UNIVERSIDADE ESTACIO DE SÁ

POLO TAMBIÁ

DESENVOLVIMENTO FULL STACK

2024.1 FULL STACK

NIVEL 3: BACKEND SEM BANCO NAO TEM!
MUNDO: 3

FRANCINALDO SOUZA BERNARDINO

Centro Universitário Estácio de Sá – Paraíba

Polo; Joao Pessoa - Tambiá

Curso: Desenvolvimento Full Stack

Disciplina: RPG0016 **Nível 3:**BackEnd sem banco não tem **Mundo:** 3

Número da Turma: 9003

Semestre Letivo: 3º

Aluno: Francinaldo Souza Bernardino

Repositorio Git: https://github.com/Anubyhs/N3M3

Título da Prática

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 SQL Server na persistência de dados.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

Pessoa.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastrobd.model;

/**
 *
 * @Francinaldo
 */
public class Pessoa {

   private int id;
   private String nome;
   private String rua;
   private String cidade;
   private String estado;
   private String telefone;
   private String email;

    public Pessoa() {
    }

    public Pessoa(Integer id, String nome, String rua, String cidade, String estado, String
telefone, String email) {
        this.id = id;
        this.nome = nome;
        this.rua = rua;
        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("Rua: " + this.rua);
        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 getRua() {
        return rua;
    }

    public void setRua(String rua) {
        this.rua = rua;
    }

    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;
    }
}
```

PessoasFisicas.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastrobd.model;

/**
 *
 * @Francinaldo
 */
public class PessoasFisicas extends Pessoa{
    private String cpf;

    public PessoasFisicas() {
    }

public PessoasFisicas(Integer id, String nome, String rua, String cidade, String estado, String
telefone, String email, String cpf) {
    super(id, nome, rua, 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;
    }

}
```

PessoaJuridica.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastrobd.model;

/**
 *
 * @Francinaldo
 */
public class PessoaJuridica extends Pessoa{
    private String cnpj;

    public PessoaJuridica() {
    }

    public PessoaJuridica(Integer id, String nome, String rua, String cidade, String estado, String telefone,
String email) {
        super(id, nome, rua, cidade, estado, telefone, email);
    }

    @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;
    }

}
```

PessoaFisicaDAO.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastrobd.model;

/**
 *
 * @Francinaldo
 */
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 cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;

public class PessoasFisicasDAO {
   public PessoasFisicas getPessoa(int id) {
      try {
         Connection conexao = ConectorBD.getConnection();

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

         String sql = "SELECT IDPessoa, NomePessoa, Telefone, Email, Logradouro, Cidade,
Estado, CPF " +
                  "FROM dbo.Pessoas " +
                  "WHERE IDPessoa = ?";

         PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
         prepared.setInt(1, id);

         ResultSet resultSet = ConectorBD.getSelect(prepared);

         if (resultSet != null && resultSet.next()) {
            PessoasFisicas pessoasFisicas = criaPessoasFisicas(resultSet);

            ConectorBD.close(resultSet);
            ConectorBD.close(prepared);
            ConectorBD.close(conexao);
            return pessoasFisicas;
         }

         ConectorBD.close(prepared);
         ConectorBD.close(conexao);
         return null;
      } catch (SQLException e) {
```

```
            System.out.println("Erro ao obter a pessoa física pelo id: " + e.getMessage());
            return null;
        }
    }

    public List<PessoasFisicas> getPessoas() {
        try {
            Connection conexao = ConectorBD.getConnection();

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

            String sql = "SELECT IDPessoa, NomePessoa, Telefone, Email, Logradouro, Cidade,
Estado, CPF " +
                    "FROM dbo.Pessoas";

            PreparedStatement prepared = conexao.prepareStatement(sql);

            ResultSet resultSet = ConectorBD.getSelect(prepared);

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

            while (resultSet != null && resultSet.next()) {
                PessoasFisicas pessoasFisicas = criaPessoasFisicas(resultSet);
                pessoas.add(pessoasFisicas);
            }

            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(PessoasFisicas pessoasFisicas) {
        Connection conexao = null;
        PreparedStatement preparedPessoa = null;

        try {
            Integer nextId = SequenceManager.getValue("CodigoPessoa");

            if (nextId == -1) {
                System.out.println("Falha ao obter o próximo ID para PessoasFisicas.");
                return false;
            }

            pessoasFisicas.setId(nextId);
            conexao = ConectorBD.getConnection();

            if (conexao == null) {
```

```
            return false;
        }

        String sqlPessoa = "INSERT INTO dbo.Pessoas (IDPessoa, NomePessoa, Telefone,
Email, Logradouro, Cidade, Estado, CPF) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa);
        preparedPessoa.setInt(1, pessoasFisicas.getId());
        preparedPessoa.setString(2, pessoasFisicas.getNome());
        preparedPessoa.setString(3, pessoasFisicas.getTelefone());
        preparedPessoa.setString(4, pessoasFisicas.getEmail());
        preparedPessoa.setString(5, pessoasFisicas.getRua());
        preparedPessoa.setString(6, pessoasFisicas.getCidade());
        preparedPessoa.setString(7, pessoasFisicas.getEstado());
        preparedPessoa.setString(8, pessoasFisicas.getCpf());

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

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

public boolean alterar(PessoasFisicas pessoasFisicas) {
    Connection conexao = null;
    PreparedStatement preparedPessoa = null;

    try {
        conexao = ConectorBD.getConnection();

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

        String sqlPessoa = "UPDATE dbo.Pessoas SET NomePessoa = ?, Telefone = ?, Email =
?, Logradouro = ?, Cidade = ?, Estado = ?, CPF = ? WHERE IDPessoa = ?";
        preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa);
        preparedPessoa.setString(1, pessoasFisicas.getNome());
        preparedPessoa.setString(2, pessoasFisicas.getTelefone());
        preparedPessoa.setString(3, pessoasFisicas.getEmail());
        preparedPessoa.setString(4, pessoasFisicas.getRua());
        preparedPessoa.setString(5, pessoasFisicas.getCidade());
        preparedPessoa.setString(6, pessoasFisicas.getEstado());
        preparedPessoa.setString(7, pessoasFisicas.getCpf());
        preparedPessoa.setInt(8, pessoasFisicas.getId());
```

```
            if (preparedPessoa.executeUpdate() <= 0) {
                ConectorBD.close(preparedPessoa);
                ConectorBD.close(conexao);
                return false;
            }
            ConectorBD.close(preparedPessoa);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            System.out.println("Erro ao alterar a pessoa física: " + e.getMessage());
            return false;
        } finally {
            ConectorBD.close(preparedPessoa);
            ConectorBD.close(conexao);
        }
    }

    public boolean excluir(int id) {
        Connection conexao = null;
        PreparedStatement preparedPessoa = null;

        try {
            conexao = ConectorBD.getConnection();

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

            String sqlPessoa = "DELETE FROM dbo.Pessoas WHERE IDPessoa = ?";
            preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa);
            preparedPessoa.setInt(1, id);

            if (preparedPessoa.executeUpdate() <= 0) {
                ConectorBD.close(preparedPessoa);
                ConectorBD.close(conexao);
                return false;
            }
            ConectorBD.close(preparedPessoa);
            ConectorBD.close(conexao);
            return true;
        } catch (SQLException e) {
            System.out.println("Erro ao excluir a pessoa física: " + e.getMessage());
            return false;
        } finally {
            ConectorBD.close(preparedPessoa);
            ConectorBD.close(conexao);
        }
    }

    private static PessoasFisicas criaPessoasFisicas(ResultSet resultSet) throws SQLException {
        PessoasFisicas pessoasFisicas = new PessoasFisicas();
        pessoasFisicas.setId(resultSet.getInt("IDPessoa"));
        pessoasFisicas.setNome(resultSet.getString("NomePessoa"));
        pessoasFisicas.setTelefone(resultSet.getString("Telefone"));
```

```
        pessoasFisicas.setEmail(resultSet.getString("Email"));
        pessoasFisicas.setRua(resultSet.getString("Logradouro"));
        pessoasFisicas.setCidade(resultSet.getString("Cidade"));
        pessoasFisicas.setEstado(resultSet.getString("Estado"));
        pessoasFisicas.setCpf(resultSet.getString("CPF"));
        return pessoasFisicas;
    }
}
```

