Criação de aplicativo Java, com acesso ao banco de dados SQL Server através do

middleware JDBC.

Eros Santos de Vasconcelos/ 202307120545

Polo Iputinga
Missão Prática Back-end Sem Banco Não Tem – 9001 – 3° semestre

Objetivo da Prática

O objetivo desta prática é desenvolver um sistema de persistência de dados utilizando o middleware JDBC e o padrão DAO (Data Access Object) em um aplicativo cadastral para gerenciamento de informações de pessoas físicas e jurídicas. A prática envolve a modelagem objeto-relacional, permitindo a interação entre a aplicação Java e um banco de dados relacional. Além disso, o projeto será versionado no GIT, e todos os códigos e a documentação estarão organizados no repositório.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

Códigos do 1º procedimento:

1. **CadastroBDTeste:**

```
package cadastrobd;


import cadastrobd.model.PessoaFisica;

import cadastrobd.model.PessoaJuridica;

import cadastrobd.model.dao.PessoaFisicaDAO;

import cadastrobd.model.dao.PessoaJuridicaDAO;

import java.sql.SQLException;


public class CadastroBDTeste {

    public static void main(String[] args) {

        try {
```

```
            PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
            PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();

            PessoaFisica pf = new PessoaFisica(0, "João Silva", "Rua A",
"Recife", "PE", "81987654321", "joao@example.com", "423.456.789-09");
            pfDAO.incluir(pf);
            System.out.println("Pessoa Física incluída:");
            pf.exibir();

            pf.setNome("João Pedro Silva");
            pfDAO.alterar(pf);
            System.out.println("Pessoa Física alterada:");
            pf.exibir();

            System.out.println("Todas as Pessoas Físicas:");
            for (PessoaFisica pessoa : pfDAO.getPessoas()) {
                pessoa.exibir();
            }

            pfDAO.excluir(pf.getId());
            System.out.println("Pessoa Física excluída.");

            PessoaJuridica pj = new PessoaJuridica(0, "Empresa X", "Rua B",
"Recife", "PE", "8130123456", "contato@empresa.com", "42.345.678/0001-99");
            pjDAO.incluir(pj);
            System.out.println("Pessoa Jurídica incluída:");
            pj.exibir();

            pj.setNome("Empresa Y");
            pjDAO.alterar(pj);
            System.out.println("Pessoa Jurídica alterada:");
            pj.exibir();
```

```
            System.out.println("Todas as Pessoas Jurídicas:");
            for (PessoaJuridica pessoa : pjDAO.getPessoas()) {
                pessoa.exibir();
            }

            pjDAO.excluir(pj.getId());
            System.out.println("Pessoa Jurídica excluída.");

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

2. Criação das Tabelas:

- **Class Pessoa:**

```
package cadastrobd.model;

public class Pessoa {
    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(String nome, String logradouro, String cidade, String estado,
String telefone, String email) {
        this.nome = nome;
        this.logradouro = logradouro;
```

```
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }

    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 String getNome() {
        return nome;
    }

    public String getLogradouro() {
        return logradouro;
    }

    public String getCidade() {
        return cidade;
    }
```

```
public String getEstado() {
    return estado;
}

public String getTelefone() {
    return telefone;
}

public String getEmail() {
    return email;
}

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

public void setNome(String nome) {
    if (nome == null || nome.trim().isEmpty()) {
        throw new IllegalArgumentException("O nome não pode ser vazio.");
    }
    this.nome = nome;
}

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

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

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

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

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

    public void exibir() {
        StringBuilder sb = new StringBuilder();
        sb.append("ID: ").append(id)
          .append(", Nome: ").append(nome)
          .append(", Logradouro: ").append(logradouro)
          .append(", Cidade: ").append(cidade)
          .append(", Estado: ").append(estado)
          .append(", Telefone: ").append(telefone)
          .append(", Email: ").append(email);

        System.out.println(sb.toString());
    }
```

- **Class PessoaFisica:**

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    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); // Chama o
construtor da classe pai
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }

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

    public void exibir() {
        System.out.println("Pessoa Física: " + getNome() + ", CPF: " + cpf);
    }
```

- **Class PessoaJuridica:**

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    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); //
Chama o construtor da classe pai
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }
```

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

        public void exibir() {
            System.out.println("Pessoa Jurídica: " + getNome() + ", CNPJ: " +
    cnpj);
        }
    }
```

