Aluno: **Luís Eduardo Wataro Nagata**

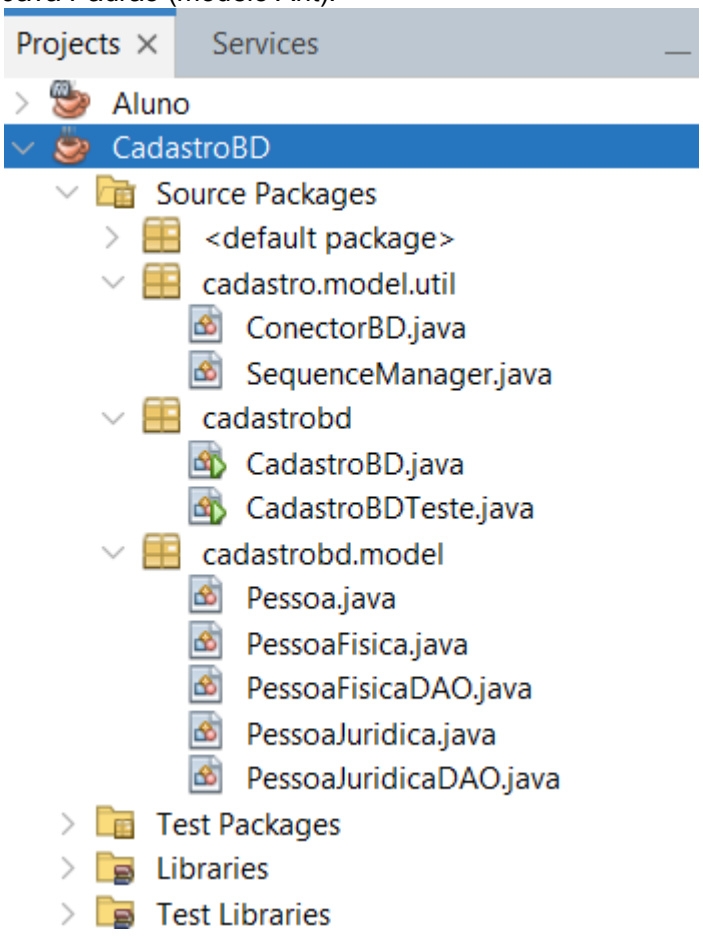
Objetivos da Prática:

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

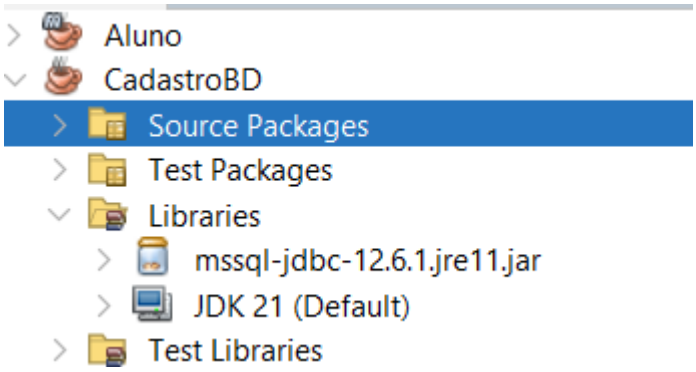
1º Procedimento | Mapeamento Objeto-Relacional e DAO

1- Criar o projeto e configurar as bibliotecas necessárias:

