

**Campus:**

Norte Shopping – Av. Pres. Vargas 2560, Rio de Janeiro, RJ, 20210-031

**Curso:** Desenvolvimento Full Stack

**Disciplina:** RPG0016 – Back-end Sem Banco Não tem

**Turma:** 9001

**Semestre Letivo:** 2024.3

**Aluno:** Gabriel Bernardo Carneiro

**Matrícula:** 202308953541

**Link repositório GitHub:** [GbDev1907/Missao3Mundo3](https://github.com/GbDev1907/Missao3Mundo3)

## **Título da Prática**

RPG0016 - BackEnd sem banco não tem

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

## **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
6. SQL Server na persistência de dados.

## Códigos solicitados:

### Pessoa.java

```
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() {
        this.id = 0;
        this.nome = "";
        this.logradouro = "";
        this.cidade = "";
        this.estado = "";
        this.telefone = "";
        this.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 void exibir() {
```

```
System.out.println("ID: " + this.id);
System.out.println("Nome: " + this.nome);
System.out.println("Logradouro: " + this.logradouro);
System.out.println("Cidade: " + this.cidade);
System.out.println("Estado: " + this.estado);
System.out.println("Telefone: " + this.telefone);
System.out.println("Email: " + this.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;  
}  
}
```

## PessoaFisica.java

```
package cadastrobd.model;  
  
public class PessoaFisica extends Pessoa {  
    private String cpf;  
  
    public PessoaFisica() {  
        super();  
        this.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);
        this.cpf = cpf;
    }

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

    public String getCpf() {
        return cpf;
    }

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

```

## PessoaFisicadoDAO.java

```

package cadastrobd.model;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;

public class PessoaFisicaDAO {

    public PessoaFisica getPessoa(int id) {
        try {
            Connection conexao = ConectorBD.getConnection();
            if (conexao == null) {

```

```

        return null;
    }
    String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaFisica pf ON
p.idPessoa = pf.idPessoa WHERE p.idPessoa = ?";
    PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
    prepared.setInt(1, id);
    ResultSet resultSet = ConectorBD.getSelect(prepared);
    if (resultSet != null && resultSet.next()) {
        PessoaFisica pessoaFisica = criaPessoaFisica(resultSet);
        ConectorBD.close(resultSet);
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return pessoaFisica;
    }
    ConectorBD.close(prepared);
    ConectorBD.close(conexao);
    return null;
} catch (SQLException e) {
    System.out.println("Erro ao obter pessoa física pelo id: " + e.getMessage());
    return null;
}
}

public List<PessoaFisica> getPessoas() {
    try {
        Connection conexao = ConectorBD.getConnection();
        if (conexao == null) {
            return null;
        }
        String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaFisica pf ON
p.idPessoa = pf.idPessoa";
        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
        ResultSet resultSet = ConectorBD.getSelect(prepared);
        List<PessoaFisica> pessoas = new ArrayList<>();
        while (resultSet != null && resultSet.next()) {
            PessoaFisica pessoaFisica = criaPessoaFisica(resultSet);
            pessoas.add(pessoaFisica);
        }
        ConectorBD.close(resultSet);
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
    }
}

```

```

        return pessoas;
    } catch (SQLException e) {
        System.out.println("Erro ao obter todas as pessoas físicas: " +
e.getMessage());
        return null;
    }
}

public boolean incluir(PessoaFisica pessoaFisica) {
    try {
        Integer nextId = SequenceManager.getValue("PessoaSequence");
        if (nextId == -1) {
            return false;
        }
        pessoaFisica.setId(nextId);
        Connection conexao = ConectorBD.getConnection();
        if (conexao == null) {
            return false;
        }
        String sql = "INSERT INTO Pessoa (idPessoa, nome, telefone, email,
logradouro, cidade, estado) VALUES (?, ?, ?, ?, ?, ?, ?)";
        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setInt(1, pessoaFisica.getId());
        prepared.setString(2, pessoaFisica.getNome());
        prepared.setString(3, pessoaFisica.getTelefone());
        prepared.setString(4, pessoaFisica.getEmail());
        prepared.setString(5, pessoaFisica.getLogradouro());
        prepared.setString(6, pessoaFisica.getCidade());
        prepared.setString(7, pessoaFisica.getEstado());
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }
        sql = "INSERT INTO PessoaFisica (idPessoa, cpf) VALUES (?, ?)";
        prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setInt(1, nextId);
        prepared.setString(2, pessoaFisica.getCpf());
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);