- **Class PessoaFisicaDAO:**

```
package cadastrobd.model.dao;

import cadastrobd.model.PessoaFisica;
import cadastrobd.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 {
    private Connection conn;

    public PessoaFisicaDAO() {
        try {
            this.conn = ConectorBD.getConnection(); // Obtendo a conexão
        } catch (SQLException e) {
```

```
            System.err.println("Erro ao conectar ao banco de dados: " +
e.getMessage());
            throw new RuntimeException(e); // Propagar exceção
        }
    }

    public void incluir(PessoaFisica pf) throws SQLException {
        String sql = "INSERT INTO Pessoa (Nome, Logradouro, Cidade, Estado,
Telefone, Email) VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sql,
PreparedStatement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, pf.getNome());
            stmt.setString(2, pf.getLogradouro());
            stmt.setString(3, pf.getCidade());
            stmt.setString(4, pf.getEstado());
            stmt.setString(5, pf.getTelefone());
            stmt.setString(6, pf.getEmail());
            stmt.executeUpdate();

            ResultSet rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                int idPessoa = rs.getInt(1);
                pf.setId(idPessoa); // Atualiza o ID da PessoaFisica
            }
        }

        String sqlPf = "INSERT INTO PessoaFisica (ID_Pessoa, CPF) VALUES (?,
?)";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPf)) {
            stmt.setInt(1, pf.getId());
            stmt.setString(2, pf.getCpf());
            stmt.executeUpdate();
        }
```

```
    }

    public void alterar(PessoaFisica pf) throws SQLException {
        String sql = "UPDATE Pessoa SET Nome=?, Logradouro=?, Cidade=?,
Estado=?, Telefone=?, Email=? WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, pf.getNome());
            stmt.setString(2, pf.getLogradouro());
            stmt.setString(3, pf.getCidade());
            stmt.setString(4, pf.getEstado());
            stmt.setString(5, pf.getTelefone());
            stmt.setString(6, pf.getEmail());
            stmt.setInt(7, pf.getId());
            stmt.executeUpdate();
        }

        String sqlPf = "UPDATE PessoaFisica SET CPF=? WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPf)) {
            stmt.setString(1, pf.getCpf());
            stmt.setInt(2, pf.getId());
            stmt.executeUpdate();
        }
    }

    public void excluir(int id) throws SQLException {
        String sqlPf = "DELETE FROM PessoaFisica WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPf)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }

        String sql = "DELETE FROM Pessoa WHERE ID_Pessoa=?";
```

```
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setInt(1, id);
                stmt.executeUpdate();
            }
        }

        public List<PessoaFisica> getPessoas() throws SQLException {
            List<PessoaFisica> pessoas = new ArrayList<>();
            String sql = "SELECT * FROM PessoaFisica pf JOIN Pessoa p ON
    pf.ID_Pessoa = p.ID_Pessoa";
            try (PreparedStatement stmt = conn.prepareStatement(sql);
                 ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    PessoaFisica pf = new PessoaFisica(
                            rs.getInt("ID_PessoaFisica"),
                            rs.getString("Nome"),
                            rs.getString("Logradouro"),
                            rs.getString("Cidade"),
                            rs.getString("Estado"),
                            rs.getString("Telefone"),
                            rs.getString("Email"),
                            rs.getString("CPF")
                    );
                    pessoas.add(pf);
                }
            }
            return pessoas;
        }
```