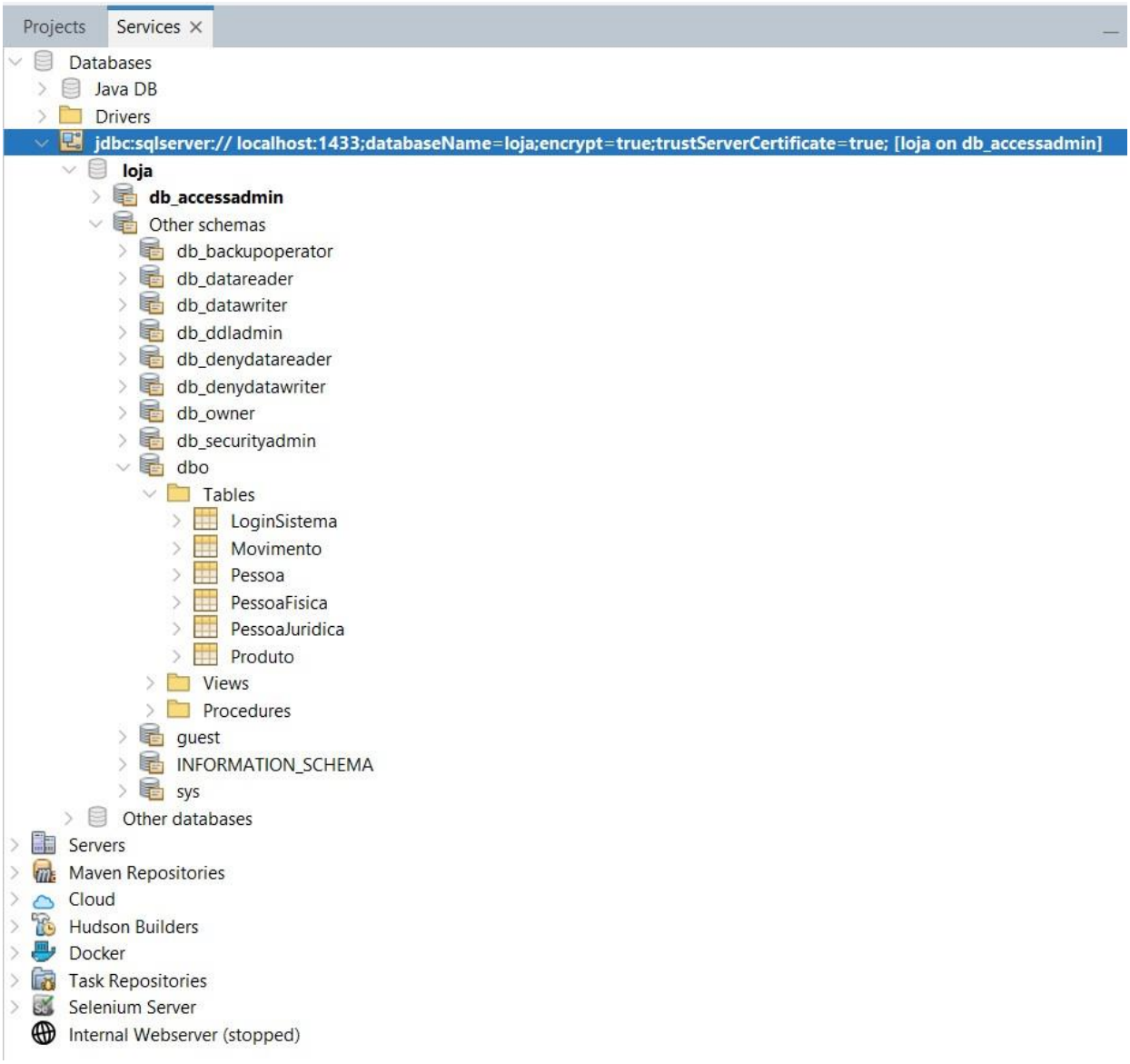
Criar um projeto no NetBeans, utilizando o nome CadastroBD, do tipo Aplicativo Java Padrão (modelo Ant).