```



```

        return false;
    }
    ConectorBD.close(prepared);
    ConectorBD.close(conexao);
    return true;
} catch (SQLException e) {
    System.out.println("Erro ao incluir pessoa física: " + e.getMessage());
    return false;
}
}

public boolean alterar(PessoaFisica pessoaFisica) {
    try {
        Connection conexao = ConectorBD.getConnection();
        if (conexao == null) {
            return false;
        }
        String sql = "UPDATE Pessoa SET nome = ?, telefone = ?, email = ?,
logradouro = ?, cidade = ?, estado = ? WHERE idPessoa = ?";
        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setString(1, pessoaFisica.getNome());
        prepared.setString(2, pessoaFisica.getTelefone());
        prepared.setString(3, pessoaFisica.getEmail());
        prepared.setString(4, pessoaFisica.getLogradouro());
        prepared.setString(5, pessoaFisica.getCidade());
        prepared.setString(6, pessoaFisica.getEstado());
        prepared.setInt(7, pessoaFisica.getId());
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }
        sql = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa = ?";
        prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setString(1, pessoaFisica.getCpf());
        prepared.setInt(2, pessoaFisica.getId());
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }
    }
}

```

```

        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return true;
    } catch (SQLException e) {
        System.out.println("Erro ao alterar pessoa física: " + e.getMessage());
        return false;
    }
}

public boolean excluir(int id) {
    try {
        Connection conexao = ConectorBD.getConnection();
        if (conexao == null) {
            return false;
        }
        String sql = "DELETE FROM PessoaFisica WHERE idPessoa = ?";
        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setInt(1, id);
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }
        sql = "DELETE FROM Pessoa WHERE idPessoa = ?";
        prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setInt(1, id);
        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return true;
    } catch (SQLException e) {
        System.out.println("Erro ao excluir pessoa física: " + e.getMessage());
        return false;
    }
}

```

```
private PessoaFisica criaPessoaFisica(ResultSet resultSet) throws SQLException
{
    PessoaFisica pessoaFisica = new PessoaFisica();
    pessoaFisica.setId(resultSet.getInt("idPessoa"));
    pessoaFisica.setNome(resultSet.getString("nome"));
    pessoaFisica.setTelefone(resultSet.getString("telefone"));
    pessoaFisica.setEmail(resultSet.getString("email"));
    pessoaFisica.setLogradouro(resultSet.getString("logradouro"));
    pessoaFisica.setCidade(resultSet.getString("cidade"));
    pessoaFisica.setEstado(resultSet.getString("estado"));
    pessoaFisica.setCpf(resultSet.getString("cpf"));
    return pessoaFisica;
}
}
```

## PessoaJuridica.java

```
package cadastrbd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {
        super();
        this.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);
        this.cnpj = cnpj;
    }

    @Override
    public void exibir() {
        super.exibir();

        System.out.println("CNPJ: " + this.cnpj);
    }
}
```

```
}

public String getCnpj() {
    return cnpj;
}

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

### **PessoaJuridicaDAO.java**

```
package cadastrabd.model;

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

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

public class PessoaJuridicaDAO {

    public PessoaJuridica getPessoa(int id) {
        try{
```

```

        Connection conexao = ConectorBD.getConnection();

        if (conexao == null) {

            return null;

        }

        String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaJuridica pj ON p.idPessoa
= pj.idPessoa WHERE p.idPessoa = ?";

        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);

        prepared.setInt(1, id);

        ResultSet resultSet = ConectorBD.getSelect(prepared);

        if (resultSet != null && resultSet.next()) {

            PessoaJuridica pessoaJuridica = criaPessoaJuridica(resultSet);

