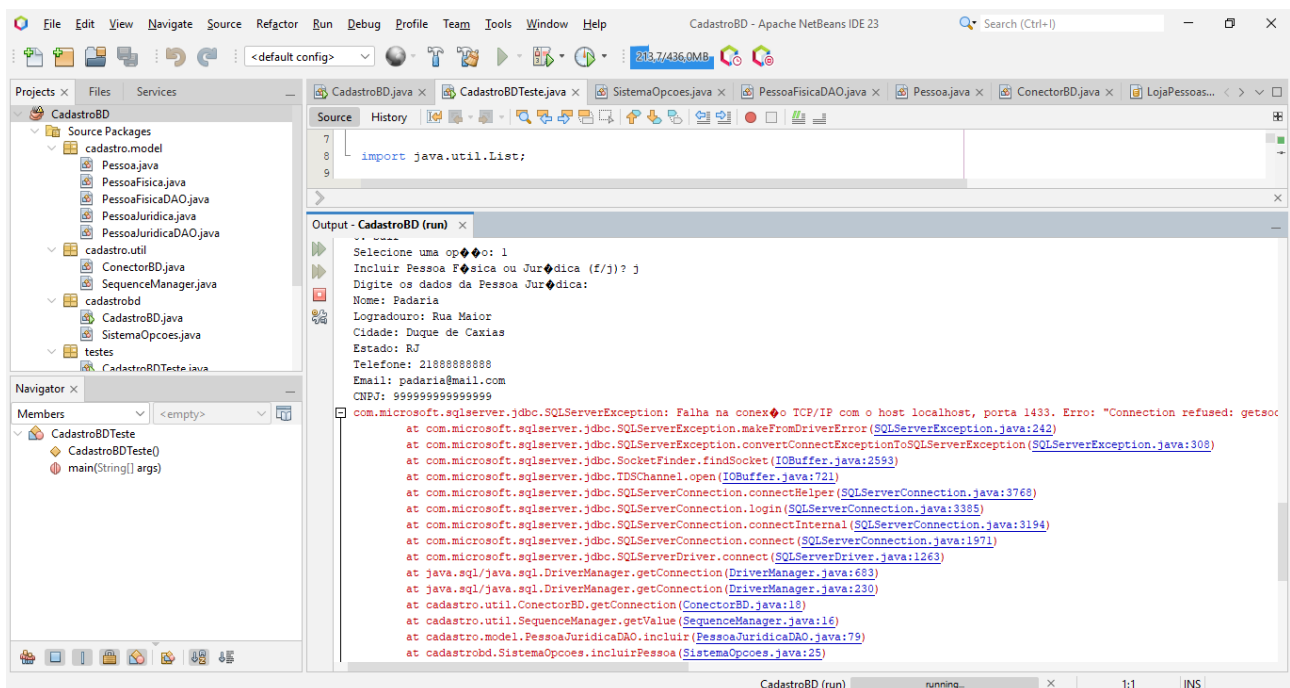
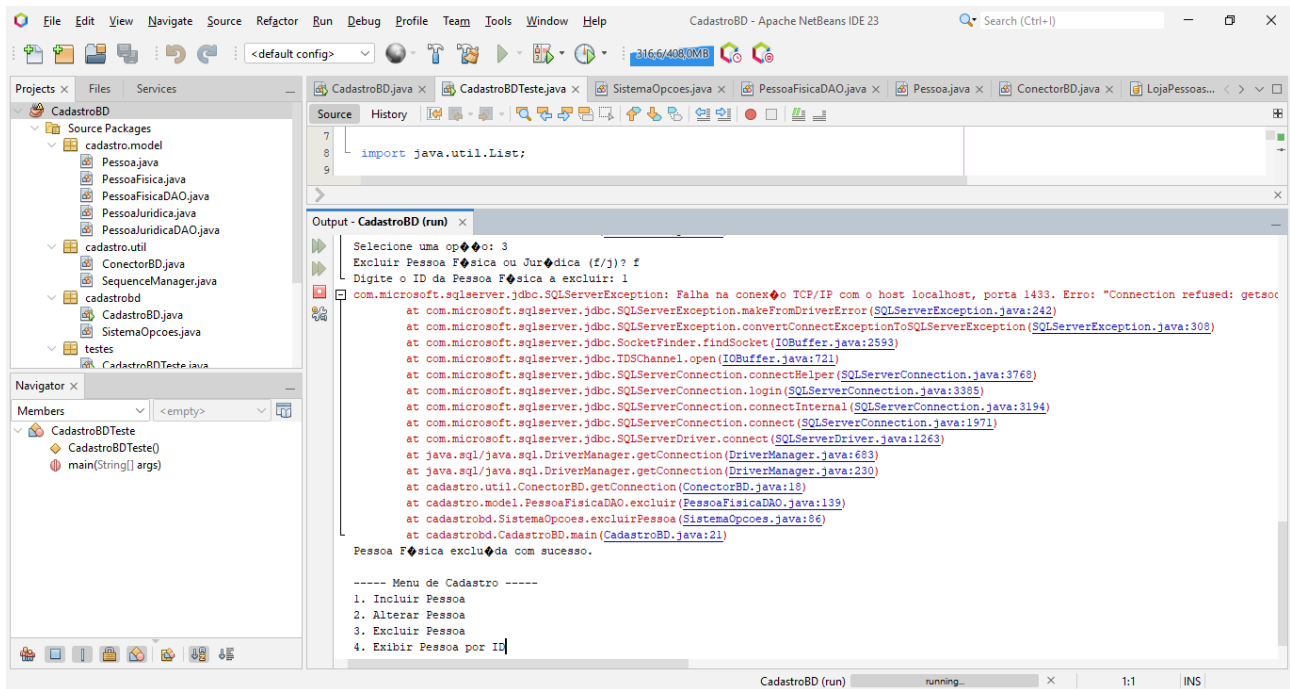