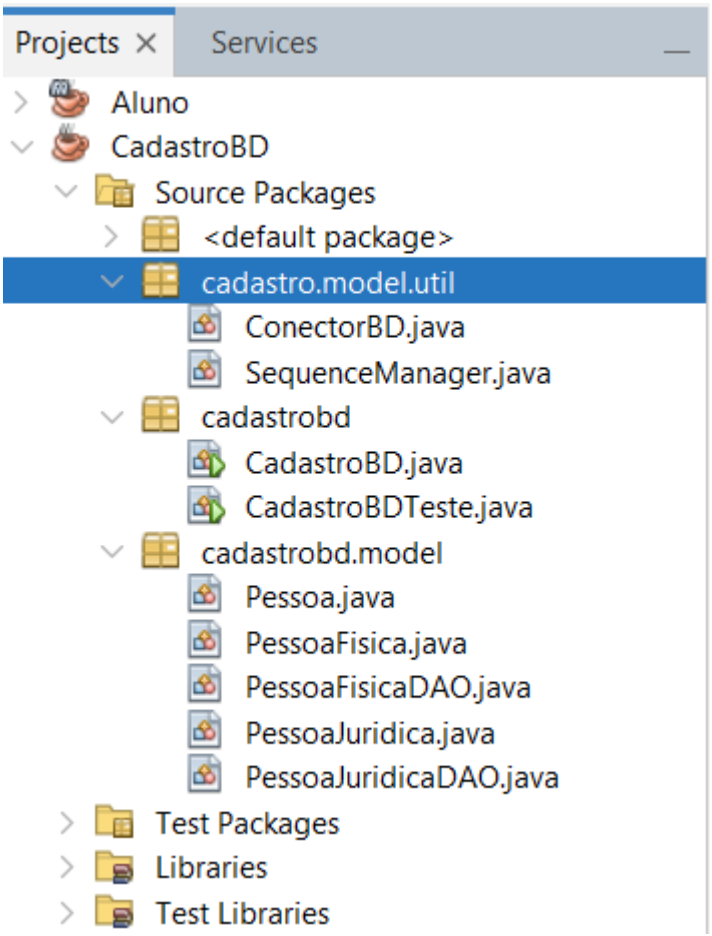
Adicionar o driver JDBC para SQL Server ao projeto, com o clique do botão direito sobre bibliotecas (libraries) e escolha da opção jar.



2- Configurar o acesso ao banco pela aba de serviços do NetBeans.



3- Voltando ao projeto, criar o pacote cadastrobd.model, e nele criar as classes apresentadas a seguir:



Classe Pessoa:

```
package cadastrobd.model;

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

    // Construtor padrão
    public Pessoa() {
    }

    // Construtor completo
    public Pessoa(int id, String nome, String logradouro, String cidade, String estado, int telefone,
String email) {
```

```
    this.id = id;
    this.nome = nome;
    this.logradouro = logradouro;
    this.cidade = cidade;
    this.estado = estado;
    this.telefone = telefone;
    this.email = email;
}

// Método para exibir os dados no console
public void exibir() {
    System.out.println("ID: " + id);
    System.out.println("Nome: " + nome);
    System.out.println("Logradouro: " + logradouro);
    System.out.println("Cidade: " + cidade);
    System.out.println("Estado: " + estado);
    System.out.println("Telefone: " + telefone);
    System.out.println("Email: " + email);
}

// Getters
public int getId() {
    return id;
}

public String getNome() {
    return nome;
}

public String getLogradouro() {
    return logradouro;
}

public String getCidade() {
    return cidade;
}

public String getEstado() {
    return estado;
}

public int getTelefone() {
    return telefone;
}
```

```
    public String getEmail() {
        return email;
    }
}
```

Classe PessoaFisica:

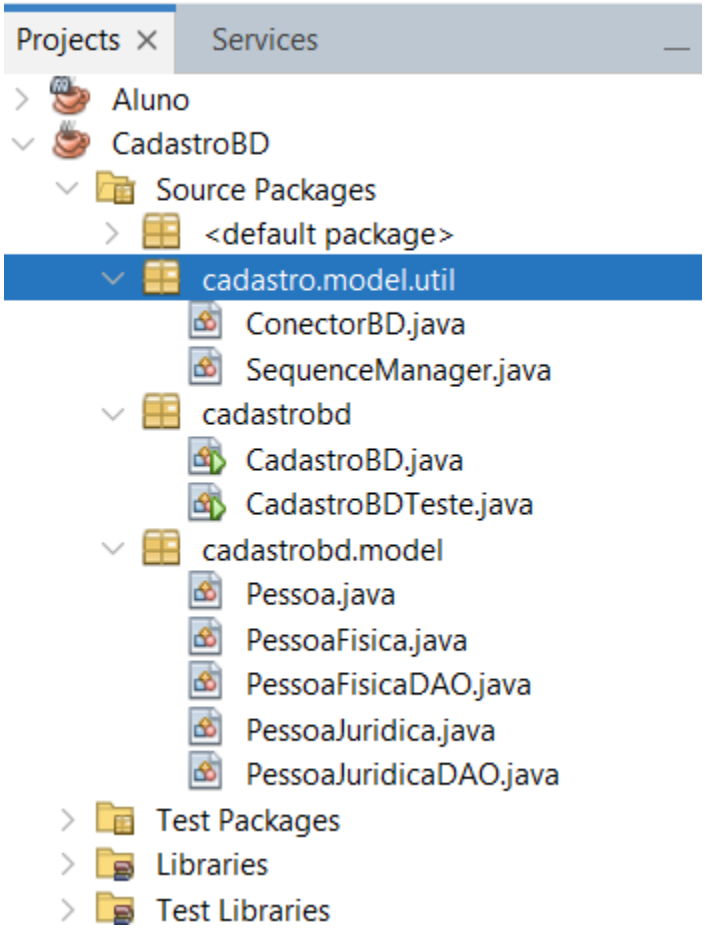
```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    // Construtor padrão
    public PessoaFisica(String maria, String rua_A, String cidade_A, String estado_A, int par,
String mariaexamplecom, String string) {
        super();
    }

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

    // Polimorfismo no método exibir
    @Override
    public void exibir() {
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
        System.out.println("Logradouro: " + getLogradouro());
        System.out.println("Cidade: " + getCidade());
        System.out.println("Estado: " + getEstado());
        System.out.println("Telefone: " + getTelefone());
        System.out.println("Email: " + getEmail());
        System.out.println("CPF: " + cpf);
    }

    // Getters e setters
    public String getCpf() {
        return cpf;
    }
```

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

    // Métodos restantes
    @Override
    public int getId() {
        return super.getId();
    }

    @Override
    public String getNome() {
        return super.getNome();
    }

    @Override
    public String getLogradouro() {
        return super.getLogradouro();
    }

    @Override
    public String getCidade() {
        return super.getCidade();
    }

    @Override
    public String getEstado() {
        return super.getEstado();
    }

    @Override
    public int getTelefone() {
        return super.getTelefone();
    }

    @Override
    public String getEmail() {
        return super.getEmail();
    }
}
```

Classe PessoaJuridica:

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
        super(); // Chama o construtor padrão da classe Pessoa
    }

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

    // Polimorfismo no método exibir
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da classe Pessoa
        System.out.println("CNPJ: " + cnpj);
    }

    // Getters e setters
    public String getCnpj() {
        return cnpj;
    }

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

4- Criar o pacotes cadastro.model.util, para inclusão das classes utilitárias que são apresentadas a seguir:



Classe ConectorBD:

```
package cadastro.model.util;

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

public class ConectorBD {
    // Configurações de acesso ao banco de dados SQL Server
    private static final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=cadastrobd;encrypt=true;trustServerCertificate=tr
    ue;"; private static final String USUARIO = "root";
    private static final String SENHA = "0044";

    // Método para obter uma conexão com o banco de dados
    public static Connection getConnection() throws SQLException {
```

```
    return DriverManager.getConnection(URL, USUARIO, SENHA);
}

// Método para criar um objeto PreparedStatement a partir de uma consulta SQL
public static PreparedStatement getPrepared(String sql) throws SQLException {
    return getConnection().prepareStatement(sql);
}