            ConectorBD.close(resultSet);

            ConectorBD.close(prepared);

            ConectorBD.close(conexao);

            return pessoaJuridica;

        }

        ConectorBD.close(prepared);

        ConectorBD.close(conexao);

        return null;

    } catch (SQLException e) {

        System.out.println("Erro ao obter pessoa jurídica pelo id: " + e.getMessage());

        return null;

    }

}

public List<PessoaJuridica> getPessoas() {

    try{

        Connection conexao = ConectorBD.getConnection();

        if (conexao == null) {

            return null;

```

```

    }

    String sql = "SELECT * FROM Pessoa p INNER JOIN PessoaJuridica pj ON p.idPessoa
= pj.idPessoa";

    PreparedStatement prepared = conexao.prepareStatement(sql);

    ResultSet resultSet = ConectorBD.getSelect(prepared);

    List<PessoaJuridica> pessoas = new ArrayList<>();

    while (resultSet != null && resultSet.next()) {

        PessoaJuridica pessoaJuridica = criaPessoaJuridica(resultSet);

        pessoas.add(pessoaJuridica);

    }

    ConectorBD.close(resultSet);

    ConectorBD.close(prepared);

    ConectorBD.close(conexao);

    return pessoas;

} catch (SQLException e) {

    System.out.println("Erro ao obter todas as pessoas jurídicas: " + e.getMessage());

    return null;

}

}

```

```

public boolean incluir(PessoaJuridica pessoaJuridica) {

    try{

        Integer nextId = SequenceManager.getValue("PessoaSequence");

        if (nextId == -1) {

            return false;

        }

        pessoaJuridica.setId(nextId);

        Connection conexao = ConectorBD.getConnection();

        if (conexao == null) {

            return false;

        }

    }
}

```

```

    }

    String sql = "INSERT INTO Pessoa(idPessoa, nome, telefone, email, logradouro,
cidade, estado) VALUES (?, ?, ?, ?, ?, ?, ?)";

    PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);

    prepared.setInt(1, pessoaJuridica.getId());
    prepared.setString(2, pessoaJuridica.getNome());
    prepared.setString(3, pessoaJuridica.getTelefone());
    prepared.setString(4, pessoaJuridica.getEmail());
    prepared.setString(5, pessoaJuridica.getLogradouro());
    prepared.setString(6, pessoaJuridica.getCidade());
    prepared.setString(7, pessoaJuridica.getEstado());

    if (prepared.executeUpdate() <= 0) {
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return false;
    }

    sql = "INSERT INTO PessoaJuridica(idPessoa, cnpj) VALUES (?, ?)";
    prepared = ConectorBD.getPrepared(conexao, sql);
    prepared.setInt(1, nextId);
    prepared.setString(2, pessoaJuridica.getCnpj());

    if (prepared.executeUpdate() <= 0) {
        ConectorBD.close(prepared);
        ConectorBD.close(conexao);
        return false;
    }

    ConectorBD.close(prepared);
    ConectorBD.close(conexao);

    return true;
} catch (SQLException e) {

    System.out.println("Erro ao incluir pessoa jurídica: " + e.getMessage());

```

```

        return false;
    }
}

public boolean alterar(PessoaJuridica pessoaJuridica) {
    try {
        Connection conexao = ConectorBD.getConnection();

        if (conexao == null) {
            return false;
        }

        String sql = "UPDATE Pessoa SET nome = ?, telefone = ?, email = ?, logradouro = ?,
cidade = ?, estado = ? WHERE idPessoa = ?";

        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);

        prepared.setString(1, pessoaJuridica.getNome());
        prepared.setString(2, pessoaJuridica.getTelefone());
        prepared.setString(3, pessoaJuridica.getEmail());
        prepared.setString(4, pessoaJuridica.getLogradouro());
        prepared.setString(5, pessoaJuridica.getCidade());
        prepared.setString(6, pessoaJuridica.getEstado());
        prepared.setInt(7, pessoaJuridica.getId());

        if (prepared.executeUpdate() <= 0) {
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return false;
        }

        sql = "UPDATE PessoaJuridica SET cnpj = ? WHERE idPessoa = ?";
        prepared = ConectorBD.getPrepared(conexao, sql);
        prepared.setString(1, pessoaJuridica.getCnpj());
        prepared.setInt(2, pessoaJuridica.getId());
    }
}

```