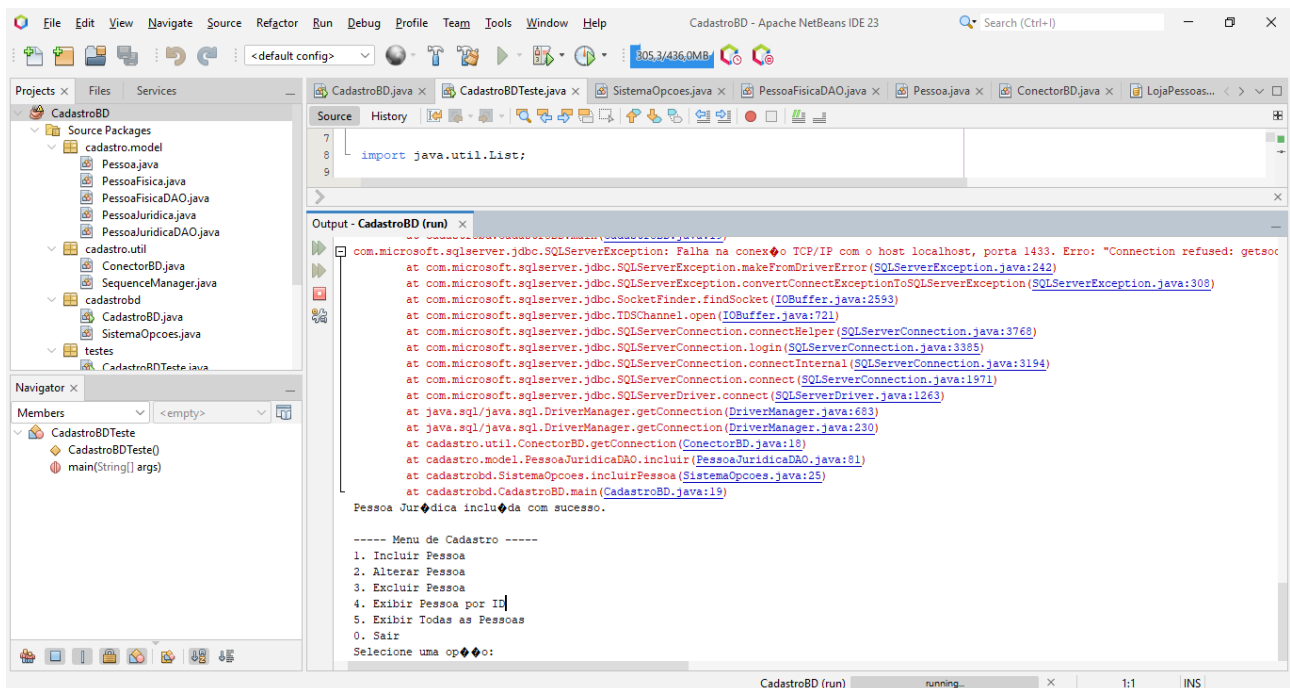
```
import java.util.List;
```

run:

```
----- Menu de Cadastro -----
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Pessoa por ID
5. Exibir Todas as Pessoas
0. Sair
Selecione uma opção: 1
Incluir Pessoa Física ou Jurídica (f/j)? f
Digite os dados da Pessoa Física:
Nome: Jorge
Logradouro: Rua dos Alfeneiros
Cidade: Duque de Caxias
Estado: RJ
Telefone: 2199999999
Email: jorge@mail.com
CPF: 123.123.123-99

com.microsoft.sqlserver.jdbc.SQLServerException: Falha na conexão TCP/IP com o host localhost, porta 1433. Error: "Connection refused: getsock
at com.microsoft.sqlserver.jdbc.SQLServerException.makeFromDriverError(SQLServerException.java:242)
at com.microsoft.sqlserver.jdbc.SQLServerException.convertConnectExceptionToSQLServerException(SQLServerException.java:308)
at com.microsoft.sqlserver.jdbc.SocketFinder.findSocket(IOBuffer.java:2593)
Pessoa Física incluída com sucesso.
at com.microsoft.sqlserver.jdbc.TDSCChannel.open(IOBuffer.java:721)
```





Análise e conclusão:

a) Qual a importância dos componentes de middleware, como o JDBC?

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel essencial em aplicações que precisam interagir com bancos de dados. O JDBC atua como uma ponte que permite que o código Java se conecte a diferentes bancos de dados de maneira uniforme. Isso é importante porque o desenvolvedor pode escrever código de manipulação de dados em Java sem se preocupar com os detalhes específicos de cada banco de dados. Ele facilita operações como conectar-se ao banco, executar consultas e recuperar resultados, proporcionando uma camada de abstração entre a aplicação e o banco de dados. Em resumo, o JDBC torna a comunicação com bancos de dados mais simples, portátil e padronizada.

b) Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?

Quando se trata de manipulação de dados em Java via JDBC, as diferenças entre Statement e PreparedStatement são relevantes para desempenho e segurança:

Statement: Utilizado para executar consultas SQL simples e estáticas. Ele é adequado para quando você tem comandos SQL que não variam. No entanto, pode ser inseguro em relação a SQL Injection, porque o comando SQL é passado diretamente como uma string, o que permite que um usuário mal-intencionado insira código perigoso.