// Método para executar uma consulta SQL e retornar o ResultSet resultante
public static ResultSet getSelect(String sql) throws SQLException {
    Statement statement = getConnection().createStatement();
    return statement.executeQuery(sql);
}

// Métodos para fechar recursos relacionados ao banco de dados
public static void close(ResultSet resultSet) {
    if (resultSet != null) {
        try {
            resultSet.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

public static void close(Statement statement) {
    if (statement != null) {
        try {
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

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

```
}
```

Classe SequenceManager:

```
package cadastro.model.util;

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

public class SequenceManager {
   public static int getValue(String sequenceName) {
      int nextValue = 0;
      String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS nextval";
      try (Connection connection = ConectorBD.getConnection();
           PreparedStatement statement = connection.prepareStatement(sql);
           ResultSet resultSet = statement.executeQuery()) {
         if (resultSet.next()) {
            nextValue = resultSet.getInt("nextval");
         }
      } catch (SQLException e) {
         e.printStackTrace();
      }
      return nextValue;
   }
}
```

5- Codificar as classes no padrão DAO, no pacote cadastro.model:

Classe PessoaFisicaDAO:

```
package cadastrobd.model;

import cadastro.model.util.ConectorBD;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
   public PessoaFisica getPessoa(int id) {
      PessoaFisica pessoa = null;
```

```
        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.id =
PessoaFisica.id WHERE Pessoa.id = ?";
        try (Connection connection = ConectorBD.getConnection();
            PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, id);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                pessoa = new PessoaFisica(
                        resultSet.getInt("id"),
                        resultSet.getString("nome"),
                        resultSet.getString("logradouro"),
                        resultSet.getString("cidade"),
                        resultSet.getString("estado"),
                        resultSet.getInt("telefone"),
                        resultSet.getString("email"),
                        resultSet.getString("cpf")
                );
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return pessoa;
    }

    public List<PessoaFisica> getPessoas() {
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.id =
PessoaFisica.id";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                PessoaFisica pessoa = new PessoaFisica(
                        resultSet.getInt("id"),
                        resultSet.getString("nome"),
                        resultSet.getString("logradouro"),
                        resultSet.getString("cidade"),
                        resultSet.getString("estado"),
                        resultSet.getInt("telefone"),
                        resultSet.getString("email"),
                        resultSet.getString("cpf")
                );
                pessoas.add(pessoa);
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return pessoas;
    }

    public void incluir(PessoaFisica pessoa) {
        String sqlPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone,
email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaFisica = "INSERT INTO PessoaFisica (id, cpf) VALUES (?, ?)";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statementPessoa = connection.prepareStatement(sqlPessoa,
PreparedStatement.RETURN_GENERATED_KEYS);
             PreparedStatement statementPessoaFisica =
connection.prepareStatement(sqlPessoaFisica)) {
            statementPessoa.setString(1, pessoa.getNome());
            statementPessoa.setString(2, pessoa.getLogradouro());
            statementPessoa.setString(3, pessoa.getCidade());
            statementPessoa.setString(4, pessoa.getEstado());
            statementPessoa.setInt(5, pessoa.getTelefone());
            statementPessoa.setString(6, pessoa.getEmail());

            statementPessoa.executeUpdate();

            ResultSet rs = statementPessoa.getGeneratedKeys();
            int id = 0;
            if (rs.next()) {
                id = rs.getInt(1);
            }

            statementPessoaFisica.setInt(1, id);
            statementPessoaFisica.setString(2, pessoa.getCpf());

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

    // Método para exibir os dados da tabela Pessoa
    public void exibirDadosTabelaPessoa() {
        String sql = "SELECT * FROM Pessoa";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
```

```
            ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id"));
                System.out.println("Nome: " + resultSet.getString("nome"));
                System.out.println("Logradouro: " + resultSet.getString("logradouro"));
                System.out.println("Cidade: " + resultSet.getString("cidade"));
                System.out.println("Estado: " + resultSet.getString("estado"));
                System.out.println("Telefone: " + resultSet.getInt("telefone"));
                System.out.println("Email: " + resultSet.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void excluir(int id) {
        String sqlPessoa = "DELETE FROM Pessoa WHERE id = ?";
    String sqlPessoaFisica = "DELETE FROM PessoaFisica WHERE id = ?";
    try (Connection connection = ConectorBD.getConnection();
         PreparedStatement statementPessoa = connection.prepareStatement(sqlPessoa);
         PreparedStatement statementPessoaFisica =
connection.prepareStatement(sqlPessoaFisica)) {
        statementPessoa.setInt(1, id);
        statementPessoa.executeUpdate();
        statementPessoaFisica.setInt(1, id);
        statementPessoaFisica.executeUpdate();
        System.out.println("Pessoa física com o ID " + id + " excluída com sucesso.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
    }
}
```