- **Class PessoaJuridicaDAO:**

```
package cadastrobd.model.dao;

import cadastrobd.model.PessoaJuridica;
```

```
import cadastrobd.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 {
    private Connection conn;

    public PessoaJuridicaDAO() {
        try {
            this.conn = ConectorBD.getConnection(); // Obtendo a conexão
        } catch (SQLException e) {
            System.err.println("Erro ao conectar ao banco de dados: " +
e.getMessage());
            throw new RuntimeException(e); // Propagar exceção
        }
    }

    public void incluir(PessoaJuridica pj) throws SQLException {
        String sql = "INSERT INTO Pessoa (Nome, Logradouro, Cidade, Estado,
Telefone, Email) VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sql,
PreparedStatement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, pj.getNome());
            stmt.setString(2, pj.getLogradouro());
            stmt.setString(3, pj.getCidade());
            stmt.setString(4, pj.getEstado());
            stmt.setString(5, pj.getTelefone());
            stmt.setString(6, pj.getEmail());
```

```
            stmt.executeUpdate();

            ResultSet rs = stmt.getGeneratedKeys();
            if (rs.next()) {
                int idPessoa = rs.getInt(1);
                pj.setId(idPessoa); // Atualiza o ID da PessoaJuridica
            }
        }

        String sqlPj = "INSERT INTO PessoaJuridica (ID_Pessoa, CNPJ) VALUES (?,
?)";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPj)) {
            stmt.setInt(1, pj.getId());
            stmt.setString(2, pj.getCnpj());
            stmt.executeUpdate();
        }
    }

    public void alterar(PessoaJuridica pj) throws SQLException {
        String sql = "UPDATE Pessoa SET Nome=?, Logradouro=?, Cidade=?, Estado=?,
Telefone=?, Email=? WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, pj.getNome());
            stmt.setString(2, pj.getLogradouro());
            stmt.setString(3, pj.getCidade());
            stmt.setString(4, pj.getEstado());
            stmt.setString(5, pj.getTelefone());
            stmt.setString(6, pj.getEmail());
            stmt.setInt(7, pj.getId());
            stmt.executeUpdate();
        }
```

```
        String sqlPj = "UPDATE PessoaJuridica SET CNPJ=? WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPj)) {
            stmt.setString(1, pj.getCnpj());
            stmt.setInt(2, pj.getId());
            stmt.executeUpdate();
        }
    }

    public void excluir(int id) throws SQLException {
        String sqlPj = "DELETE FROM PessoaJuridica WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlPj)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }

        String sql = "DELETE FROM Pessoa WHERE ID_Pessoa=?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }

    public List<PessoaJuridica> getPessoas() throws SQLException {
        List<PessoaJuridica> pessoas = new ArrayList<>();
        String sql = "SELECT * FROM PessoaJuridica pj JOIN Pessoa p ON
pj.ID_Pessoa = p.ID_Pessoa";
        try (PreparedStatement stmt = conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                PessoaJuridica pj = new PessoaJuridica(
                        rs.getInt("ID_PessoaJuridica"),
                        rs.getString("Nome"),
```

```
                    rs.getString("Logradouro"),
                    rs.getString("Cidade"),
                    rs.getString("Estado"),
                    rs.getString("Telefone"),
                    rs.getString("Email"),
                    rs.getString("CNPJ")
            );
            pessoas.add(pj);
        }
    }
    return pessoas;
}
```

3. ConectorBD:

```
package cadastrobd.util;

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

public class ConectorBD {

    private static final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServe
rCertificate=true";
    private static final String USER = "loja";
    private static final String PASSWORD = "loja";

    public static Connection getConnection() throws SQLException {
        try {
```

```
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (SQLException e) {
            System.err.println("Erro ao tentar conectar ao banco de dados: " + e.getMessage());
            throw e;  // Repassa a exceção para tratamento externo
        }
    }

    public static PreparedStatement getPrepared(Connection conn, String sql) throws SQLException {
        return conn.prepareStatement(sql);
    }

    public static PreparedStatement getPrepared(Connection conn, String sql, int autoGeneratedKeys) throws SQLException {
        return conn.prepareStatement(sql, autoGeneratedKeys);
    }

    public static ResultSet getSelect(PreparedStatement stmt) throws SQLException {
        return stmt.executeQuery();
    }

    public static void close(Connection conn) {
        closeResource(conn, "conexão");
    }

    public static void close(PreparedStatement stmt) {
        closeResource(stmt, "PreparedStatement");
    }

    public static void close(ResultSet rs) {
        closeResource(rs, "ResultSet");
```