PreparedStatement: É uma versão mais robusta, onde o SQL é pré-compilado e você pode definir parâmetros dinamicamente. Além de ser mais eficiente (pois a consulta é compilada apenas uma vez e pode ser reutilizada), ele é muito mais seguro, pois trata automaticamente qualquer entrada do usuário, prevenindo ataques de SQL Injection.

c) Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (Data Access Object) é uma prática de design que isola a lógica de acesso ao banco de dados do restante da aplicação. Isso significa que todo o código relacionado a operações de banco de dados (como conexões, queries, updates) fica concentrado em

uma camada separada. A principal vantagem é a manutenibilidade: qualquer mudança na forma como os dados são armazenados ou acessados (por exemplo, mudança de um banco de dados relacional para um NoSQL) pode ser feita exclusivamente na camada DAO, sem precisar alterar a lógica de negócios da aplicação.

Além disso, o DAO favorece a reutilização de código e facilita o teste, uma vez que você pode simular ou "mockar" a camada de acesso a dados para testar outras partes da aplicação.

d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Quando se lida com um modelo estritamente relacional em banco de dados, a herança geralmente é refletida com uso de uma técnica conhecida como "tabelas de junção" (ou "tabelas de associação"). Esta abordagem é chamada de modelagem de herança de tabela única, ou modelagem de herança de tabela por classe. Eis como funciona:

- Tabela Base (ou Superclasse): Uma tabela é criada para representar a classe base ou superclasse. Esta tabela contém os atributos comuns a todas as subclasses.

- Tabelas de Subclasse: Para cada subclasse, é criada uma tabela separada contendo apenas os atributos específicos daquela subclasse. Essas tabelas também terão uma chave estrangeira que referencia a tabela base.

- Chave Estrangeira: A chave primária da tabela base é usada como chave estrangeira nas tabelas de subclasse para estabelecer a relação entre elas.

- Junção de Tabelas: Quando uma consulta é feita para recuperar dados de uma hierarquia de herança, é necessário fazer uma junção (JOIN) entre a tabela base e as tabelas de subclasse usando as chaves primárias e estrangeiras correspondentes.

## 2º Procedimento:

a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

Persistência em arquivo: É o processo de armazenar dados diretamente em arquivos no sistema. Pode ser feita de forma simples (usando arquivos de texto, JSON, CSV, etc.) ou mais estruturada (arquivos binários). No entanto, a organização, consulta e manipulação de dados em arquivos pode ser manual e menos eficiente. Não há uma estrutura padrão para gerenciar esses dados, como consultas complexas ou transações simultâneas.

Persistência em banco de dados: Envolve armazenar dados em um sistema de banco de dados (como MySQL, PostgreSQL, etc.). Os bancos de dados são projetados para gerenciar grandes volumes de dados de maneira eficiente, com suporte a consultas avançadas (SQL), transações, controle de concorrência e recuperação de falhas. Eles são mais escaláveis e adequados para aplicações que exigem consistência, integridade e segurança dos dados.

b) Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

O uso de operadores lambda, a partir do Java 8, simplificou a impressão de valores contidos em entidades, ao permitir que desenvolvedores escrevam código mais legível e

conciso, sem a necessidade de criar classes anônimas para operações simples. Com operadores lambda, é possível passar comportamentos de forma direta e declarativa. Por exemplo, para imprimir os valores de uma lista, antes do Java 8, seria necessário criar um loop explícito. Com lambdas, isso pode ser feito de maneira simplificada usando métodos de stream e operações como **forEach**

*lista.forEach(elemento -> System.out.println(elemento));*

Esse código substitui várias linhas de um loop for ou loop aprimorado for-each, tradicionais, ao mesmo tempo que torna o código mais expressivo, focado no que realmente se deseja realizar (a ação de imprimir), e não em como realizar o loop ou controle de fluxo.

c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?

O método main é o primeiro método a ser executado quando uma classe Java é compilada e executada, caso, obviamente, a classe contenha um método main. A principal razão pela qual o método main é definido como estático, é justamente para conceder a possibilidade de ser diretamente executado sem necessidade de instanciar um objeto do tipo da classe, para depois invocar o método main, ou seja, uma comodidade para evitar código desnecessário.

Por definição, atributos ou métodos estáticos pertencem à classe onde são definidos, e não ao objeto instanciado a partir dessa classe. Assim, se o atributo ou método forem públicos, podem ser referenciados externamente por outras classes, através somente do nome da classe e não por objeto. Contudo, métodos estáticos, em suas implementações, ao tentar invocar outros métodos sem uso de objetos de referência, somente podem invocar diretamente outros métodos estáticos. Logo, ambos métodos precisam ser estáticos, tanto o método que invoca o outro método, como o método invocado.

