BackEnd sem banco não tem

Faculdade Estácio - Polo Barra

Curso : Desenvolvimento Full Stack

Disciplina : BackEnd sem banco não tem

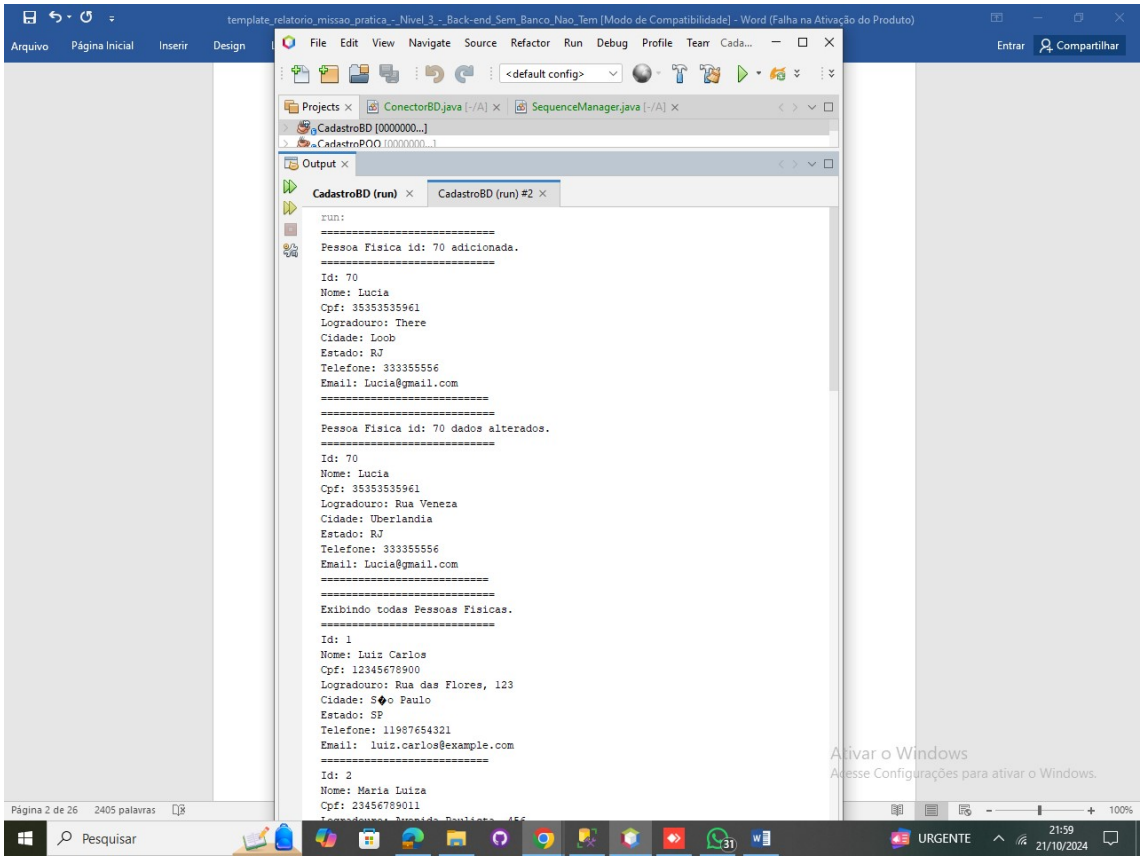
NÚMERO DA Turma : RPG0016 9001- Semestre Letivo: **3**