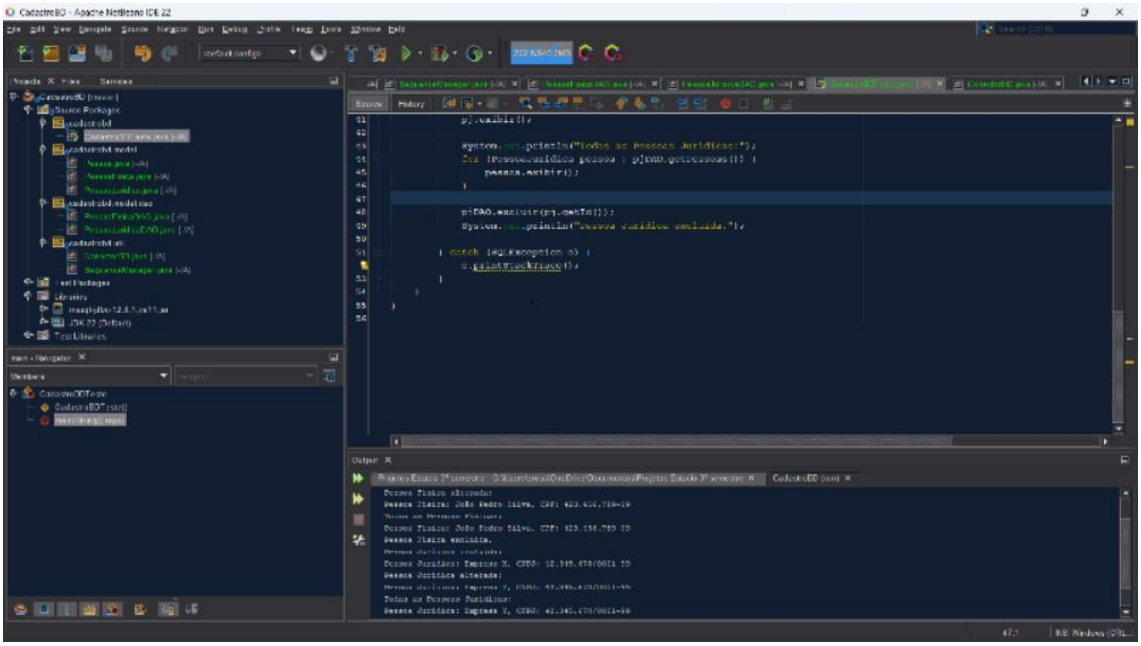
```
    }

    private static void closeResource(AutoCloseable resource, String
resourceName) {
        if (resource != null) {
            try {
                resource.close();
            } catch (Exception e) {
                System.err.println("Erro ao fechar " + resourceName + ": "
+ e.getMessage());
            }
        }
    }
}
```

SequenceManager:

```
package cadastrobd.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) throws SQLException {
        Connection conn = null;
```

```
        PreparedStatement stmt = null;
        ResultSet rs = null;
        int value = 0;

        if (sequenceName == null || sequenceName.trim().isEmpty()) {
            throw new IllegalArgumentException("Nome da sequência não pode
ser nulo ou vazio.");
        }

        try {
            conn = ConectorBD.getConnection();
            String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS
value";
            stmt = conn.prepareStatement(sql);
            rs = stmt.executeQuery();

            if (rs.next()) {
                value = rs.getInt("value");
            }
        } finally {
            ConectorBD.close(rs);
            ConectorBD.close(stmt);
            ConectorBD.close(conn);
        }

        return value;
    }
}
```

Resultado da execução:



a) **Qual a importância dos componentes de middleware, como o JDBC?** O middleware, como o JDBC, é crucial para a comunicação entre a aplicação Java e o banco de dados. Ele fornece uma interface padronizada para executar comandos SQL, manipular dados e gerenciar conexões, abstraindo detalhes complexos e permitindo que os desenvolvedores se concentrem na lógica de negócio.

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

O PreparedStatement é preferível ao Statement porque permite a reutilização de consultas com parâmetros, reduzindo a vulnerabilidade a injeções SQL e melhorando a performance ao compilar a consulta apenas uma vez. Além disso, o PreparedStatement oferece uma forma mais segura e conveniente de definir parâmetros.

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

O padrão DAO promove a separação de responsabilidades, isolando a lógica de acesso a dados da lógica de negócio. Isso torna o código mais modular e fácil de entender, permitindo que alterações na estrutura do banco de dados sejam feitas sem impactar a lógica da aplicação, resultando em um software mais fácil de manter e escalar.