## Códigos do projeto

```
package cadastrobd;
```

```
import java.util.Scanner;
```

```
public class CadastroBD {
    private static final Scanner scanner = new Scanner(System.in);
    private static final SistemaOpcoes operacoes = new SistemaOpcoes();

    public static void main(String[] args) {
        try (scanner) {
            int opcao;

            do {
                exibirMenu();
                opcao = scanner.nextInt();
                scanner.nextLine();

                switch (opcao) {
                    case 1 -> operacoes.incluirPessoa(scanner);
                    case 2 -> operacoes.alterarPessoa(scanner);
                    case 3 -> operacoes.excluirPessoa(scanner);
                    case 4 -> operacoes.exibirPessoaPorId(scanner);
                    case 5 -> operacoes.exibirTodasPessoas(scanner);
                    case 0 -> System.out.println("Finalizando o sistema...");
                }
            } while (opcao != 0);
        }
    }
}
```

```

        default -> System.out.println("Opção inválida, tente novamente.");
    }

    } while (opcao != 0);
}

private static void exibirMenu() {
    System.out.println("\n----- Menu de Cadastro -----");
    System.out.println("1. Incluir Pessoa");
    System.out.println("2. Alterar Pessoa");
    System.out.println("3. Excluir Pessoa");
    System.out.println("4. Exibir Pessoa por ID");
    System.out.println("5. Exibir Todas as Pessoas");
    System.out.println("0. Sair");
    System.out.print("Selecione uma opção: ");
}
}

```

```
package testes;
```

```
import cadastro.model.PessoaFisica;
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridica;
import cadastro.model.PessoaJuridicaDAO;
```

```
import java.util.List;
```

```
public class CadastroBDTeste {
    public static void main(String[] args) {
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

        // 1° instanciar uma pessoa física e persistir no banco de dados
        PessoaFisica pessoaFisica = new PessoaFisica(0, "João Silva", "Rua 1", "Cidade A", "Estado
X", "123456789", "joao@email.com", "123.456.789-00");
        pessoaFisicaDAO.incluir(pessoaFisica);
        System.out.println("Pessoa Física criada: " + pessoaFisica.getNome());

        // 2° alterar os dados da pessoa física no banco
        pessoaFisica.setNome("João Silva Alterado");
        pessoaFisica.setCidade("Cidade B");
        pessoaFisicaDAO.alterar(pessoaFisica);
        System.out.println("Pessoa Física alterada: " + pessoaFisica.getNome());

        // 3° consultar todas as pessoas físicas do banco de dados e listar
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        System.out.println("\nLista de Pessoas Físicas:");
        for (PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
        }

        // 4° exclusão da pessoa física criada anteriormente
    }
}

```

```

        pessoaFisicaDAO.excluir(pessoaFisica.getId());
        System.out.println("\nPessoa Física excluída: " + pessoaFisica.getNome());

        // 5° instanciar uma pessoa jurídica e persistir no banco de dados
        PessoaJuridica pessoaJuridica = new PessoaJuridica(0, "Empresa XYZ", "Avenida Central",
"Cidade C", "Estado Y", "987654321", "contato@empresa.com", "00.123.456/0001-00");
        pessoaJuridicaDAO.incluir(pessoaJuridica);
        System.out.println("\nPessoa Jurídica criada: " + pessoaJuridica.getNome());

        // 6° alterar os dados da pessoa jurídica no banco
        pessoaJuridica.setNome("Empresa XYZ Alterada");
        pessoaJuridica.setCidade("Cidade D");
        pessoaJuridicaDAO.alterar(pessoaJuridica);
        System.out.println("\nPessoa Jurídica alterada: " + pessoaJuridica.getNome());

        // 7° Consultar todas as pessoas jurídicas do banco de dados e listar
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        System.out.println("\nLista de Pessoas Jurídicas:");
        for (PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
        }

        // 8° exclusão da pessoa jurídica criada anteriormente
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());
        System.out.println("\nPessoa Jurídica excluída: " + pessoaJuridica.getNome());
    }
}

```

```

package cadastro.model;

```

```

import java.io.Serializable;

```

```

public class Pessoa implements Serializable {
    private int id;
    private String nome;
    private String logradouro;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;

    public Pessoa() {}

    public Pessoa(int id, String nome, String logradouro, String cidade, String estado, String
telefone, String email) {
        this.id = id;
        this.nome = nome;
        this.logradouro = logradouro;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }
}

```

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getLogradouro() {
    return logradouro;
}

public void setLogradouro(String logradouro) {
    this.logradouro = logradouro;
}

public String getCidade() {
    return cidade;
}

public void setCidade(String cidade) {
    this.cidade = cidade;
}

public String getEstado() {
    return estado;
}

public void setEstado(String estado) {
    this.estado = estado;
}

public String getTelefone() {
    return telefone;
}

public void setTelefone(String telefone) {
    this.telefone = telefone;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public void exibir() {
```



```

        System.out.println(this.toString());
    }

    @Override
    public String toString() {
        return "Pessoa{" +
            "id=" + id +
            ", nome=" + nome + "\" +
            ", logradouro=" + logradouro + "\" +
            ", cidade=" + cidade + "\" +
            ", estado=" + estado + "\" +
            ", telefone=" + telefone + "\" +
            ", email=" + email + "\" +
            '}';
    }
}

```

```

package cadastro.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {
        super();
    }

    public PessoaFisica(int id, String nome, String logradouro, String cidade, String estado, String
telefone, String email, String cpf) {
        super(id, nome, logradouro, cidade, estado, telefone, email);
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf);
    }

    @Override
    public String toString() {
        return super.toString() + ", cpf=" + cpf + "\" + '}';
    }
}

```

```

package cadastro.model;

import cadastro.util.ConectorBD;
import cadastro.util.SequenceManager;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    private final ConectorBD conectorBD = new ConectorBD();
    private final SequenceManager sequenceManager = new SequenceManager();