PessoaJuridicaDAO.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 cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;
import java.util.logging.Logger;
import java.util.logging.Level;

public class PessoaJuridicaDAO {
   private static final Logger LOGGER = Logger.getLogger(PessoaJuridicaDAO.class.getName());

   public PessoaJuridica getPessoa(int id) {
      String sql = "SELECT p.IDPessoa, p.NomePessoa, p.Telefone, p.Email, p.Logradouro, p.Cidade,
p.Estado, pj.CNPJ " +
            "FROM dbo.Pessoas p INNER JOIN dbo.PessoasJuridicas pj ON p.IDPessoa = pj.IDPessoa
WHERE p.IDPessoa = ?";
      try (Connection conexao = ConectorBD.getConnection();
          PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql)) {

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

         prepared.setInt(1, id);
         try (ResultSet resultSet = ConectorBD.getSelect(prepared)) {
            if (resultSet.next()) {
               return criaPessoaJuridica(resultSet);
            }
         }
      } catch (SQLException e) {
         LOGGER.log(Level.SEVERE, "Erro ao obter a pessoa jurídica pelo id: " + id, e);
      }
      return null;
   }

   public List<PessoaJuridica> getPessoas() {
      String sql = "SELECT p.IDPessoa, p.NomePessoa, p.Telefone, p.Email, p.Logradouro, p.Cidade,
p.Estado, pj.CNPJ " +
            "FROM dbo.Pessoas p INNER JOIN dbo.PessoasJuridicas pj ON p.IDPessoa = pj.IDPessoa";
      List<PessoaJuridica> pessoas = new ArrayList<>();
      try (Connection conexao = ConectorBD.getConnection();
          PreparedStatement prepared = ConectorBD.getPrepared(conexao, sql);
          ResultSet resultSet = ConectorBD.getSelect(prepared)) {

         if (conexao == null) {
            return pessoas;
         }
```

```
            while (resultSet.next()) {
                pessoas.add(criaPessoaJuridica(resultSet));
            }
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, "Erro ao obter todas as pessoas jurídicas", e);
        }
        return pessoas;
    }

    public boolean incluir(PessoaJuridica pessoaJuridica) {
        Integer nextId = SequenceManager.getValue("CodigoPessoa");
        if (nextId == -1) {
            return false;
        }
        pessoaJuridica.setId(nextId);

        Connection conexao = null;
        try {
            conexao = ConectorBD.getConnection();
            if (conexao == null) {
                return false;
            }

            conexao.setAutoCommit(false);

            if (!inserirPessoa(conexao, pessoaJuridica) || !inserirPessoaJuridica(conexao, pessoaJuridica)) {
                conexao.rollback();
                return false;
            }

            conexao.commit();
            return true;
        } catch (SQLException e) {
            LOGGER.log(Level.SEVERE, "Erro ao incluir a pessoa jurídica", e);
            if (conexao != null) {
                try {
                    conexao.rollback();
                } catch (SQLException ex) {
                    LOGGER.log(Level.SEVERE, "Erro ao realizar rollback", ex);
                }
            }
            return false;
        } finally {
            if (conexao != null) {
                try {
                    conexao.close();
                } catch (SQLException e) {
                    LOGGER.log(Level.SEVERE, "Erro ao fechar conexão", e);
                }
            }
        }
    }

    private boolean inserirPessoa(Connection conexao, PessoaJuridica pessoaJuridica) throws
SQLException {
```

```
      String sqlPessoa = "INSERT INTO dbo.Pessoas (IDPessoa, NomePessoa, Telefone, Email, Logradouro,
Cidade, Estado) VALUES (?, ?, ?, ?, ?, ?, ?)";
      try (PreparedStatement preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa)) {
         preparedPessoa.setInt(1, pessoaJuridica.getId());
         preparedPessoa.setString(2, pessoaJuridica.getNome());
         preparedPessoa.setString(3, pessoaJuridica.getTelefone());
         preparedPessoa.setString(4, pessoaJuridica.getEmail());
         preparedPessoa.setString(5, pessoaJuridica.getRua());
         preparedPessoa.setString(6, pessoaJuridica.getCidade());
         preparedPessoa.setString(7, pessoaJuridica.getEstado());

         return preparedPessoa.executeUpdate() > 0;
      }
   }

   private boolean inserirPessoaJuridica(Connection conexao, PessoaJuridica pessoaJuridica) throws
SQLException {
      String sqlPessoaJuridica = "INSERT INTO dbo.PessoasJuridicas (IDPessoa, CNPJ) VALUES (?, ?)";
      try (PreparedStatement preparedPessoaJuridica = ConectorBD.getPrepared(conexao,
sqlPessoaJuridica)) {
         preparedPessoaJuridica.setInt(1, pessoaJuridica.getId());
         preparedPessoaJuridica.setString(2, pessoaJuridica.getCnpj());

         return preparedPessoaJuridica.executeUpdate() > 0;
      }
   }

   public boolean alterar(PessoaJuridica pessoaJuridica) {
      Connection conexao = null;
      try {
         conexao = ConectorBD.getConnection();
         if (conexao == null) {
            return false;
         }

         conexao.setAutoCommit(false);

         if (!alterarPessoa(conexao, pessoaJuridica) || !alterarPessoaJuridica(conexao, pessoaJuridica)) {
            conexao.rollback();
            return false;
         }

         conexao.commit();
         return true;
      } catch (SQLException e) {
         LOGGER.log(Level.SEVERE, "Erro ao alterar a pessoa jurídica", e);
         if (conexao != null) {
            try {
               conexao.rollback();
            } catch (SQLException ex) {
               LOGGER.log(Level.SEVERE, "Erro ao realizar rollback", ex);
            }
         }
         return false;
      } finally {
         if (conexao != null) {
```