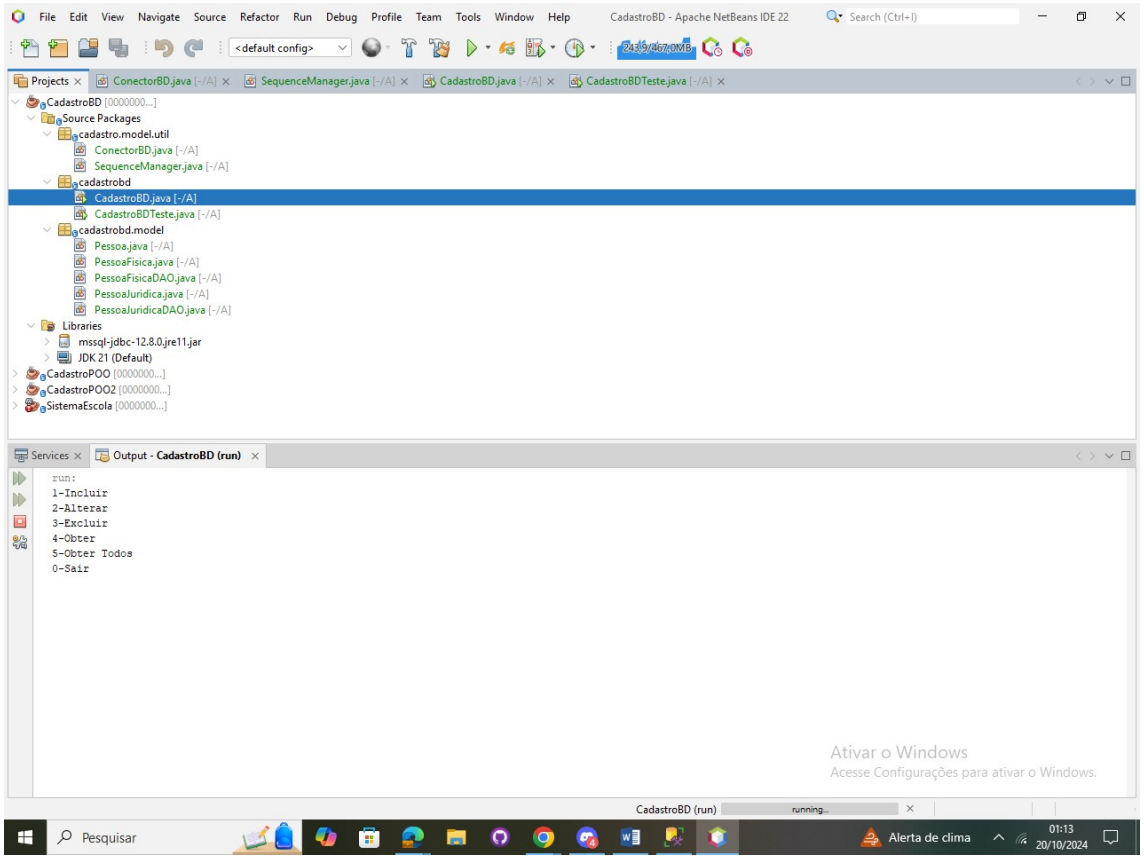
Integrante: Lucia Maria de Lima Martins

Repositório:
https://github.com/luciamartins/MissaoNivel3Mundo3.git

Objetivo 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.
6. do SQL Server na persistência de dados.


template_relatorio_missao_pratica_-_Nivel_3_-_Back-end_Sem_Banco_Nao_Tem [Modo de Compatibilidade] - Word (Falha na Ativação do Produto)
Arquivo
Página Inicial
Inserir
Design
Entrar
Compartilhar
File Edit View Navigate Source Refactor Run Debug Profile Team
<default config>
Projects
ConectorBD.java [-/A]
SequenceManager.java [-/A]
CadastroBD [0000000...]
Output
CadastroBD (run)
CadastroBD (run) #2
run:
==========
Pessoa Fisica id: 70 adicionada.
==========
Id: 70
Nome: Lucia
Cpf: 35353535961
Logradouro: There
Cidade: Loob
Estado: RJ
Telefone: 333355556
Email: Lucia@gmail.com
==========
==========
Pessoa Fisica id: 70 dados alterados.
==========
Id: 70
Nome: Lucia
Cpf: 35353535961
Logradouro: Rua Veneza
Cidade: Uberlandia
Estado: RJ
Telefone: 333355556
Email: Lucia@gmail.com
==========
==========
Exibindo todas Pessoas Fisicas.
==========
Id: 1
Nome: Luiz Carlos
Cpf: 12345678900
Logradouro: Rua das Flores, 123
Cidade: S&#65533;o Paulo
Estado: SP
Telefone: 11987654321
Email: luiz.carlos@example.com
==========
Id: 2
Nome: Maria Luiza
Cpf: 23456789011
Ativar o Windows
Acesse Configurações para ativar o Windows.
Página 2 de 26
2405 palavras
100%
Pesquisar
URGENTE
21:59
21/10/2024