    // obter uma pessoa física através do id (findById(int id))
    public PessoaFisica getPessoa(int id) {
        String sql = "SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.id = pf.id WHERE p.id = ?";
        PessoaFisica pessoaFisica = null;

        try (Connection connection = conectorBD.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setInt(1, id);
            ResultSet resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                pessoaFisica = new PessoaFisica(
                    resultSet.getInt("id"),
                    resultSet.getString("nome"),
                    resultSet.getString("logradouro"),
                    resultSet.getString("cidade"),
                    resultSet.getString("estado"),
                    resultSet.getString("telefone"),
                    resultSet.getString("email"),
                    resultSet.getString("cpf")
                );
            }
            conectorBD.close(resultSet);
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return pessoaFisica;
    }

    // obter todas as pessoas físicas (findAll())
    public List<PessoaFisica> getPessoas() {
        String sql = "SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.id = pf.id";
        List<PessoaFisica> pessoasFisicas = new ArrayList<>();

        try (Connection connection = conectorBD.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery()) {

            while (resultSet.next()) {
                PessoaFisica pessoaFisica = new PessoaFisica(
                    resultSet.getInt("id"),

```

```

        resultSet.getString("nome"),
        resultSet.getString("logradouro"),
        resultSet.getString("cidade"),
        resultSet.getString("estado"),
        resultSet.getString("telefone"),
        resultSet.getString("email"),
        resultSet.getString("cpf")
    );
    pessoasFisicas.add(pessoaFisica);
}
} catch (SQLException e) {
    e.printStackTrace();
}

return pessoasFisicas;
}

// adicionar no banco
public void incluir(PessoaFisica pessoaFisica) {
    String sqlPessoa = "INSERT INTO Pessoa (id, nome, logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
    String sqlPessoaFisica = "INSERT INTO PessoaFisica (id, cpf) VALUES (?, ?)";

    int id = sequenceManager.getValue("PessoaSequence");

    try (Connection connection = conectorBD.getConnection();
        PreparedStatement preparedStatementPessoa =
            connection.prepareStatement(sqlPessoa);
        PreparedStatement preparedStatementPessoaFisica =
            connection.prepareStatement(sqlPessoaFisica)) {

        // insere primeiro em pessoa
        preparedStatementPessoa.setInt(1, id);
        preparedStatementPessoa.setString(2, pessoaFisica.getNome());
        preparedStatementPessoa.setString(3, pessoaFisica.getLogradouro());
        preparedStatementPessoa.setString(4, pessoaFisica.getCidade());
        preparedStatementPessoa.setString(5, pessoaFisica.getEstado());
        preparedStatementPessoa.setString(6, pessoaFisica.getTelefone());
        preparedStatementPessoa.setString(7, pessoaFisica.getEmail());
        preparedStatementPessoa.executeUpdate();

        //insere para pessoa física
        preparedStatementPessoaFisica.setInt(1, id);
        preparedStatementPessoaFisica.setString(2, pessoaFisica.getCpf());
        preparedStatementPessoaFisica.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// alterar um registro
public void alterar(PessoaFisica pessoaFisica) {
    String sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado = ?, telefone = ?, email = ? WHERE id = ?";
    String sqlPessoaFisica = "UPDATE PessoaFisica SET cpf = ? WHERE id = ?";

```

```

        try (Connection connection = conectorBD.getConnection());
            PreparedStatement preparedStatementPessoa =
connection.prepareStatement(sqlPessoa);
            PreparedStatement preparedStatementPessoaFisica =
connection.prepareStatement(sqlPessoaFisica)) {

                // altera primeiro na table pessoa
                preparedStatementPessoa.setString(1, pessoaFisica.getNome());
                preparedStatementPessoa.setString(2, pessoaFisica.getLogradouro());
                preparedStatementPessoa.setString(3, pessoaFisica.getCidade());
                preparedStatementPessoa.setString(4, pessoaFisica.getEstado());
                preparedStatementPessoa.setString(5, pessoaFisica.getTelefone());
                preparedStatementPessoa.setString(6, pessoaFisica.getEmail());
                preparedStatementPessoa.setInt(7, pessoaFisica.getId());
                preparedStatementPessoa.executeUpdate();

                // altera na table pessoa fisica
                preparedStatementPessoaFisica.setString(1, pessoaFisica.getCpf());
                preparedStatementPessoaFisica.setInt(2, pessoaFisica.getId());
                preparedStatementPessoaFisica.executeUpdate();

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        // exclusão
        public void excluir(int id) {
            String sqlPessoaFisica = "DELETE FROM PessoaFisica WHERE id = ?";
            String sqlPessoa = "DELETE FROM Pessoa WHERE id = ?";

            try (Connection connection = conectorBD.getConnection());
                PreparedStatement preparedStatementPessoaFisica =
connection.prepareStatement(sqlPessoaFisica);
                PreparedStatement preparedStatementPessoa =
connection.prepareStatement(sqlPessoa)) {

                    // exclusão inicial na tabela filho (pessoa fisica)
                    preparedStatementPessoaFisica.setInt(1, id);
                    preparedStatementPessoaFisica.executeUpdate();

                    // exclusão na tabela pai (pessoa)
                    preparedStatementPessoa.setInt(1, id);
                    preparedStatementPessoa.executeUpdate();

                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

```

```
package cadastro.model;
```

```
public class PessoaJuridica extends Pessoa {
```

```

private String cnpj;

public PessoaJuridica() {
    super();
}

public PessoaJuridica(int id, String nome, String logradouro, String cidade, String estado, String
telefone, String email, String cnpj) {
    super(id, nome, logradouro, cidade, estado, telefone, email);
    this.cnpj = cnpj;
}

public String getCnpj() {
    return cnpj;
}

public void setCnpj(String cnpj) {
    this.cnpj = cnpj;
}

@Override
public void exibir() {
    super.exibir();
    System.out.println("CNPJ: " + cnpj);
}