```
            try {
                conexao.close();
            } catch (SQLException e) {
                LOGGER.log(Level.SEVERE, "Erro ao fechar conexão", e);
            }
        }
    }
}

    private boolean alterarPessoa(Connection conexao, PessoaJuridica pessoaJuridica) throws
SQLException {
        String sqlPessoa = "UPDATE dbo.Pessoas SET NomePessoa = ?, Telefone = ?, Email = ?, Logradouro
= ?, Cidade = ?, Estado = ? WHERE IDPessoa = ?";
        try (PreparedStatement preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa)) {
            preparedPessoa.setString(1, pessoaJuridica.getNome());
            preparedPessoa.setString(2, pessoaJuridica.getTelefone());
            preparedPessoa.setString(3, pessoaJuridica.getEmail());
            preparedPessoa.setString(4, pessoaJuridica.getRua());
            preparedPessoa.setString(5, pessoaJuridica.getCidade());
            preparedPessoa.setString(6, pessoaJuridica.getEstado());
            preparedPessoa.setInt(7, pessoaJuridica.getId());

            return preparedPessoa.executeUpdate() > 0;
        }
    }

    private boolean alterarPessoaJuridica(Connection conexao, PessoaJuridica pessoaJuridica) throws
SQLException {
        String sqlPessoaJuridica = "UPDATE dbo.PessoasJuridicas SET CNPJ = ? WHERE IDPessoa = ?";
        try (PreparedStatement preparedPessoaJuridica = ConectorBD.getPrepared(conexao,
sqlPessoaJuridica)) {
            preparedPessoaJuridica.setString(1, pessoaJuridica.getCnpj());
            preparedPessoaJuridica.setInt(2, pessoaJuridica.getId());

            return preparedPessoaJuridica.executeUpdate() > 0;
        }
    }

    public boolean excluir(int id) {
        Connection conexao = null;
        try {
            conexao = ConectorBD.getConnection();
            if (conexao == null) {
                return false;
            }

            conexao.setAutoCommit(false);

            if (!excluirPessoaJuridica(conexao, id) || !excluirPessoa(conexao, id)) {
                conexao.rollback();
                return false;
            }

            conexao.commit();
            return true;
        } catch (SQLException e) {
```

```
            LOGGER.log(Level.SEVERE, "Erro ao excluir a pessoa jurídica: " + id, e);
            if (conexao != null) {
                try {
                    conexao.rollback();
                } catch (SQLException ex) {
                    LOGGER.log(Level.SEVERE, "Erro ao realizar rollback", ex);
                }
            }
            return false;
        } finally {
            if (conexao != null) {
                try {
                    conexao.close();
                } catch (SQLException e) {
                    LOGGER.log(Level.SEVERE, "Erro ao fechar conexão", e);
                }
            }
        }
    }

    private boolean excluirPessoaJuridica(Connection conexao, int id) throws SQLException {
        String sqlPessoaJuridica = "DELETE FROM dbo.PessoasJuridicas WHERE IDPessoa = ?";
        try (PreparedStatement preparedPessoaJuridica = ConectorBD.getPrepared(conexao,
sqlPessoaJuridica)) {
            preparedPessoaJuridica.setInt(1, id);
            return preparedPessoaJuridica.executeUpdate() > 0;
        }
    }

    private boolean excluirPessoa(Connection conexao, int id) throws SQLException {
        String sqlPessoa = "DELETE FROM dbo.Pessoas WHERE IDPessoa = ?";
        try (PreparedStatement preparedPessoa = ConectorBD.getPrepared(conexao, sqlPessoa)) {
            preparedPessoa.setInt(1, id);
            return preparedPessoa.executeUpdate() > 0;
        }
    }

    private PessoaJuridica criaPessoaJuridica(ResultSet resultSet) throws SQLException {
        PessoaJuridica pessoaJuridica = new PessoaJuridica();
        pessoaJuridica.setId(resultSet.getInt("IDPessoa"));
        pessoaJuridica.setNome(resultSet.getString("NomePessoa"));
        pessoaJuridica.setTelefone(resultSet.getString("Telefone"));
        pessoaJuridica.setEmail(resultSet.getString("Email"));
        pessoaJuridica.setRua(resultSet.getString("Logradouro"));
        pessoaJuridica.setCidade(resultSet.getString("Cidade"));
        pessoaJuridica.setEstado(resultSet.getString("Estado"));
        pessoaJuridica.setCnpj(resultSet.getString("CNPJ"));
        return pessoaJuridica;
    }
}
```

ConectorBD.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastrobd.model.util;

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

/**
 *
 * @Francinaldo
 */
public class ConectorBD {
    private static final String DRIVER = "com.microsoft.sqlserver.jdbc.SQLServerDriver";
    private static final String URL =
"jdbc:sqlserver://localhost\\\\SQLEXPRESS:1433;databaseName=loja;user=app_master;passwo
rd=sua_senha;encrypt=true;trustServerCertificate=true;";
    private static final String USER = "app_master";
    private static final String PASSWORD = "123456789";

    public static Connection getConnection() {
        try {
            Class.forName(DRIVER).newInstance();
            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

```
// Dentro da sua classe SequenceManager.java
package cadastrobd.model.util; // Adicione a declaração do pacote

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

public class SequenceManager {

    public static Integer getValue(String sequenceName) {
        if (!sequenceName.equalsIgnoreCase("CodigoPessoa")) {
            System.out.println("Sequence não suportada: " + sequenceName);
            return -1; // Ou lance uma exceção
        }

        Connection conexao = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        int nextId = -1;

        try {
            conexao = ConectorBD.getConnection();
            if (conexao == null) {
                System.out.println("Erro ao obter conexão com o banco de dados.");
                return -1;
            }

            // Consulta para obter o próximo valor e incrementá-lo atomicamente
            String sql = "UPDATE dbo.CodigoPessoa " +
                         "SET ProximoCodigo = ProximoCodigo + 1 " +
                         "OUTPUT inserted.ProximoCodigo;";

            preparedStatement = conexao.prepareStatement(sql);
            resultSet = preparedStatement.executeQuery();

            if (resultSet.next()) {
                nextId = resultSet.getInt("ProximoCodigo");
            }