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
CadastroBD - Apache NetBeans IDE 22
Search (Ctrl+I)
<default config>
Projects
ConectorBD.java [-/A]
SequenceManager.java [-/A]
CadastroBD.java [-/A]
CadastroBDTeste.java [-/A]
CadastroBD [0000000...]
Source Packages
cadastro.model.util
ConectorBD.java [-/A]
SequenceManager.java [-/A]
cadastrobd
CadastroBD.java [-/A]
CadastroBDTeste.java [-/A]
cadastrobd.model
Pessoa.java [-/A]
PessoaFisica.java [-/A]
PessoaFisicaDAO.java [-/A]
PessoaJuridica.java [-/A]
PessoaJuridicaDAO.java [-/A]
Libraries
mssql-jdbc-12.8.0.jre11.jar
JDK 21 (Default)
CadastroPOO [0000000...]
CadastroPOO2 [0000000...]
SistemaEscola [0000000...]
Services
Output - CadastroBD (run)
run:
1-Incluir
2-Alterar
3-Excluir
4-Obter
5-Obter Todos
0-Sair
Ativar o Windows
Acesse Configurações para ativar o Windows.
CadastroBD (run)
running...
Pesquisar
Alerta de clima
01:13
20/10/2024

1º Procedimento | Mapeamento Objeto-Relacional e DAO

```
package cadastrobd.model;
public class Pessoa {
    private Integer id;
    private String nome;
    private String logradouro;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;
    public Pessoa() {
    }
    public Pessoa(Integer 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 setId(int id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getNome() {
        return nome;
```

```
}
public void setLogradouro(String logradouro) {
   this.logradouro = logradouro;
}
public String getLogradouro() {
   return logradouro;
}
public void setCidade(String cidade) {
   this.cidade = cidade;
}
public String getCidade() {
   return cidade;
}
public void setEstado(String estado) {
   this.estado = estado;
}
public String getEstado() {
   return estado;
}
public void setTelefone(String telefone) {
   this.telefone = telefone;
}
public String getTelefone() {
   return telefone;
}
public void setEmail(String email) {
   this.email = email;
}
public String getEmail() {
   return email;
}
public void exibir() {
   System.out.println("\nId: " + this.getId()
         + "\nNome: " + this.getNome()
         + "\nLogradouro: " + this.getLogradouro()
         + "\nCidade: " + this.getCidade()
         + "\nEstado: " + this.getEstado()
         + "\nTelefone: " + this.getTelefone()
```

```
            + "\nEmail: " + this.getEmail());
    }
}
```

```
package cadastrobd.model;
public class PessoaFisica extends Pessoa {
    private String cpf;
    public PessoaFisica(Integer 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 void setCpf(String cpf){
        this.cpf = cpf;
    }
    public String getCpf(){
        return cpf;
    }
    @Override
    public void exibir(){
        System.out.println("Id: " + this.getId() +
                "\nNome: " + this.getNome() +
                "\nCpf: " + this.getCpf() +
                "\nLogradouro: " + this.getLogradouro() +
                "\nCidade: " + this.getCidade() +
                "\nEstado: " + this.getEstado() +
                "\nTelefone: " + this.getTelefone() +
                "\nEmail: " + this.getEmail() +
                "\n==========================");
    }
}
```

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica(Integer id, String nome, String logradouro, String cidade, String estado, String telefone, String email, String cnpj){

        super(id, nome, logradouro, cidade, estado, telefone, email);

        this.cnpj = cnpj;

    }

    public void setCnpj(String cnpj){

        this.cnpj = cnpj;

    }

    public String getCnpj(){

        return cnpj;

    }

    @Override

    public void exibir(){

        System.out.println("\nId: " + this.getId() +

                "\nNome: " + this.getNome() +

                "\nCpf: " + this.getCnpj() +

                "\nLogradouro: " + this.getLogradouro() +

                "\nCidade: " + this.getCidade() +

                "\nEstado: " + this.getEstado() +

                "\nTelefone: " + this.getTelefone() +

                "\nEmail: " + this.getEmail() +

                "\n============================");

    }

}
```

```
package cadastro.model.util;

import java.sql.Statement;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.DriverManager;

public class ConectorBD {

    public Connection getConnection() throws Exception {

        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

        return
DriverManager.getConnection("jdbc:sqlserver://localhost:55967;databaseName=loja;encrypt=true;trustSe
rverCertificate =true;",  "loja", "loja");