Classe PessoaJuridicaDAO:

```
package cadastrobd.model;

import cadastro.model.util.ConectorBD;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
```

```
import java.util.List;

public class PessoaJuridicaDAO {

    // Método para obter uma pessoa jurídica pelo ID
    public PessoaJuridica getPessoa(int id) {
        PessoaJuridica pessoa = null;
        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaJuridica ON Pessoa.id =
PessoaJuridica.id WHERE Pessoa.id = ?";
        try (Connection connection = ConectorBD.getConnection();
            PreparedStatement statement = connection.prepareStatement(sql)) {
            statement.setInt(1, id);
            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                pessoa = new PessoaJuridica(
                        resultSet.getInt("id"),
                        resultSet.getString("nome"),
                        resultSet.getString("logradouro"),
                        resultSet.getString("cidade"),
                        resultSet.getString("estado"),
                        resultSet.getInt("telefone"),
                        resultSet.getString("email"),
                        resultSet.getString("cnpj")
                );
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return pessoa;
    }

    // Método para obter todas as pessoas jurídicas
    public List<PessoaJuridica> getPessoas() {
        List<PessoaJuridica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaJuridica ON Pessoa.id =
PessoaJuridica.id";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                PessoaJuridica pessoa = new PessoaJuridica(
                        resultSet.getInt("id"),
                        resultSet.getString("nome"),
                        resultSet.getString("logradouro"),
```

```
                    resultSet.getString("cidade"),
                    resultSet.getString("estado"),
                    resultSet.getInt("telefone"),
                    resultSet.getString("email"),
                    resultSet.getString("cnpj")
                );
                pessoas.add(pessoa);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return pessoas;
    }

    // Método para incluir uma nova pessoa jurídica
    public void incluir(PessoaJuridica pessoa) {
        String sqlPessoa = "INSERT INTO Pessoa (nome, logradouro, cidade, estado, telefone,
email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (id, cnpj) VALUES (?, ?)";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statementPessoa = connection.prepareStatement(sqlPessoa,
PreparedStatement.RETURN_GENERATED_KEYS);
             PreparedStatement statementPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {
            statementPessoa.setString(1, pessoa.getNome());
            statementPessoa.setString(2, pessoa.getLogradouro());
            statementPessoa.setString(3, pessoa.getCidade());
            statementPessoa.setString(4, pessoa.getEstado());
            statementPessoa.setInt(5, pessoa.getTelefone());
            statementPessoa.setString(6, pessoa.getEmail());

            statementPessoa.executeUpdate();

            ResultSet rs = statementPessoa.getGeneratedKeys();
            int id = 0;
            if (rs.next()) {
                id = rs.getInt(1);
            }

            statementPessoaJuridica.setInt(1, id);
            statementPessoaJuridica.setString(2, pessoa.getCnpj());

            statementPessoaJuridica.executeUpdate();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }

    // Método para exibir os dados da tabela Pessoa (para teste)
    public void exibirDadosTabelaPessoa() {
        String sql = "SELECT * FROM Pessoa";
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement statement = connection.prepareStatement(sql);
             ResultSet resultSet = statement.executeQuery()) {
            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id") +
                        ", Nome: " + resultSet.getString("nome") +
                        ", Logradouro: " + resultSet.getString("logradouro") +
                        ", Cidade: " + resultSet.getString("cidade") +
                        ", Estado: " + resultSet.getString("estado") +
                        ", Telefone: " + resultSet.getInt("telefone") +
                        ", Email: " + resultSet.getString("email"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void excluir(int id) {
       String sqlPessoa = "DELETE FROM Pessoa WHERE id = ?";
    String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE id = ?";
    try (Connection connection = ConectorBD.getConnection();
         PreparedStatement statementPessoa = connection.prepareStatement(sqlPessoa);
         PreparedStatement statementPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {
        statementPessoa.setInt(1, id);
        statementPessoa.executeUpdate();
        statementPessoaJuridica.setInt(1, id);
        statementPessoaJuridica.executeUpdate();
        System.out.println("Pessoa jurídica com o ID " + id + " excluída com sucesso.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
    }
}
```