        } catch (SQLException e) {
            System.out.println("Erro ao obter o próximo valor da sequência CodigoPessoa: " +
e.getMessage());
        } finally {
            ConectorBD.close(resultSet);
            ConectorBD.close(preparedStatement);
            ConectorBD.close(conexao);
        }
        return nextId;
    }
}
```

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

Os componentes de middleware desempenham um papel importante no desenvolvimento de sistemas distribuídos e na integração de diferentes tecnologias. O JDBC (Java Database Connectivity) é um exemplo específico de middleware utilizado para realizar a conexão de aplicativos Java a bancos de dados diversos.

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

Ambos Statement e PreparedStatement são interfaces fornecidas pelo JDBC (Java Database Connectivity) para executar consultas SQL em um banco de dados a partir de um aplicativo Java. No entanto, há diferenças significativas entre eles, especialmente em termos de segurança, desempenho e conveniência.

Segurança: PreparedStatement oferece maior segurança contra ataques de injeção de SQL. Isso ocorre porque os parâmetros em um PreparedStatement são tratados como valores separados da instrução SQL em si, enquanto em um Statement os valores são diretamente incorporados na consulta. Assim, o uso de PreparedStatement ajuda a prevenir a inserção maliciosa de código SQL.

Desempenho: PreparedStatement geralmente é mais eficiente em termos de desempenho, especialmente quando a mesma instrução SQL é executada repetidamente com diferentes conjuntos de parâmetros. Isso ocorre porque o banco de dados pode pré-compilar e otimizar a consulta uma vez e reutilizá-la com diferentes valores de parâmetro, em vez de analisar e otimizar a consulta cada vez que é executada, como acontece com um Statement.

Cache de consulta: Muitos bancos de dados mantêm um cache de consultas preparadas, o que pode melhorar ainda mais o desempenho do PreparedStatement em comparação com o Statement. Consultas preparadas frequentemente executadas podem ser armazenadas em cache pelo banco de dados, reduzindo o tempo necessário para analisar e otimizar a consulta.

Conveniência: PreparedStatement pode ser mais conveniente de usar em alguns casos, especialmente quando se trata de inserir ou atualizar múltiplos registros com valores dinâmicos. A sintaxe do PreparedStatement permite a inclusão de marcadores de posição (placeholders) na consulta, facilitando a substituição de valores de parâmetro em tempo de execução.

Embora tanto Statement quanto PreparedStatement possam ser usados para executar consultas SQL em um banco de dados via JDBC, PreparedStatement é geralmente preferido devido à sua maior segurança, melhor desempenho e, em muitos casos, maior conveniência.

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

O padrão DAO (Data Access Object) é uma abordagem comum para separar a lógica de acesso a dados do restante do código de negócios em um aplicativo. Ele encapsula a lógica para acessar dados do banco de dados em uma camada separada, proporcionando vários benefícios que melhoram a manutenibilidade do software

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

Quando se trabalha com um modelo estritamente relacional em um banco de dados, a herança é refletida de forma diferente do que em linguagens de programação orientadas a objetos, onde a herança é uma parte fundamental da modelagem.

Em um modelo estritamente relacional, a herança é frequentemente representada por meio de técnicas como:

Tabelas separadas para cada subtipo: Cada subtipo de uma hierarquia de herança é representado por uma tabela separada no banco de dados. Essa abordagem é conhecida como modelagem de "tabelas de subtipo". Cada tabela contém apenas os atributos específicos do subtipo, além dos atributos herdados da tabela pai.

Tabelas de junção (join tables): Quando a herança é modelada usando uma tabela de junção, uma tabela adicional é criada para cada relação de herança. Essa tabela de junção contém chaves estrangeiras que fazem referência tanto à tabela pai quanto à tabela filho, estabelecendo a relação entre elas.

2º Procedimento | Alimentando a Base

CadastroBD.java

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package cadastrobd;

import cadastrobd.model.PessoasFisicas;
import cadastrobd.model.PessoasFisicasDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.util.Scanner;

/**
 *
 * @Francinaldo
 */
public class CadastroBD {
    private static final Scanner sc = new Scanner(System.in);
    private static final PessoasFisicasDAO pfDAO = new PessoasFisicasDAO();
    private static final PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();


    public static void main(String[] args) {
        int opcao = -1;
        while (opcao != 0) {
            System.out.println("===== CADASTRO BD =====");
            System.out.println("Selecione uma opcao:");
            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 execucao");

            opcao = Integer.parseInt(sc.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;
            case 0:
                System.out.println("Finalizando a execucao.");
                break;
            default:
                System.out.println("Opcao invalida.");
        }

        System.out.println();
    }
  }

  private static String lerTipoDePessoa() {
    System.out.println("Escolha um opcao:\n\tPara Pessoa Fisica digite F\n\tPara Pessoa Juridica digite
J");
    String tipo = sc.nextLine();

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

    if (tipo.equalsIgnoreCase("F") || tipo.equalsIgnoreCase("J")) {
      return tipo;
    } else {
      System.out.println("Opcao invalida, tente novamente.");
      return lerTipoDePessoa();
    }
  }

  private static PessoasFisicas definirDadosPessoasFisicas(PessoasFisicas PessoasFisicas) {
    try {
      System.out.println("Digite o nome: ");
      PessoasFisicas.setNome(sc.nextLine());
      System.out.println("Digite o cpf: ");
      PessoasFisicas.setCpf(sc.nextLine());
      System.out.println("Digite o telefone: ");
      PessoasFisicas.setTelefone(sc.nextLine());
      System.out.println("Digite o email: ");
      PessoasFisicas.setEmail(sc.nextLine());
      System.out.println("Digite a rua: ");
      PessoasFisicas.setRua(sc.nextLine());
      System.out.println("Digite a cidade: ");
      PessoasFisicas.setCidade(sc.nextLine());
      System.out.println("Digite o estado: ");
      PessoasFisicas.setEstado(sc.nextLine());
      return PessoasFisicas;
    } catch (Exception e) {
      System.out.println("Nao foi possivel cadastrar os dados da Pessoa física:");
      e.printStackTrace();
      System.out.println("Tente novamente.");
      return null;
    }
  }
```

```
private static PessoaJuridica definirDadosPessoaJuridica(PessoaJuridica pessoaJuridica) {
   try {
      System.out.println("Digite o nome: ");
      pessoaJuridica.setNome(sc.nextLine());
      System.out.println("Digite o CNPJ: ");
      pessoaJuridica.setCnpj(sc.nextLine());
      System.out.println("Digite o telefone: ");
      pessoaJuridica.setTelefone(sc.nextLine());
      System.out.println("Digite o email: ");
      pessoaJuridica.setEmail(sc.nextLine());
      System.out.println("Digite a rua: ");
      pessoaJuridica.setRua(sc.nextLine());
      System.out.println("Digite a cidade: ");
      pessoaJuridica.setCidade(sc.nextLine());
      System.out.println("Digite o estado: ");
      pessoaJuridica.setEstado(sc.nextLine());
      return pessoaJuridica;
   } catch (Exception e) {
      System.out.println("Nao foi possivel cadastrar os dados da Pessoa Jurídica:");
      e.printStackTrace();
      System.out.println("Tente novamente.");
      return null;
   }
}

private static void inserirPessoa() {
   String tipo = lerTipoDePessoa();

   if (tipo.equalsIgnoreCase("F")) {
      PessoasFisicas PessoasFisicas = definirDadosPessoasFisicas(new PessoasFisicas());
      if (PessoasFisicas == null) {
         System.out.println("Erro ao cadastrar Pessoa Física.");
         return;
      }
      if (!pfDAO.incluir(PessoasFisicas)) {
         System.out.println("Erro ao cadastrar Pessoa Física.");
         return;
      }
      System.out.println("Pessoa Física cadastrada com sucesso.");
      return;
   }
   if (tipo.equalsIgnoreCase("J")) {
      PessoaJuridica pessoaJuridica = definirDadosPessoaJuridica(new PessoaJuridica());
      if (pessoaJuridica == null) {
         System.out.println("Falha ao cadastrar Pessoa Jurídica.");
         return;
      }
      if (!pjDAO.incluir(pessoaJuridica)) {
         System.out.println("Falha ao cadastrar Pessoa Jurídica.");
         return;
      }
      System.out.println("Pessoa Jurídica cadastrar com sucesso.");
   }
}

private static void alterarPessoa() {
```