    }

    public PreparedStatement getPrepared(String sql) throws Exception {

        Connection con = getConnection();

        return con.prepareStatement(sql);

    }

    public ResultSet getSelect(String sql) throws Exception{

        PreparedStatement ps = getPrepared(sql);

        return ps.executeQuery();

    }

    public void close(Connection conn) throws Exception {

        if(conn != null){

            try{

                conn.close();

            }catch(SQLException e){

                e.printStackTrace();

            }

        }
```

```
    }
    public void close(Statement st) throws Exception {
        if(st != null){
            try{
                st.close();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
    }
    public void close(ResultSet rs) throws Exception {
        if(rs != null){
            try{
                rs.close();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
    }
}
```

```
package cadastro.model.util;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
public class SequenceManager {
    public int getValue(String sequence) throws SQLException, Exception{
        ResultSet rs = new ConectorBD().getSelect("SELECT NEXT VALUE FOR ".concat(sequence));
        if(rs.next()){
            return rs.getInt(1);
        }else{
            throw new SQLException("Value not achivable");
        }
    }
package cadastrobd.model;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    private final ConectorBD connector;
    public PessoaFisicaDAO(){
        connector = new ConectorBD();
    }
    public PessoaFisica getPessoa(int id) throws SQLException, Exception{
```

```
String sql = "SELECT pf.Pessoa_idPessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone, p.email, pf.cpf "
    + "FROM PessoaFisica pf "
    + "INNER JOIN Pessoa p ON pf.Pessoa_idPessoa = p.idPessoa "
    + "WHERE pf.Pessoa_idPessoa = ?";
PessoaFisica pessoa = null;
Connection con = null;
PreparedStatement stmt = null;
ResultSet rs = null;
try{
    con = connector.getConnection();
    stmt = con.prepareStatement(sql);
    stmt.setInt(1, id);
    rs = stmt.executeQuery();
    if(rs.next()){
        pessoa = new PessoaFisica(
        rs.getInt("Pessoa_idPessoa"),
        rs.getString("nome"),
        rs.getString("logradouro"),
        rs.getString("cidade"),
        rs.getString("estado"),
        rs.getString("telefone"),
        rs.getString("email"),
        rs.getString("cpf")
        );
        return pessoa;
    }else{
        return null;
    }
```

```
        }finally{
            connector.close(rs);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public List<PessoaFisica> getPessoas() throws SQLException, Exception{
        String sql = "SELECT pf.Pessoa_idPessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone, p.email, pf.cpf "
            + "FROM Pessoa p "
            + "INNER JOIN PessoaFisica pf ON p.idPessoa = pf.Pessoa_idPessoa;";
        List<PessoaFisica> lista = new ArrayList<>();
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        try{
            con = connector.getConnection();
            stmt = con.prepareStatement(sql);
            rs = stmt.executeQuery();
            while(rs.next()){
                lista.add(new PessoaFisica(
                rs.getInt("Pessoa_idPessoa"),
                rs.getString("nome"),
                rs.getString("logradouro"),
                rs.getString("cidade"),
                rs.getString("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("cpf")
```

```
            ));
        }
        return lista;
    }finally{
        connector.close(rs);
        connector.close(stmt);
        connector.close(con);
    }
}
  public void incluir(PessoaFisica pessoa) throws SQLException, Exception{
    String sqlPessoa = "INSERT INTO Pessoa(idPessoa,nome,logradouro,cidade,estado,telefone,email)
VALUES(?,?,?,?,?,?,?);";
    String sqlPessoaFisica = "INSERT INTO PessoaFisica(Pessoa_idPessoa,cpf)VALUES(?,?);";
    Connection con = null;
    PreparedStatement stmt = null;
    PreparedStatement stmtPf = null;
    try{
        con = connector.getConnection();
        con.setAutoCommit(false);
        SequenceManager sequenceManager = new SequenceManager();
        int idNovaPessoa = sequenceManager.getValue("orderPessoa");
        stmt = con.prepareStatement(sqlPessoa);
        stmt.setInt(1, idNovaPessoa);
        stmt.setString(2, pessoa.getNome());
        stmt.setString(3, pessoa.getLogradouro());
        stmt.setString(4, pessoa.getCidade());
        stmt.setString(5, pessoa.getEstado());
        stmt.setString(6, pessoa.getTelefone());
        stmt.setString(7, pessoa.getEmail());
```