6- Criar uma classe principal de testes com o nome CadastroBDTeste, efetuando as operações seguintes no método main:

Classe CadastroBDTeste:

```
package cadastrobd;

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

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

        System.out.println("Pessoas Físicas:");
        for (PessoaFisica pessoaFisica : pessoaFisicaDAO.getPessoas()) {
            System.out.println("ID: " + pessoaFisica.getId());
            System.out.println("Nome: " + pessoaFisica.getNome());
            System.out.println("Logradouro: " + pessoaFisica.getLogradouro());
            System.out.println("Cidade: " + pessoaFisica.getCidade());
            System.out.println("Estado: " + pessoaFisica.getEstado());
            System.out.println("Telefone: " + pessoaFisica.getTelefone());
            System.out.println("Email: " + pessoaFisica.getEmail());
            System.out.println("CPF: " + pessoaFisica.getCpf());
        }

        System.out.println("\nPessoas Jurídicas:");
        for (PessoaJuridica pessoaJuridica : pessoaJuridicaDAO.getPessoas()) {
            System.out.println("ID: " + pessoaJuridica.getId());
            System.out.println("Nome: " + pessoaJuridica.getNome());
            System.out.println("Logradouro: " + pessoaJuridica.getLogradouro());
            System.out.println("Cidade: " + pessoaJuridica.getCidade());
            System.out.println("Estado: " + pessoaJuridica.getEstado());
            System.out.println("Telefone: " + pessoaJuridica.getTelefone());
            System.out.println("Email: " + pessoaJuridica.getEmail());
            System.out.println("CNPJ: " + pessoaJuridica.getCnpj());
        }
    }
}
```

7- A saída do sistema deverá ser semelhante à que é apresentada a seguir:

8- Verificar os resultados obtidos através do console de saída do NetBeans:

run:
Pessoas Físicas:
ID: 7
Nome: Joao
Logradouro: Rua 12, casa 3, Quitanda
Cidade: Riacho do Sul
Estado: PA
Telefone: 0
Email: joao@riacho.com
CPF: 111111111111
ID: 19
Nome: João
Logradouro: Rua A
Cidade: Cidade A
Estado: Estado A
Telefone: 123456789
Email: joao@example.com
CPF: 123.456.789-11

Pessoas Jurídicas:
ID: 15
Nome: JJC
Logradouro: Rua 11, Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 100
Email: jjc@riacho.com
CNPJ: 2222222222

ID: 20
Nome: Empresa X
Logradouro: Rua B
Cidade: Cidade B
Estado: Estado B
Telefone: 987654321
Email: empresa@example.com
CNPJ: 12.345.678/0001-00
BUILD SUCCESSFUL (total time: 0 seconds)

Análise e Conclusão:

A. **Qual a importância dos componentes de middleware, como o JDBC?**

R- Os componentes de middleware, como o JDBC, são importantes porque simplificam o acesso a sistemas distribuídos, promovem a interoperabilidade entre diferentes tecnologias, facilitam a reutilização de código, gerenciam transações e melhoram a escalabilidade e o desempenho dos sistemas distribuídos.