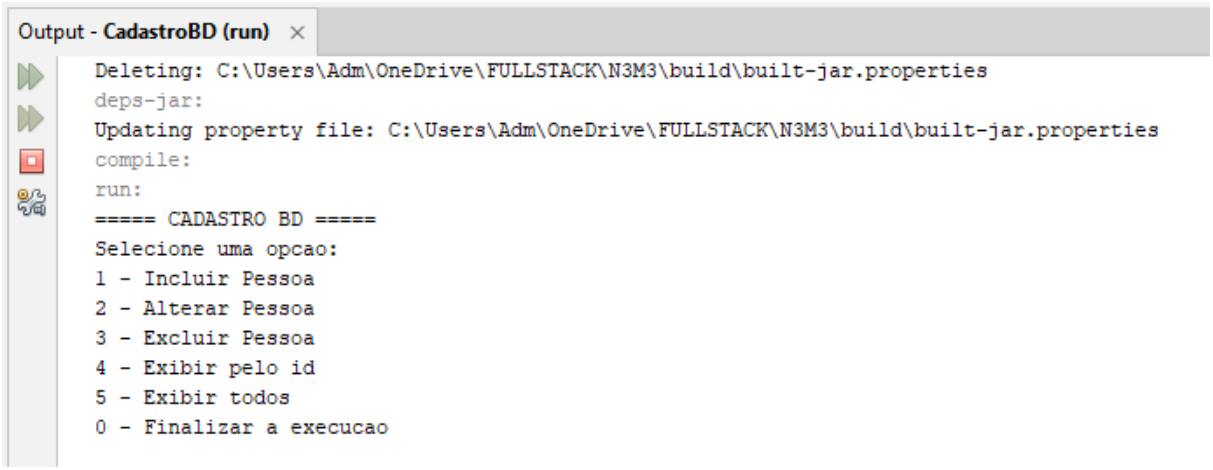
```
String tipo = lerTipoDePessoa();

if (tipo.equalsIgnoreCase("F")) {
   System.out.println("Digite o id da Pessoa Física que deseja alterar: ");
   try {
      int idPessoasFisicas = Integer.parseInt(sc.nextLine());
      PessoasFisicas PessoasFisicas = pfDAO.getPessoa(idPessoasFisicas);

      if (PessoasFisicas == null) {
         System.out.println("Pessoa Física não encontrada.");
         return;
      }
      PessoasFisicas.exibir();
      PessoasFisicas = definirDadosPessoasFisicas(PessoasFisicas); // Aqui o ID já existe

      if (PessoasFisicas == null) {
         System.out.println("Falha ao alterar Pessoa Física.");
         return;
      }

      if (!pfDAO.alterar(PessoasFisicas)) {
         System.out.println("Falha ao alterar Pessoa Física.");
         return;
      }
      System.out.println("Pessoa Física alterada com sucesso.");
   } catch (NumberFormatException e) {
      System.out.println("ID inválido. Digite um número inteiro.");
   }
   return;
}

if (tipo.equalsIgnoreCase("J")) {
   System.out.println("Digite o id da Pessoa Jurídica que deseja alterar: ");
   try {
      int idPessoaJuridica = Integer.parseInt(sc.nextLine());
      PessoaJuridica pessoaJuridica = pjDAO.getPessoa(idPessoaJuridica);

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

      pessoaJuridica.exibir();
      pessoaJuridica = definirDadosPessoaJuridica(pessoaJuridica); // Aqui o ID já existe

      if (pessoaJuridica == null) {
         System.out.println("Falha ao alterar Pessoa Jurídica.");
         return;
      }

      if (!pjDAO.alterar(pessoaJuridica)) {
         System.out.println("Falha ao alterar Pessoa Jurídica.");
         return;
      }

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

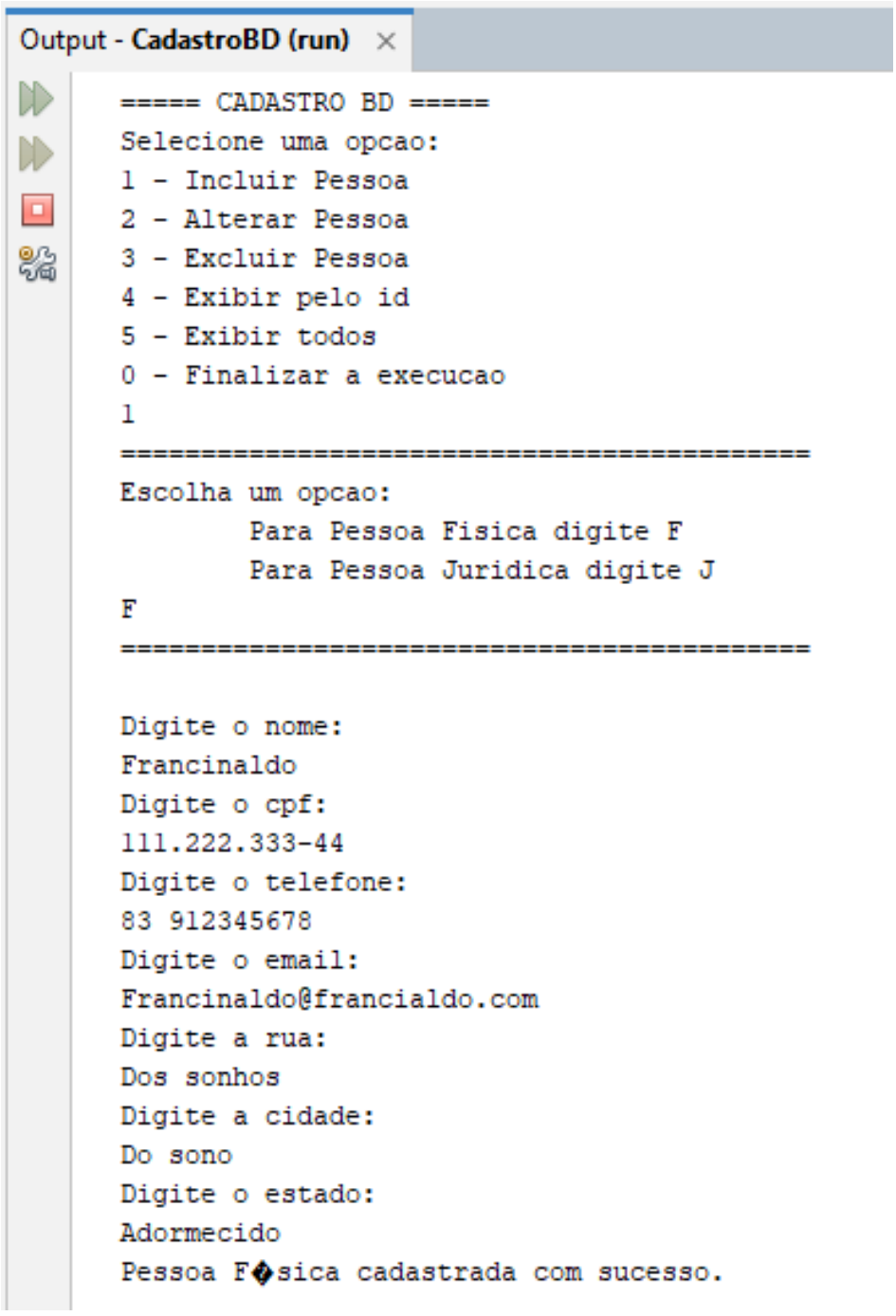
```
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Digite um número inteiro.");
        }
    }
}