```
            stmt.executeUpdate();
            pessoa.setId(idNovaPessoa);
            stmtPf = con.prepareStatement(sqlPessoaFisica);
            stmtPf.setInt(1, idNovaPessoa);
            stmtPf.setString(2, pessoa.getCpf());
            stmtPf.executeUpdate();
            con.commit();
        }catch(SQLException e){
            System.err.println("Adf: " + e);
        }finally{
            connector.close(stmtPf);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public void alterar(PessoaFisica pessoa) throws SQLException, Exception{
        String sqlPessoa = "UPDATE PESSOA SET nome=?,logradouro=?,cidade=?,estado=?,telefone=?,email=? WHERE idPessoa = ?";
        String sqlPessoaFisica = "UPDATE PESSOAFISICA SET cpf=? WHERE Pessoa_idPessoa = ?;";
        Connection con = null;
        PreparedStatement stmt = null;
        PreparedStatement stmtPf = null;
        try{
            con = connector.getConnection();
            stmt = con.prepareStatement(sqlPessoa);
            stmtPf = con.prepareStatement(sqlPessoaFisica);
            stmt.setString(1, pessoa.getNome());
            stmt.setString(2, pessoa.getLogradouro());
            stmt.setString(3, pessoa.getCidade());
```

```
            stmt.setString(4, pessoa.getEstado());
            stmt.setString(5, pessoa.getTelefone());
            stmt.setString(6, pessoa.getEmail());
            stmt.setInt(7, pessoa.getId());
            stmt.executeUpdate();
            stmtPf.setString(1, pessoa.getCpf());
            stmtPf.setInt(2, pessoa.getId());
            stmtPf.executeUpdate();
        }catch(Exception e){
            System.err.println("Alf: " + e);
        }finally{
            connector.close(stmtPf);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public void excluir(int chave) throws SQLException, Exception{
        String sqlPessoa = "DELETE FROM PESSOA WHERE idPessoa = ?;";
        String sqlPessoaFisica = "DELETE FROM PESSOAFISICA WHERE Pessoa_idPessoa = ?;";
        Connection con = null;
        PreparedStatement stmt = null;
        PreparedStatement stmtPf = null;
        try{
            con = connector.getConnection();
            stmtPf = con.prepareStatement(sqlPessoaFisica);
            stmtPf.setInt(1, chave);
            stmtPf.executeUpdate();
            stmt = con.prepareStatement(sqlPessoa);
            stmt.setInt(1, chave);
```

```
            stmt.executeUpdate();
        }catch(Exception e){
            System.err.println("Exf: " + e);
        }finally{
            connector.close(stmt);
            connector.close(stmtPf);
            connector.close(con);
        }
    }
}
package cadastrobd.model;
import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
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 final ConectorBD connector;
    public PessoaJuridicaDAO(){
        connector = new ConectorBD();
    }
    public PessoaJuridica getPessoa(int id) throws SQLException, Exception{
        String sql = "SELECT pj.Pessoa_idPessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone,
p.email, pj.cnpj "
            + "FROM PessoaJuridica pj "
            + "INNER JOIN Pessoa p ON pj.Pessoa_idPessoa = p.idPessoa "
```

```
    + "WHERE pj.Pessoa_idPessoa = ?";
PessoaJuridica pessoa = null;
Connection con = null;
PreparedStatement stmt = null;
ResultSet rs = null;
try{
    con = connector.getConnection();
    stmt = con.prepareStatement(sql);
    stmt.setInt(1, id);
    rs = stmt.executeQuery();
    if(rs.next()){
        pessoa = new PessoaJuridica(
        rs.getInt("Pessoa_idPessoa"),
        rs.getString("nome"),
        rs.getString("logradouro"),
        rs.getString("cidade"),
        rs.getString("estado"),
        rs.getString("telefone"),
        rs.getString("email"),
        rs.getString("cnpj")
        );
        return pessoa;
    }else{
        return null;
    }
}finally{
    connector.close(rs);
    connector.close(stmt);
    connector.close(con);
```