@Override
public String toString() {
    return super.toString() + ", cnpj=\"" + cnpj + "\"" + '\n';
}
}

```

```

package cadastro.model;

import cadastro.util.ConectorBD;
import cadastro.util.SequenceManager;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaDAO {
    private final ConectorBD conectorBD = new ConectorBD();
    private final SequenceManager sequenceManager = new SequenceManager();

    // obter uma pessoa juridica através do id (findById(int id))
    public PessoaJuridica getPessoa(int id) {
        String sql = "SELECT * FROM Pessoa p JOIN PessoaJuridica pj ON p.id = pj.id WHERE p.id
= ?";
        PessoaJuridica pessoaJuridica = null;

        try (Connection connection = conectorBD.getConnection();
            PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

```

```

        preparedStatement.setInt(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        if (resultSet.next()) {
            pessoaJuridica = new PessoaJuridica(
                resultSet.getInt("id"),
                resultSet.getString("nome"),
                resultSet.getString("logradouro"),
                resultSet.getString("cidade"),
                resultSet.getString("estado"),
                resultSet.getString("telefone"),
                resultSet.getString("email"),
                resultSet.getString("cnpj")
            );
        }
        conectorBD.close(resultSet);
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return pessoaJuridica;
}

// obter todas as pessoas juridicas (findAll())
public List<PessoaJuridica> getPessoas() {
    String sql = "SELECT * FROM Pessoa p JOIN PessoaJuridica pj ON p.id = pj.id";
    List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

    try (Connection connection = conectorBD.getConnection();
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        ResultSet resultSet = preparedStatement.executeQuery()) {

        while (resultSet.next()) {
            PessoaJuridica pessoaJuridica = new PessoaJuridica(
                resultSet.getInt("id"),
                resultSet.getString("nome"),
                resultSet.getString("logradouro"),
                resultSet.getString("cidade"),
                resultSet.getString("estado"),
                resultSet.getString("telefone"),
                resultSet.getString("email"),
                resultSet.getString("cnpj")
            );
            pessoasJuridicas.add(pessoaJuridica);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return pessoasJuridicas;
}

// adicionar no banco
public void incluir(PessoaJuridica pessoaJuridica) {
    String sqlPessoa = "INSERT INTO Pessoa (id, nome, logradouro, cidade, estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";

```

```

String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (id, cnpj) VALUES (?, ?)";

int id = sequenceManager.getValue("PessoaSequence");

try (Connection connection = conectorBD.getConnection();
    PreparedStatement preparedStatementPessoa =
connection.prepareStatement(sqlPessoa);
    PreparedStatement preparedStatementPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {

    // insere primeiro em pessoa
    preparedStatementPessoa.setInt(1, id);
    preparedStatementPessoa.setString(2, pessoaJuridica.getNome());
    preparedStatementPessoa.setString(3, pessoaJuridica.getLogradouro());
    preparedStatementPessoa.setString(4, pessoaJuridica.getCidade());
    preparedStatementPessoa.setString(5, pessoaJuridica.getEstado());
    preparedStatementPessoa.setString(6, pessoaJuridica.getTelefone());
    preparedStatementPessoa.setString(7, pessoaJuridica.getEmail());
    preparedStatementPessoa.executeUpdate();

    //insere em pessoa jurídica
    preparedStatementPessoaJuridica.setInt(1, id);
    preparedStatementPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
    preparedStatementPessoaJuridica.executeUpdate();

} catch (SQLException e) {
    e.printStackTrace();
}

}

// alterar um registro
public void alterar(PessoaJuridica pessoaJuridica) {
    String sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado = ?,
telefone = ?, email = ? WHERE id = ?";
    String sqlPessoaJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE id = ?";

    try (Connection connection = conectorBD.getConnection();
        PreparedStatement preparedStatementPessoa =
connection.prepareStatement(sqlPessoa);
        PreparedStatement preparedStatementPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {

        // altera primeiro na tabela pessoa
        preparedStatementPessoa.setString(1, pessoaJuridica.getNome());
        preparedStatementPessoa.setString(2, pessoaJuridica.getLogradouro());
        preparedStatementPessoa.setString(3, pessoaJuridica.getCidade());
        preparedStatementPessoa.setString(4, pessoaJuridica.getEstado());
        preparedStatementPessoa.setString(5, pessoaJuridica.getTelefone());
        preparedStatementPessoa.setString(6, pessoaJuridica.getEmail());
        preparedStatementPessoa.setInt(7, pessoaJuridica.getId());
        preparedStatementPessoa.executeUpdate();

        // altera na tabela pessoa jurídica
        preparedStatementPessoaJuridica.setString(1, pessoaJuridica.getCnpj());
        preparedStatementPessoaJuridica.setInt(2, pessoaJuridica.getId());
        preparedStatementPessoaJuridica.executeUpdate();
    }
}

```

```

    } catch (SQLException e) {
        e.printStackTrace();
    }
}

// exclusão
public void excluir(int id) {
    String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE id = ?";
    String sqlPessoa = "DELETE FROM Pessoa WHERE id = ?";

    try (Connection connection = conectorBD.getConnection();
        PreparedStatement preparedStatementPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica);
        PreparedStatement preparedStatementPessoa =
connection.prepareStatement(sqlPessoa)) {