private static void excluirPessoa() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja excluir: ");
        try {
            int idPessoasFisicas = Integer.parseInt(sc.nextLine());
            PessoasFisicas PessoasFisicas = pfDAO.getPessoa(idPessoasFisicas);

            if (PessoasFisicas == null) {
                System.out.println("Pessoa Física não encontrada.");
                return;
            }

            if (!pfDAO.excluir(idPessoasFisicas)) {
                System.out.println("Falha ao excluir Pessoa Física.");
                return;
            }

            System.out.println("Pessoa Física excluída com sucesso.");
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Digite um número inteiro.");
        }
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        System.out.println("Digite o id da Pessoa Jurídica que deseja excluir: ");
        try {
            int idPessoaJuridica = Integer.parseInt(sc.nextLine());
            PessoaJuridica pessoaJuridica = pjDAO.getPessoa(idPessoaJuridica);

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

            if (!pjDAO.excluir(idPessoaJuridica)) {
                System.out.println("Falha ao excluir Pessoa Jurídica.");
                return;
            }

            System.out.println("Pessoa Jurídica excluída com sucesso.");
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Digite um número inteiro.");
        }
    }
}

private static void exibirPessoaPeloId() {
```

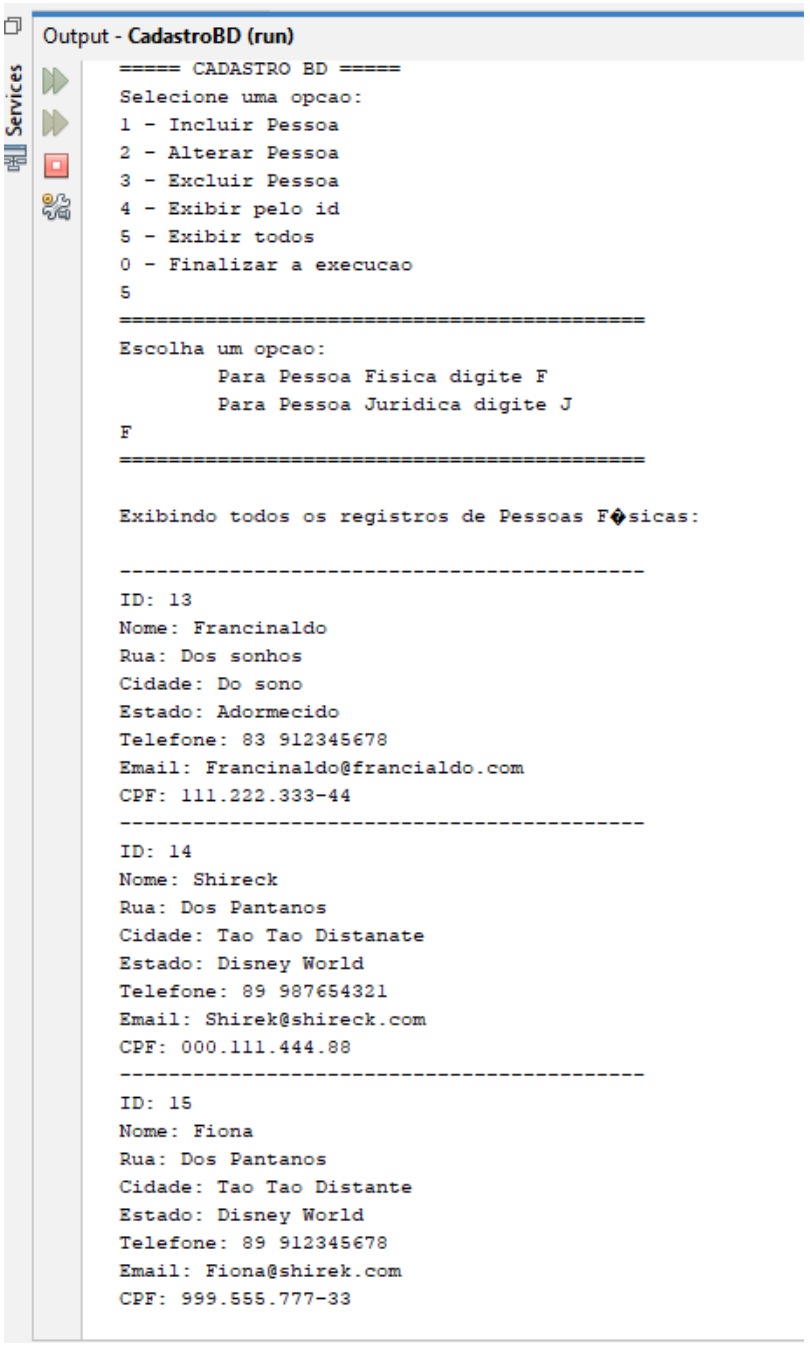
```
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        System.out.println("Digite o id da Pessoa Física que deseja exibir: ");
        try {
            int idPessoasFisicas = Integer.parseInt(sc.nextLine());
            PessoasFisicas PessoasFisicas = pfDAO.getPessoa(idPessoasFisicas);