```

        if (prepared.executeUpdate() <= 0) {

            ConectorBD.close(prepared);

            ConectorBD.close(conexao);

            return false;

        }

        ConectorBD.close(prepared);

        ConectorBD.close(conexao);

        return true;

    } catch (SQLException e) {

        System.out.println("Erro ao alterar pessoa jurídica: " + e.getMessage());

        return false;

    }

}

public boolean excluir(int id) {

    try{

        Connection conexao = ConectorBD.getConnection();

        if (conexao == null) {

            return false;

        }

        String sql = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";

        PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);

        prepared.setInt(1, id);

        if (prepared.executeUpdate() <= 0) {

            ConectorBD.close(prepared);

            ConectorBD.close(conexao);

            return false;

        }

        sql = "DELETE FROM Pessoa WHERE idPessoa = ?";

        prepared = ConectorBD.getPrepared(conexao, sql);

```

```

        prepared.setInt(1, id);

        if (prepared.executeUpdate() <= 0) {

            ConectorBD.close(prepared);

            ConectorBD.close(conexao);

            return false;

        }

        ConectorBD.close(prepared);

        ConectorBD.close(conexao);

        return true;

    } catch (SQLException e) {

        System.out.println("Erro ao excluir pessoa jurídica: " + e.getMessage());

        return false;

    }

}

private static PessoaJuridica criaPessoaJuridica(ResultSet resultSet) throws
SQLException {

    PessoaJuridica pessoaJuridica = new PessoaJuridica();

    pessoaJuridica.setId(resultSet.getInt("idPessoa"));

    pessoaJuridica.setNome(resultSet.getString("nome"));

    pessoaJuridica.setTelefone(resultSet.getString("telefone"));

    pessoaJuridica.setEmail(resultSet.getString("email"));

    pessoaJuridica.setLogradouro(resultSet.getString("logradouro"));

    pessoaJuridica.setCidade(resultSet.getString("cidade"));

    pessoaJuridica.setEstado(resultSet.getString("estado"));

    pessoaJuridica.setCnpj(resultSet.getString("cnpj"));

    return pessoaJuridica;

}

}

```

## ConectorBD.java

```
package cadastro.model.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 DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";

    private static final String URL =

"jdbc:sqlserver://localhost\\MSSQL:1433;databaseName=loja;encrypt=true;trustServerCertificate=true;";

    private static final String USER = "loja";

    private static final String PASSWORD = "loja";

    public static Connection getConnection() {

        try{

            // Carrega o driver JDBC na memória

            Class.forName(DRIVER).newInstance();

            // Retorna uma conexão com o banco de dados

            return DriverManager.getConnection(URL, USER, PASSWORD);

        } catch (ClassNotFoundException | SQLException e) {

            System.out.println("Erro ao conectar com o banco de dados: " + e.getMessage());

            return null;

        } catch (InstantiationException e) {
```

```

        throw new RuntimeException(e);
    } catch (IllegalAccessException e) {
        throw new RuntimeException(e);
    }
}

public static PreparedStatement getPrepared(Connection conexao, String sql) {
    try {
        return conexao.prepareStatement(sql);
    } catch (SQLException e) {
        System.out.println("Erro ao preparar o SQL: " + e.getMessage());
        return null;
    }
}

public static ResultSet getSelect(PreparedStatement consulta) {
    try {
        return consulta.executeQuery();
    } catch (SQLException e) {
        System.out.println("Erro ao executar a consulta: " + e.getMessage());
        return null;
    }
}

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

```

```
        System.out.println("Erro ao fechar o Statement: " + e.getMessage());
    }
}

public static void close(ResultSet resultado) {
    try{
        if (resultado != null) {
            resultado.close();
        }
    } catch (SQLException e) {
        System.out.println("Erro ao fechar o ResultSet: " + e.getMessage());
    }
}

public static void close(Connection con) {
    try{
        if (con != null) {
            con.close();
        }
    } catch (SQLException e) {
        System.out.println("Erro ao fechar a conexão: " + e.getMessage());
    }
}
}
```

## SequenceManager.java