        // exclusão inicial na tabela filho (pessoa jurídica)
        preparedStatementPessoaJuridica.setInt(1, id);
        preparedStatementPessoaJuridica.executeUpdate();

        // exclusão na tabela pai (pessoa)
        preparedStatementPessoa.setInt(1, id);
        preparedStatementPessoa.executeUpdate();

    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

```
package cadastro.util;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

```

```
public class ConectorBD {
```

```

    //configurar conexão
    private final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;";
    private final String USER = "lojaadministrador";
    private final String PASSWORD = "loja12345";

```

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

```

    public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

```

```
    // obter o PreparedStatement
```

```

    public PreparedStatement getPrepared(String sql) throws SQLException {

```



```

        Connection connection = getConnection();
        return connection.prepareStatement(sql);
    }

    // executor de consultas
    public ResultSet getSelect(String sql) throws SQLException {
        PreparedStatement preparedStatement = getPrepared(sql);
        return preparedStatement.executeQuery();
    }

    // metodos para fechar o banco
    public void close(PreparedStatement preparedStatement) {
        if (preparedStatement != null) {
            try {
                preparedStatement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    // metodos para fechar o banco sobrecarregados
    public void close(ResultSet resultSet) {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public void close(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```
package cadastro.util;
```

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

```

```

public class SequenceManager {
    ConectorBD conectorBD = new ConectorBD();

    //obter o próximo valor de uma sequência
    public int getValue(String sequenceName) {

```

```

int nextValue = 0;
String sql = "SELECT NEXT VALUE FOR " + sequenceName;

try (Connection connection = conectorBD.getConnection();
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    ResultSet resultSet = preparedStatement.executeQuery()) {

    if (resultSet.next()) {
        nextValue = resultSet.getInt(1);
    }
} catch (SQLException e) {
    e.printStackTrace();
}

return nextValue;
}
}

```

```

package cadastrobd;

```

```

import cadastro.model.PessoaFisica;
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridica;
import cadastro.model.PessoaJuridicaDAO;

```

```

import java.util.List;
import java.util.Scanner;

```

```

public class SistemaOpcoes {
    private final PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
    private final PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    public void incluirPessoa(Scanner scanner) {
        System.out.print("Incluir Pessoa Física ou Jurídica (f/j)? ");
        String tipo = scanner.nextLine().toLowerCase();

        if (tipo.equals("f")) {
            PessoaFisica pessoaFisica = criarPessoaFisica(scanner);
            pessoaFisicaDAO.incluir(pessoaFisica);
            System.out.println("Pessoa Física incluída com sucesso.");
        } else if (tipo.equals("j")) {
            PessoaJuridica pessoaJuridica = criarPessoaJuridica(scanner);
            pessoaJuridicaDAO.incluir(pessoaJuridica);
            System.out.println("Pessoa Jurídica incluída com sucesso.");
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    public void alterarPessoa(Scanner scanner) {
        System.out.print("Alterar Pessoa Física ou Jurídica (f/j)? ");
        String tipo = scanner.nextLine().toLowerCase();
    }
}

```

```

if (tipo.equals("f")) {
    System.out.print("Digite o ID da Pessoa Física a alterar: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consumir a quebra de linha

    PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
    if (pessoaFisica == null) {
        System.out.println("Pessoa Física não encontrada.");
        return;
    }

    System.out.println("Dados atuais: ");
    pessoaFisica.exibir();

    PessoaFisica novaPessoaFisica = criarPessoaFisica(scanner);
    novaPessoaFisica.setId(id); // manter o mesmo ID
    pessoaFisicaDAO.alterar(novaPessoaFisica);
    System.out.println("Pessoa Física alterada com sucesso.");
} else if (tipo.equals("j")) {
    System.out.print("Digite o ID da Pessoa Jurídica a alterar: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consumir a quebra de linha

    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(id);
    if (pessoaJuridica == null) {
        System.out.println("Pessoa Jurídica não encontrada.");
        return;
    }

    System.out.println("Dados atuais: ");
    pessoaJuridica.exibir();

    PessoaJuridica novaPessoaJuridica = criarPessoaJuridica(scanner);
    novaPessoaJuridica.setId(id); // manter o mesmo ID
    pessoaJuridicaDAO.alterar(novaPessoaJuridica);
    System.out.println("Pessoa Jurídica alterada com sucesso.");
} else {
    System.out.println("Tipo inválido.");
}
}

public void excluirPessoa(Scanner scanner) {
    System.out.print("Excluir Pessoa Física ou Jurídica (f/j)? ");
    String tipo = scanner.nextLine().toLowerCase();

    if (tipo.equals("f")) {
        System.out.print("Digite o ID da Pessoa Física a excluir: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consumir a quebra de linha

        pessoaFisicaDAO.excluir(id);
        System.out.println("Pessoa Física excluída com sucesso.");
    } else if (tipo.equals("j")) {
        System.out.print("Digite o ID da Pessoa Jurídica a excluir: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consumir a quebra de linha
    }
}

```

```

        pessoaJuridicaDAO.excluir(id);
        System.out.println("Pessoa Jurídica excluída com sucesso.");
    } else {
        System.out.println("Tipo inválido.");
    }
}

public void exibirPessoaPorId(Scanner scanner) {
    System.out.print("Exibir Pessoa Física ou Jurídica (f/j)? ");
    String tipo = scanner.nextLine().toLowerCase();

    if (tipo.equals("f")) {
        System.out.print("Digite o ID da Pessoa Física: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consumir a quebra de linha