            if (PessoasFisicas == null) {
                System.out.println("Pessoa Física não encontrada.");
                return;
            }
            PessoasFisicas.exibir();
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Digite um número inteiro.");
        }
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        System.out.println("Digite o id da Pessoa Jurídica que deseja exibir: ");
        try {
            int idPessoaJuridica = Integer.parseInt(sc.nextLine());
            PessoaJuridica pessoaJuridica = pjDAO.getPessoa(idPessoaJuridica);

            if (pessoaJuridica == null) {
                System.out.println("Pessoa Jurídica não encontrada.");
                return;
            }
            pessoaJuridica.exibir();
        } catch (NumberFormatException e) {
            System.out.println("ID inválido. Digite um número inteiro.");
        }
    }
}

private static void exibirTodasAsPessoas() {
    String tipo = lerTipoDePessoa();

    if (tipo.equalsIgnoreCase("F")) {
        java.util.List<PessoasFisicas> pessoasFisicas = pfDAO.getPessoas();
        if (pessoasFisicas == null) {
            System.out.println("Não existem Pessoas Físicas cadastradas.");
            return;
        }

        System.out.println("Exibindo todos os registros de Pessoas Físicas:\n");
        for (PessoasFisicas PessoasFisicas : pessoasFisicas) {
            System.out.println("----------------------------------------");
            PessoasFisicas.exibir();
        }
        return;
    }

    if (tipo.equalsIgnoreCase("J")) {
        java.util.List<PessoaJuridica> pessoasJuridicas = pjDAO.getPessoas();
```