```
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 sequence) {
        try {
            Connection conexao = ConectorBD.getConnection();

            if (conexao == null) {
                return -1;
            }

            String sql = "SELECT NEXT VALUE FOR dbo." + sequence;

            PreparedStatement consulta = ConectorBD.getPrepared(conexao, sql);

            ResultSet resultado = ConectorBD.getSelect(consulta);

            if (resultado == null || !resultado.next()) {
                ConectorBD.close(conexao);

                return -1;
            }

            int value = resultado.getInt(1);

            ConectorBD.close(resultado);

            ConectorBD.close(consulta);

            ConectorBD.close(conexao);
        }
    }
}
```

```

        return value;

    } catch (SQLException e) {

        System.out.println("Erro ao obter o valor da sequência: " + e.getMessage());

        return -1;

    }

}

}

}

```

## CadastroBD.java

```

package cadastrobd;

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

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

public class CadastroBD {

    private static Scanner scanner = new Scanner(System.in);

    private static PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();

    private static PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    public static void main(String[] args) {

        int opcao = -1;

        while (opcao != 0) {

            System.out.println("=====");

            System.out.println("Selecione uma opção:");

```

```
System.out.println("1 - Incluir Pessoa");

System.out.println("2 - Alterar Pessoa");

System.out.println("3 - Excluir Pessoa");

System.out.println("4 - Exibir pelo id");

System.out.println("5 - Exibir todos");

System.out.println("0 - Finalizar a execução");

opcao = Integer.parseInt(scanner.nextLine());

System.out.println("=====");

switch (opcao) {

    case 1:

        inserirPessoa();

        break;

    case 2:

        alterarPessoa();

        break;

    case 3:

        excluirPessoa();

        break;

    case 4:

        exibirPessoaPeloid();

        break;

    case 5:

        exibirTodasAsPessoas();

        break;

}

System.out.println();

}

}
```



```
private static String lerTipoDePessoa() {

    System.out.println("Escolha o tipo:\n\tPara Pessoa Física digite F\n\tPara Pessoa Jurídica digite J");

    String tipo = scanner.nextLine();

    System.out.println("=====\n");

    if (tipo.equalsIgnoreCase("F") || tipo.equalsIgnoreCase("J")) {

        return tipo;

    } else {

        System.out.println("Opção inválida, tente novamente.");

        return lerTipoDePessoa();

    }

}

private static PessoaFisica definirDadosPessoaFisica(PessoaFisica pessoaFisica) {

    try{

        System.out.println("Digite o nome:");

        pessoaFisica.setNome(scanner.nextLine());

        System.out.println("Digite o CPF:");

        pessoaFisica.setCpf(scanner.nextLine());

        System.out.println("Digite o telefone:");

        pessoaFisica.setTelefone(scanner.nextLine());

        System.out.println("Digite o email:");

        pessoaFisica.setEmail(scanner.nextLine());

        System.out.println("Digite o logradouro:");

        pessoaFisica.setLogradouro(scanner.nextLine());

        System.out.println("Digite a cidade:");

        pessoaFisica.setCidade(scanner.nextLine());

        System.out.println("Digite o estado:");

        pessoaFisica.setEstado(scanner.nextLine());

        return pessoaFisica;

    }
```

```
} catch (Exception e) {

    System.out.println("Erro ao inserir os dados da Pessoa Física:");

    e.printStackTrace();

    System.out.println("Por favor, tente novamente.");

    return null;

}

}

private static PessoaJuridica definirDadosPessoaJuridica(PessoaJuridica
pessoaJuridica) {

    try {

        System.out.println("Digite o nome:");

        pessoaJuridica.setNome(scanner.nextLine());

        System.out.println("Digite o CNPJ:");

        pessoaJuridica.setCnpj(scanner.nextLine());