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

Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional? Em um modelo relacional, a herança é geralmente representada através de tabelas separadas para cada classe, onde uma tabela principal contém os atributos comuns, e tabelas filhas armazenam os atributos específicos. Essa abordagem mantém a integridade referencial e permite que as relações sejam estabelecidas entre as entidades.

2º Procedimento | Alimentando a Base

Códigos do 2º procedimento:

1. **CadastroBDTeste:**

```
package cadastrobd;


import java.util.Scanner;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.dao.PessoaFisicaDAO;
import cadastrobd.model.dao.PessoaJuridicaDAO;


public class CadastroBDTeste {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
        PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();
        int opcao = -1;


        while (opcao != 0) {
            System.out.println("Selecione uma opção:");
            System.out.println("1 - Incluir");
            System.out.println("2 - Alterar");
```

```
            System.out.println("3 - Excluir");
            System.out.println("4 - Exibir pelo ID");
            System.out.println("5 - Exibir todos");
            System.out.println("0 - Sair");

            opcao = scanner.nextInt();
            scanner.nextLine();  // Limpa o buffer do teclado

            switch (opcao) {
                case 1:
                    System.out.println("Tipo de Pessoa (1 - Física, 2 - Jurídica):");
                    int tipoPessoa = scanner.nextInt();
                    scanner.nextLine();  // Limpa o buffer
                    if (tipoPessoa == 1) {
                        // Usando construtor padrão
                        PessoaFisica pf = new PessoaFisica();
                        System.out.println("Nome: ");
                        pf.setNome(scanner.nextLine());
                        System.out.println("Logradouro: ");
                        pf.setLogradouro(scanner.nextLine());
                        System.out.println("Cidade: ");
                        pf.setCidade(scanner.nextLine());
                        System.out.println("Estado: ");
                        pf.setEstado(scanner.nextLine());
                        System.out.println("Telefone: ");
                        pf.setTelefone(scanner.nextLine());
                        System.out.println("Email: ");
                        pf.setEmail(scanner.nextLine());
                        System.out.println("CPF: ");
                        pf.setCpf(scanner.nextLine());
                        try {
```

```
                        pfDAO.incluir(pf);
                        System.out.println("Pessoa Física incluída com
sucesso.");
                    } catch (Exception e) {
                        System.out.println("Erro ao incluir pessoa física: "
+ e.getMessage());
                    }
                } else if (tipoPessoa == 2) {
                    PessoaJuridica pj = new PessoaJuridica();
                    System.out.println("Nome: ");
                    pj.setNome(scanner.nextLine());
                    System.out.println("Logradouro: ");
                    pj.setLogradouro(scanner.nextLine());
                    System.out.println("Cidade: ");
                    pj.setCidade(scanner.nextLine());
                    System.out.println("Estado: ");
                    pj.setEstado(scanner.nextLine());
                    System.out.println("Telefone: ");
                    pj.setTelefone(scanner.nextLine());
                    System.out.println("Email: ");
                    pj.setEmail(scanner.nextLine());
                    System.out.println("CNPJ: ");
                    pj.setCnpj(scanner.nextLine());
                    try {
                        pjDAO.incluir(pj);
                        System.out.println("Pessoa Jurídica incluída com
sucesso.");
                    } catch (Exception e) {
                        System.out.println("Erro ao incluir pessoa jurídica:
" + e.getMessage());
                    }
                }
                break;
```

```
                case 2:
                    // Código de alteração (semelhante ao de inclusão)
                    break;

                case 3:
                    // Código de exclusão
                    break;

                case 4:
                    // Código para exibir pelo ID usando os métodos
getPessoaFisica e getPessoaJuridica
                    break;

                case 5:
                    // Código para exibir todos
                    break;

                case 0:
                    System.out.println("Encerrando o programa...");
                    break;

                default:
                    System.out.println("Opção inválida.");
                    break;
            }
        }

        scanner.close();
    }
```