```
        if (pessoasJuridicas == null) {
            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();
        }
    }
  }
}
```

Resultados:


Output - CadastroBD (run)
Deleting: C:\Users\Adm\OneDrive\FULLSTACK\N3M3\build\built-jar.properties
deps-jar:
Updating property file: C:\Users\Adm\OneDrive\FULLSTACK\N3M3\build\built-jar.properties
compile:
run:
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao

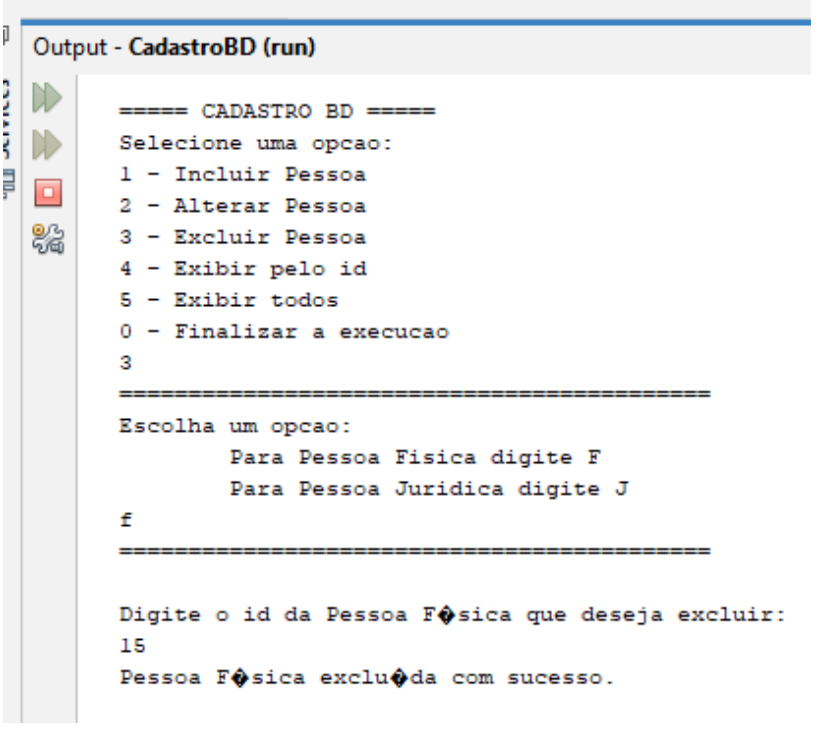

Output - CadastroBD (run)
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao
1
==========
Escolha um opcao:
Para Pessoa Fisica digite F
Para Pessoa Juridica digite J
F
==========
Digite o nome:
Francinaldo
Digite o cpf:
111.222.333-44
Digite o telefone:
83 912345678
Digite o email:
Francinaldo@francialdo.com
Digite a rua:
Dos sonhos
Digite a cidade:
Do sono
Digite o estado:
Adormecido
Pessoa F◆sica cadastrada com sucesso.

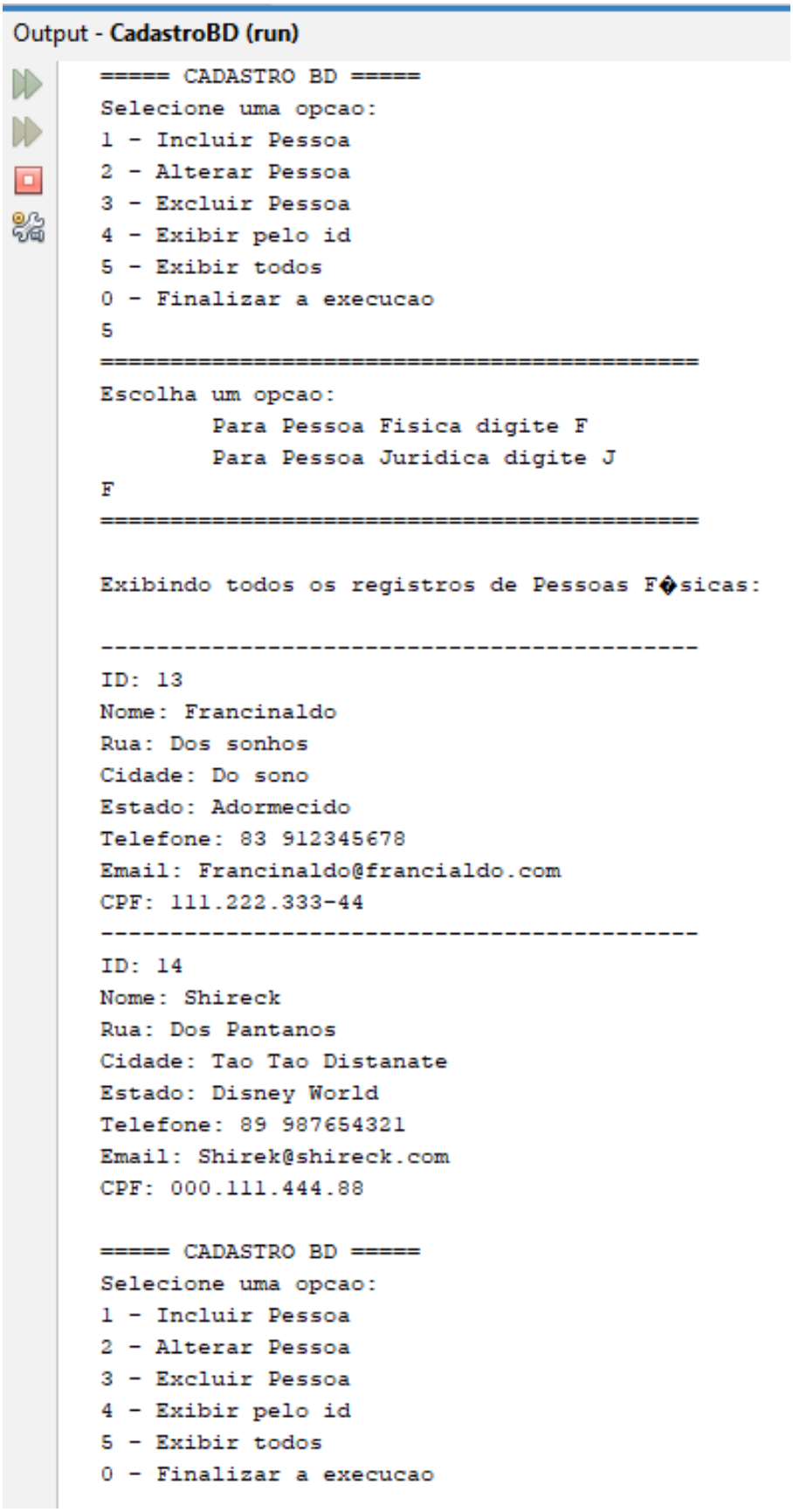

Output - CadastroBD (run)
Services
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao
5
==========
Escolha um opcao:
Para Pessoa Fisica digite F
Para Pessoa Juridica digite J
F
==========
Exibindo todos os registros de Pessoas F?sicas:

ID: 13
Nome: Francinaldo
Rua: Dos sonhos
Cidade: Do sono
Estado: Adormecido
Telefone: 83 912345678
Email: Francinaldo@francialdo.com
CPF: 111.222.333-44

ID: 14
Nome: Shireck
Rua: Dos Pantanos
Cidade: Tao Tao Distanate
Estado: Disney World
Telefone: 89 987654321
Email: Shirek@shireck.com
CPF: 000.111.444.88

ID: 15
Nome: Fiona
Rua: Dos Pantanos
Cidade: Tao Tao Distante
Estado: Disney World
Telefone: 89 912345678
Email: Fiona@shirek.com
CPF: 999.555.777-33


Output - CadastroBD (run)
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao
3
==========
Escolha um opcao:
Para Pessoa Fisica digite F
Para Pessoa Juridica digite J
f
==========
Digite o id da Pessoa F?sica que deseja excluir:
15
Pessoa F?sica exclu?da com sucesso.


Output - CadastroBD (run)
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao
5
==========
Escolha um opcao:
Para Pessoa Fisica digite F
Para Pessoa Juridica digite J
F
==========
Exibindo todos os registros de Pessoas F?sicas:

ID: 13
Nome: Francinaldo
Rua: Dos sonhos
Cidade: Do sono
Estado: Adormecido
Telefone: 83 912345678
Email: Francinaldo@francialdo.com
CPF: 111.222.333-44

ID: 14
Nome: Shireck
Rua: Dos Pantanos
Cidade: Tao Tao Distanate
Estado: Disney World
Telefone: 89 987654321
Email: Shirek@shireck.com
CPF: 000.111.444.88
===== CADASTRO BD =====
Selecione uma opcao:
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Exibir pelo id
5 - Exibir todos
0 - Finalizar a execucao

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 são duas formas de armazenar dados, cada uma com suas características e finalidades específicas. Aqui estão algumas diferenças entre elas:

Formato de armazenamento:

- **Persistência em Arquivo**: Os dados são armazenados em arquivos no sistema de arquivos do computador. Podem ser em formatos simples, como texto puro, JSON, XML, binário, etc.
- **Persistência em Banco de Dados**: Os dados são armazenados em tabelas, geralmente seguindo um modelo de dados relacionais (SQL) ou não-relacionais (NoSQL).

Capacidade de consulta:

- **Persistência em Arquivo**: Geralmente, é mais difícil e menos eficiente realizar consultas complexas nos dados armazenados em arquivos. Dependendo do formato de armazenamento, pode ser necessário processar todo o arquivo para encontrar as informações desejadas.
- **Persistência em Banco de Dados**: Os bancos de dados oferecem recursos poderosos para consultas, como SQL ou linguagens de consulta específicas para NoSQL. Isso permite consultas eficientes e flexíveis, incluindo filtragem, ordenação, junção de tabelas, entre outros.

Concorrência e transações:

- **Persistência em Arquivo**: Pode ser complicado gerenciar a concorrência e garantir a consistência dos dados em ambientes onde múltiplos processos ou threads estão acessando e modificando o mesmo arquivo simultaneamente.
- **Persistência em Banco de Dados**: Os bancos de dados são projetados para lidar com transações concorrentes de forma segura, garantindo consistência, isolamento, atomicidade e durabilidade (propriedades ACID).

Escalabilidade:

- **Persistência em Arquivo**: A escalabilidade pode ser limitada, especialmente em ambientes onde muitos processos precisam acessar e modificar os mesmos arquivos simultaneamente.
- **Persistência em Banco de Dados**: Os bancos de dados são projetados para serem escaláveis, permitindo distribuir os dados em vários servidores e lidar com grandes volumes de dados e tráfego simultâneo.

Manutenção e Backup:

Persistência em Arquivo: A manutenção e o backup dos dados podem exigir abordagens mais manuais e customizadas, dependendo do volume e da importância dos dados.

Persistência em Banco de Dados: Os sistemas de gerenciamento de banco de dados geralmente oferecem recursos para backup, restauração e manutenção automatizados dos dados.

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 no Java simplificou a impressão dos valores contidos nas entidades principalmente através do uso de expressões lambda e do novo método `forEach()` introduzido nas versões mais recentes do Java, como Java 8 em diante.

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

Em Java, o método `main` é o ponto de entrada para qualquer aplicação Java. Ele é o método que o Java procura quando você inicia sua aplicação. Quando você chama o programa Java pela linha de comando, você faz isso sem criar uma instância da classe que contém o método `main`. Por isso, o método `main` precisa ser marcado como `static`.