        System.out.println("Digite o telefone:");

        pessoaJuridica.setTelefone(scanner.nextLine());

        System.out.println("Digite o email:");

        pessoaJuridica.setEmail(scanner.nextLine());

        System.out.println("Digite o logradouro:");

        pessoaJuridica.setLogradouro(scanner.nextLine());

        System.out.println("Digite a cidade:");

        pessoaJuridica.setCidade(scanner.nextLine());

        System.out.println("Digite o estado:");

        pessoaJuridica.setEstado(scanner.nextLine());

        return pessoaJuridica;

    } catch (Exception e) {

        System.out.println("Erro ao inserir os dados da Pessoa Jurídica:");

        e.printStackTrace();

        System.out.println("Por favor, tente novamente.");

    }

}
```

```
        return null;
    }
}

private static void inserirPessoa() {
    String tipo = lerTipoDePessoa();
    if (tipo.equalsIgnoreCase("F")) {
        PessoaFisica pessoaFisica = definirDadosPessoaFisica(new PessoaFisica());
        if (pessoaFisica == null) {
            System.out.println("Falha ao incluir Pessoa Física.");
            return;
        }
        if (!pessoaFisicaDAO.incluir(pessoaFisica)) {
            System.out.println("Falha ao incluir Pessoa Física.");
            return;
        }
        System.out.println("Pessoa Física incluída com sucesso.");
        return;
    }
    if (tipo.equalsIgnoreCase("J")) {
        PessoaJuridica pessoaJuridica = definirDadosPessoaJuridica(new PessoaJuridica());
        if (pessoaJuridica == null) {
            System.out.println("Falha ao incluir Pessoa Jurídica.");
            return;
        }
        if (!pessoaJuridicaDAO.incluir(pessoaJuridica)) {
            System.out.println("Falha ao incluir Pessoa Jurídica.");
            return;
        }
        System.out.println("Pessoa Jurídica incluída com sucesso.");
    }
}
```

```

    }
}

private static void alterarPessoa() {
    String tipo = lerTipoDePessoa();
    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja alterar:");
        int idPessoaFisica = Integer.parseInt(scanner.nextLine());
        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);
        if (pessoaFisica == null) {
            System.out.println("Pessoa Física não encontrada.");
            return;
        }
        pessoaFisica.exibir();
        pessoaFisica = definirDadosPessoaFisica(pessoaFisica);
        if (pessoaFisica == null) {
            System.out.println("Falha ao alterar Pessoa Física.");
            return;
        }
        if (!pessoaFisicaDAO.alterar(pessoaFisica)) {
            System.out.println("Falha ao alterar Pessoa Física.");
            return;
        }
        System.out.println("Pessoa Física alterada com sucesso.");
        return;
    }
    if (tipo.equalsIgnoreCase("J")) {
        System.out.println("Digite o id da Pessoa Jurídica que deseja alterar:");
        int idPessoaJuridica = Integer.parseInt(scanner.nextLine());
        PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);
    }
}

```

```
        if (pessoaJuridica == null) {

            System.out.println("Pessoa Jurídica não encontrada.");

            return;

        }

        pessoaJuridica.exibir();

        pessoaJuridica = definirDadosPessoaJuridica(pessoaJuridica);

        if (pessoaJuridica == null) {

            System.out.println("Falha ao alterar Pessoa Jurídica.");

            return;

        }

        if (!pessoaJuridicaDAO.alterar(pessoaJuridica)) {

            System.out.println("Falha ao alterar Pessoa Jurídica.");

            return;

        }

        System.out.println("Pessoa Jurídica alterada com sucesso.");

    }

}

private static void excluirPessoa() {

    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {

        System.out.println("Digite o id da Pessoa Física que deseja excluir:");

        int idPessoaFisica = Integer.parseInt(scanner.nextLine());

        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);

        if (pessoaFisica == null) {

            System.out.println("Pessoa Física não encontrada.");

            return;

        }

        if (!pessoaFisicaDAO.excluir(idPessoaFisica)) {

            System.out.println("Falha ao excluir Pessoa Física.");