        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
        if (pessoaFisica == null) {
            System.out.println("Pessoa Física não encontrada.");
        } else {
            pessoaFisica.exibir();
        }
    } else if (tipo.equals("j")) {
        System.out.print("Digite o ID da Pessoa Jurídica: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // consumir a quebra de linha

        PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(id);
        if (pessoaJuridica == null) {
            System.out.println("Pessoa Jurídica não encontrada.");
        } else {
            pessoaJuridica.exibir();
        }
    } else {
        System.out.println("Tipo inválido.");
    }
}

public void exibirTodasPessoas(Scanner scanner) {
    System.out.print("Exibir todas as Pessoas Físicas ou Jurídicas (f/j)? ");
    String tipo = scanner.nextLine().toLowerCase();

    if (tipo.equals("f")) {
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        if (pessoasFisicas.isEmpty()) {
            System.out.println("Nenhuma Pessoa Física encontrada.");
        } else {
            for (PessoaFisica pessoa : pessoasFisicas) {
                pessoa.exibir();
            }
        }
    } else if (tipo.equals("j")) {
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        if (pessoasJuridicas.isEmpty()) {
            System.out.println("Nenhuma Pessoa Jurídica encontrada.");
        } else {
            for (PessoaJuridica pessoa : pessoasJuridicas) {

```

```

        pessoa.exibir();
    }
} else {
    System.out.println("Tipo inválido.");
}
}

private PessoaFisica criarPessoaFisica(Scanner scanner) {
    System.out.println("Digite os dados da Pessoa Física:");
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Logradouro: ");
    String logradouro = scanner.nextLine();
    System.out.print("Cidade: ");
    String cidade = scanner.nextLine();
    System.out.print("Estado: ");
    String estado = scanner.nextLine();
    System.out.print("Telefone: ");
    String telefone = scanner.nextLine();
    System.out.print("Email: ");
    String email = scanner.nextLine();
    System.out.print("CPF: ");
    String cpf = scanner.nextLine();

    return new PessoaFisica(0, nome, logradouro, cidade, estado, telefone, email, cpf);
}

private PessoaJuridica criarPessoaJuridica(Scanner scanner) {
    System.out.println("Digite os dados da Pessoa Jurídica:");
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Logradouro: ");
    String logradouro = scanner.nextLine();
    System.out.print("Cidade: ");
    String cidade = scanner.nextLine();
    System.out.print("Estado: ");
    String estado = scanner.nextLine();
    System.out.print("Telefone: ");
    String telefone = scanner.nextLine();
    System.out.print("Email: ");
    String email = scanner.nextLine();
    System.out.print("CNPJ: ");
    String cnpj = scanner.nextLine();

    return new PessoaJuridica(0, nome, logradouro, cidade, estado, telefone, email, cnpj);
}
}

```

```
package testes;
```

```
import cadastro.model.PessoaFisica;
import cadastro.model.PessoaFisicaDAO;
import cadastro.model.PessoaJuridica;
```

```

import cadastro.model.PessoaJuridicaDAO;

import java.util.List;

public class CadastroBDTeste {
    public static void main(String[] args) {
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
        PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

        // 1° instanciar uma pessoa física e persistir no banco de dados
        PessoaFisica pessoaFisica = new PessoaFisica(0, "João Silva", "Rua 1", "Cidade A", "Estado
X", "123456789", "joao@email.com", "123.456.789-00");
        pessoaFisicaDAO.incluir(pessoaFisica);
        System.out.println("Pessoa Física criada: " + pessoaFisica.getNome());

        // 2° alterar os dados da pessoa física no banco
        pessoaFisica.setNome("João Silva Alterado");
        pessoaFisica.setCidade("Cidade B");
        pessoaFisicaDAO.alterar(pessoaFisica);
        System.out.println("Pessoa Física alterada: " + pessoaFisica.getNome());

        // 3° consultar todas as pessoas físicas do banco de dados e listar
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        System.out.println("\nLista de Pessoas Físicas:");
        for (PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
        }

        // 4° exclusão da pessoa física criada anteriormente
        pessoaFisicaDAO.excluir(pessoaFisica.getId());
        System.out.println("\nPessoa Física excluída: " + pessoaFisica.getNome());

        // 5° instanciar uma pessoa jurídica e persistir no banco de dados
        PessoaJuridica pessoaJuridica = new PessoaJuridica(0, "Empresa XYZ", "Avenida Central",
"Cidade C", "Estado Y", "987654321", "contato@empresa.com", "00.123.456/0001-00");
        pessoaJuridicaDAO.incluir(pessoaJuridica);
        System.out.println("\nPessoa Jurídica criada: " + pessoaJuridica.getNome());

        // 6° alterar os dados da pessoa jurídica no banco
        pessoaJuridica.setNome("Empresa XYZ Alterada");
        pessoaJuridica.setCidade("Cidade D");
        pessoaJuridicaDAO.alterar(pessoaJuridica);
        System.out.println("Pessoa Jurídica alterada: " + pessoaJuridica.getNome());

        // 7° Consultar todas as pessoas jurídicas do banco de dados e listar
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        System.out.println("\nLista de Pessoas Jurídicas:");
        for (PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
        }

        // 8° exclusão da pessoa jurídica criada anteriormente
        pessoaJuridicaDAO.excluir(pessoaJuridica.getId());
        System.out.println("\nPessoa Jurídica excluída: " + pessoaJuridica.getNome());
    }
}

```