2. PessoaFisica:

```
package cadastrobd.model;
```

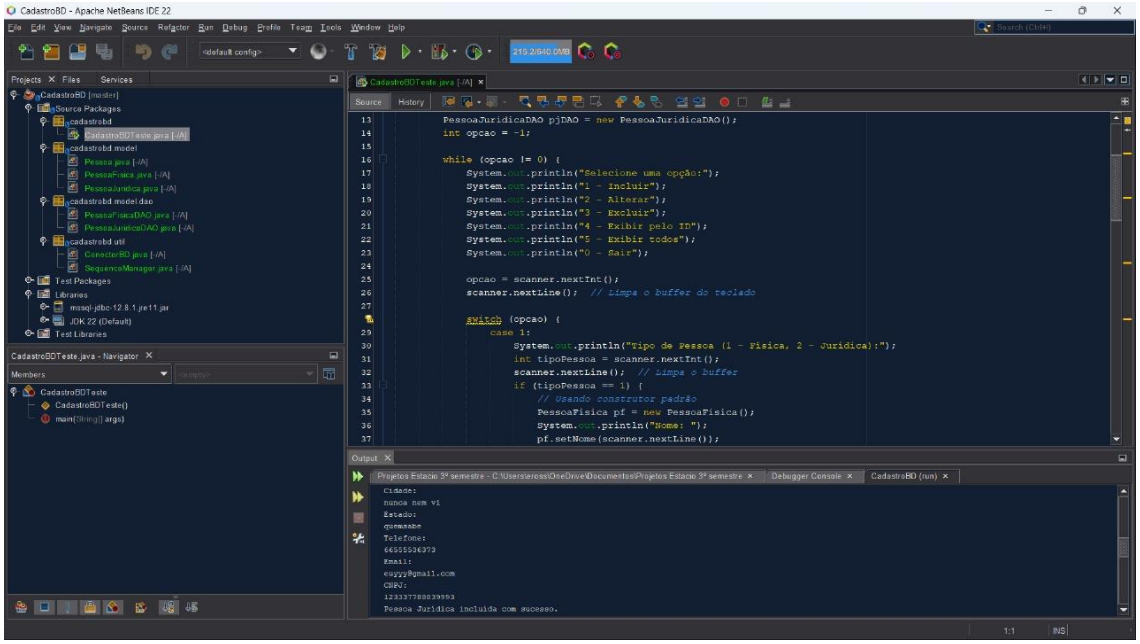
```
public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {
    }

    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;
    }

    public void exibir() {
        System.out.println("Pessoa Física: " + getNome() + ", CPF: " + cpf);
    }
```

3. PessoaJuridica:

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {
    }
```

```
    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); // Chama o
construtor da classe pai
        this.cnpj = cnpj;
    }

    public String getCnpj() {
        return cnpj;
    }

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

    public void exibir() {
        System.out.println("Pessoa Jurídica: " + getNome() + ", CNPJ: " + cnpj);
    }
}
```

Resultado da execução:



a. Diferenças entre a persistência em arquivo e a persistência em banco de dados:

A persistência em arquivos é mais simples de implementar, mas é limitada em termos de controle de concorrência, segurança e escalabilidade. Quando usamos um banco de dados, como no caso do SQL Server, temos a capacidade de manipular grandes volumes de dados com mais eficiência, realizar consultas complexas, garantir integridade referencial entre tabelas e aplicar medidas de segurança robustas, como controle de acessos.

b. Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java? O operador lambda, introduzido nas versões mais recentes do Java, facilita a manipulação e exibição de dados de coleções. Por exemplo, para imprimir todos os dados de uma lista de PessoaFisica, é possível usar:

pessoasFisicas.forEach(pf -> System.out.println(pf.getNome()));

Isso simplifica o código, evitando o uso de loops tradicionais e tornando a leitura e manutenção do código mais fácil.

c. Por que métodos acionados diretamente pelo método main, sem o uso de um

objeto, precisam ser marcados como static?

Os métodos acionados diretamente pelo main precisam ser marcados como static porque o método main também é estático. Isso significa que ele pertence à classe e não a uma instância de um objeto. Como métodos estáticos não têm acesso à instância da classe, outros métodos chamados diretamente a partir do main também devem ser estáticos.

Conclusão

A prática foi fundamental para consolidar os conceitos de persistência de dados com JDBC e o uso de padrão DAO no Java. A implementação do CRUD no banco de dados demonstrou a eficiência e a segurança proporcionadas por um sistema de gerenciamento de banco de dados relacional, quando comparado à persistência em arquivos. O uso de lambda expressions contribuiu para um código mais simples e elegante. A necessidade de marcar métodos como static, devido à estrutura do main, destacou a importância da compreensão dos modificadores de acesso em Java.