        }

    }

}
```

```

        return;
    }

    System.out.println("Pessoa Física excluída com sucesso.");

    return;
}

if (tipo.equalsIgnoreCase("J")) {

    System.out.println("Digite o id da Pessoa Jurídica que deseja excluir:");

    int idPessoaJuridica = Integer.parseInt(scanner.nextLine());

    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);

    if (pessoaJuridica == null) {

        System.out.println("Pessoa Jurídica não encontrada.");

        return;
    }

    if (!pessoaJuridicaDAO.excluir(idPessoaJuridica)) {

        System.out.println("Falha ao excluir Pessoa Jurídica.");

        return;
    }

    System.out.println("Pessoa Jurídica excluída com sucesso.");

}

}

private static void exibirPessoaPeloid() {

    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {

        System.out.println("Digite o id da Pessoa Física que deseja exibir:");

        int idPessoaFisica = Integer.parseInt(scanner.nextLine());

        PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(idPessoaFisica);

        if (pessoaFisica == null) {

            System.out.println("Pessoa Física não encontrada.");

            return;
        }
    }
}

```

```

    }

    pessoaFisica.exibir();

    return;
}

if (tipo.equalsIgnoreCase("J")) {

    System.out.println("Digite o id da Pessoa Jurídica que deseja exibir:");

    int idPessoaJuridica = Integer.parseInt(scanner.nextLine());

    PessoaJuridica pessoaJuridica = pessoaJuridicaDAO.getPessoa(idPessoaJuridica);

    if (pessoaJuridica == null) {

        System.out.println("Pessoa Jurídica não encontrada.");

        return;

    }

    pessoaJuridica.exibir();

}

}

private static void exibirTodasAsPessoas() {

    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {

        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();

        if (pessoasFisicas == null || pessoasFisicas.isEmpty()) {

            System.out.println("Não existem Pessoas Físicas cadastradas.");

            return;

        }

        System.out.println("Exibindo todos os registros de Pessoas Físicas:\n");

        for (PessoaFisica pessoaFisica : pessoasFisicas) {

            System.out.println("-----");

            pessoaFisica.exibir();

        }

        return;
    }
}

```

```
}  
  
if (tipo.equalsIgnoreCase("J")) {  
    List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();  
  
    if (pessoasJuridicas == null || pessoasJuridicas.isEmpty()) {  
        System.out.println("Não existem Pessoas Jurídicas cadastradas.");  
        return;  
    }  
  
    System.out.println("Exibindo todos os registros de Pessoas Jurídicas:");  
  
    for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {  
        System.out.println("-----");  
        pessoaJuridica.exibir();  
    }  
}  
  
}
```



Resultado da saída do código:

```
Exibindo todos os registros de Pessoas Juridicas.
-----
ID: 2
Nome: JJC
Logradouro: Rua 11, Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 1212-1212
Email: jjc@riacho.com
CNPJ: 222222222222222
-----
ID: 13
Nome: Companhia das coisas
Logradouro: Rua 11
Cidade: Cidade Nova
Estado: PA
Telefone: 123456789
Email: companhia@gmail.com
CNPJ: 123456789
-----
Selecione uma opção:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execução
0
-----
Escolha o tipo:
--- Para Pessoa Fisica digite F
--- Para Pessoa Juridica digite J
J
-----
Digite o id da Pessoa Juridica que deseja exibir:
13
ID: 13
Nome: Companhia das coisas
Logradouro: Rua 11
Cidade: Cidade Nova
Estado: PA
Telefone: 123456789
Email: companhia@gmail.com
CNPJ: 123456789
```

```
-- Lista dados completos de pessoas físicas:
SELECT * FROM Pessoa p JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa;
GO