B. **Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**

R- A diferença principal entre Statement e PreparedStatement está na forma como eles lidam com consultas SQL no Java:

Statement: Utilizado para executar consultas estáticas e simples.
As consultas são diretamente concatenadas aos comandos SQL, o que pode tornar o código vulnerável a injeção de SQL se não for tratado adequadamente.
Menos eficiente em termos de desempenho quando consultas são executadas repetidamente, pois o banco de dados precisa analisar e compilar a consulta a cada execução.

PreparedStatement: Utilizado para executar consultas parametrizadas.
As consultas são pré-compiladas pelo banco de dados, o que resulta em melhor desempenho quando a mesma consulta é executada várias vezes com diferentes parâmetros.
Ajuda a prevenir ataques de injeção de SQL, pois os parâmetros são tratados separadamente da consulta SQL principal.

C. **Como o padrão DAO melhora a manutenibilidade do software?**

R- O padrão DAO (Data Access Object) melhora a manutenibilidade do software separando a lógica de acesso a dados do restante da aplicação. Isso significa que as operações de acesso a dados, como salvar, recuperar, atualizar e excluir, são encapsuladas em classes dedicadas, chamadas de DAOs. Como resultado:

1. Separação de Responsabilidades: O DAO isola o código de acesso aos dados, permitindo que as outras partes da aplicação se concentrem em suas próprias funcionalidades sem se preocupar com os detalhes de como os dados são acessados e manipulados.

2. Facilidade de Manutenção: Se houver alterações na forma como os dados são armazenados ou acessados, as alterações precisarão ser feitas apenas no DAO correspondente, em vez de em vários locais em toda a aplicação. Isso torna as atualizações e manutenções mais simples e menos propensas a erros.

3. Reutilização de Código: Como as operações de acesso a dados estão encapsuladas em classes DAO, elas podem ser facilmente reutilizadas em diferentes partes da aplicação, promovendo uma melhor modularidade e reduzindo a duplicação de código.

4. Testabilidade: O uso de DAOs facilita a criação de testes unitários, pois as operações de acesso a dados podem ser isoladas e testadas independentemente do restante da aplicação.

Em resumo, o padrão DAO ajuda a promover um design mais modular e organizado, o que resulta em um software mais fácil de manter, entender e testar.

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

R- Quando lidamos com um modelo estritamente relacional em um banco de dados, a herança pode ser refletida de diferentes maneiras, dependendo da abordagem adotada. Uma abordagem comum é a **herança por tabelas concretas**, onde cada classe concreta no modelo de objetos é mapeada para uma tabela separada no banco de dados. Isso resulta em uma tabela para cada subclasse, contendo não apenas os atributos exclusivos da subclasse, mas também os atributos herdados da superclasse.

Outra abordagem é a **herança por tabela única**, onde todos os atributos de todas as subclasses são agrupados em uma única tabela no banco de dados, juntamente com um indicador de tipo que especifica a classe concreta de cada registro. Isso pode resultar em muitos campos nulos para registros que não possuem todos os atributos das subclasses.

Uma terceira abordagem é a **herança por tabelas por classe**, onde cada classe no modelo de objetos é mapeada para uma tabela separada no banco de dados, contendo apenas seus próprios atributos. Não há uma tabela para a superclasse, e as subclasses não compartilham tabelas.

A escolha entre essas abordagens depende das necessidades específicas do aplicativo, do desempenho, da facilidade de consulta e da complexidade do modelo de objetos. Cada abordagem tem suas vantagens e desvantagens em termos de desempenho, facilidade de consulta e normalização do banco de dados.