-- Lista dados completos de pessoas jurídicas:
SELECT * FROM Pessoa p JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa;
GO
```

100 %

# Results 0 Messages

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	CPF
1	1	Joao	Rua 12, casa 3, Quitanda	Riacho do Sul	PA	1111-1111	joao@riacho.com	1	111111111111
2	12	Maria da Silva	Rua das Mulas Casas	Cidade 2	AC	987654321	maria.da.silva@gmail.com	12	123456789

	idPessoa	nome	logradouro	cidade	estado	telefone	email	idPessoa	CNPJ
1	2	JJC	Rua 11, Centro	Riacho do Norte	PA	1212-1212	jjc@riacho.com	2	222222222222222
2	13	Companhia das coisas	Rua 11	Cidade Nova	PA	123456789	companhia@gmail.com	13	123456789

## **5. Análise e Conclusão**

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

Os componentes de middleware, como o JDBC, são super importantes porque atuam como intermediários entre a aplicação e o banco de dados. O JDBC, por exemplo, facilita a comunicação, permitindo que as aplicações Java acessem e manipulem dados de maneira simples. Ele cuida de toda a parte de conexão, execução de comandos e gerenciamento de resultados, o que economiza tempo e esforço dos desenvolvedores. Resumindo, eles tornam a integração entre sistemas mais fácil e eficiente.

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

A diferença entre Statement e PreparedStatement está na execução de consultas SQL. Statement envia o SQL para o banco de dados toda vez, sendo mais lento e menos seguro. Já PreparedStatement prepara a consulta uma vez e a reutiliza com diferentes parâmetros, o que melhora o desempenho e oferece mais segurança contra SQL Injection. Para segurança e eficiência, prefira PreparedStatement.

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

O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao separar a lógica de acesso a dados da lógica de negócio. Isso significa que você pode mudar como os dados são armazenados (por exemplo, trocar de banco de dados) sem precisar alterar a lógica da aplicação. Além disso, o DAO centraliza as operações de acesso a dados, tornando o código mais organizado e fácil de entender. Com isso, fica mais simples fazer alterações, testar e até manter o código no futuro.

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

Em um modelo relacional, a herança pode ser representada de duas maneiras: uma é usando uma tabela por classe, onde cada classe tem sua própria tabela, com a tabela da classe "pai" armazenando atributos comuns e as "filhas" armazenando atributos específicos que referenciam a tabela "pai". A outra é usar uma tabela única, onde todas as classes (pai e filhas) são armazenadas em uma única tabela com todos os atributos, mas colunas que não se aplicam ficam

vazias. Cada abordagem tem seus prós e contras, e a escolha depende das necessidades do projeto.

## **5. Análise e Conclusão (segunda parte)**

### **A. Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?**

A persistência em arquivo e a persistência em banco de dados têm algumas diferenças importantes. A persistência em arquivo envolve armazenar dados em arquivos de texto ou binários, o que é simples e direto, mas pode ser difícil de gerenciar e escalar, especialmente com grandes volumes de dados ou quando múltiplos usuários precisam acessar as informações simultaneamente. Já a persistência em banco de dados organiza os dados em tabelas, permitindo consultas complexas, relacionamentos e transações seguras, além de oferecer suporte a múltiplos usuários e garantir integridade dos dados. Em resumo, enquanto arquivos são mais simples, bancos de dados são mais robustos e adequados para aplicações que exigem maior controle e eficiência.

### **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 nas versões mais recentes do Java simplificou muito a impressão dos valores nas entidades. Antes, você precisava de loops e classes anônimas para iterar sobre listas e imprimir os dados, o que tornava o código mais verboso e complicado. Com lambdas, você pode usar métodos como `forEach`, passando uma expressão simples para definir o que fazer com cada item da lista. Isso deixa o código mais limpo e legível, permitindo que você escreva menos e foque no que realmente importa.

### **C. Por que os métodos acionados diretamente pelo método `main`, sem o uso de um objeto, precisam ser marcados como `static`?**

Os métodos acionados diretamente pelo método `main` precisam ser marcados como `static` porque o `main` é um método estático e não está associado a uma instância específica de uma classe. Isso significa que ele pode ser chamado sem precisar criar um objeto da classe. Se você tentar chamar um método não estático dentro do `main`, o Java não saberá a qual instância ele pertence, resultando em um erro. Portanto, para que o método possa ser acessado diretamente, ele precisa ser `static`, assim pode ser chamado diretamente sem a necessidade de criar um objeto.