```
        }
    }
    public List<PessoaJuridica> getPessoas() throws SQLException, Exception{
        String sql = "SELECT pj.Pessoa_idPessoa, p.nome, p.logradouro, p.cidade, p.estado, p.telefone, p.email, pj.cnpj "
            + "FROM Pessoa p "
            + "INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.Pessoa_idPessoa;";
        List<PessoaJuridica> lista = new ArrayList<>();
        Connection con = null;
        PreparedStatement stmt = null;
        ResultSet rs = null;
        try{
            con = connector.getConnection();
            stmt = con.prepareStatement(sql);
            rs = stmt.executeQuery();
            while(rs.next()){
                lista.add(new PessoaJuridica(
                rs.getInt("Pessoa_idPessoa"),
                rs.getString("nome"),
                rs.getString("logradouro"),
                rs.getString("cidade"),
                rs.getString("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("cnpj")
                ));
            }
            return lista;
        }finally{
```

```
            connector.close(rs);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public void incluir(PessoaJuridica pessoa) throws SQLException, Exception{
        String sqlPessoa = "INSERT INTO Pessoa(idPessoa,nome,logradouro,cidade,estado,telefone,email)
VALUES(?,?,?,?,?,?,?);";
        String sqlPessoaJuridica = "INSERT INTO PessoaJuridica(Pessoa_idPessoa,cnpj) VALUES(?,?);";
        Connection con = null;
        PreparedStatement stmt = null;
        PreparedStatement stmtPj = null;
        try{
            con = connector.getConnection();
            con.setAutoCommit(false);

            SequenceManager sequenceManager = new SequenceManager();
            int idNovaPessoa = sequenceManager.getValue("orderPessoa");

            stmt = con.prepareStatement(sqlPessoa);
            stmt.setInt(1, idNovaPessoa);
            stmt.setString(2, pessoa.getNome());
            stmt.setString(3, pessoa.getLogradouro());
            stmt.setString(4, pessoa.getCidade());
            stmt.setString(5, pessoa.getEstado());
            stmt.setString(6, pessoa.getTelefone());
            stmt.setString(7, pessoa.getEmail());
            stmt.executeUpdate();
            pessoa.setId(idNovaPessoa);
```

```
            stmtPj = con.prepareStatement(sqlPessoaJuridica);
            stmtPj.setInt(1, idNovaPessoa);
            stmtPj.setString(2, pessoa.getCnpj());
            stmtPj.executeUpdate();
            con.commit();
        }catch(Exception e){
            System.err.println("Adj: " + e);
        }
            finally{
            connector.close(stmtPj);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public void alterar(PessoaJuridica pessoa) throws SQLException, Exception{
        String sqlPessoa = "UPDATE PESSOA SET nome=?,logradouro
=?,cidade=?,estado=?,telefone=?,email=? WHERE idPessoa=?;";
        String sqlPessoaJuridica = "UPDATE PESSOAJURIDICA SET cnpj=? WHERE
Pessoa_idPessoa=?;";
        Connection con = null;
        PreparedStatement stmt = null;
        PreparedStatement stmtPj = null;
        try{
            con = connector.getConnection();
            stmt = con.prepareStatement(sqlPessoa);
            stmtPj = con.prepareStatement(sqlPessoaJuridica);
            stmt.setString(1, pessoa.getNome());
            stmt.setString(2, pessoa.getLogradouro());
            stmt.setString(3, pessoa.getCidade());
            stmt.setString(4, pessoa.getEstado());
```

```
            stmt.setString(5, pessoa.getTelefone());
            stmt.setString(6, pessoa.getEmail());
            stmt.setInt(7, pessoa.getId());
            stmt.executeUpdate();
            stmtPj.setString(1, pessoa.getCnpj());
            stmtPj.setInt(2, pessoa.getId());
            stmtPj.executeUpdate();
        }catch(Exception e){
            System.err.println("Alj: " + e);
        }finally{
            connector.close(stmtPj);
            connector.close(stmt);
            connector.close(con);
        }
    }
    public void excluir(int chave) throws SQLException, Exception{
        String sqlPessoa = "DELETE FROM PESSOA WHERE idPessoa=?";
        String sqlPessoaJuridica = "DELETE FROM PESSOAJURIDICA WHERE Pessoa_idPessoa=?";
        Connection con = null;
        PreparedStatement stmt = null;
        PreparedStatement stmtPj = null;
        try{
            con = connector.getConnection();
            stmtPj = con.prepareStatement(sqlPessoaJuridica);
            stmt = con.prepareStatement(sqlPessoa);
            stmtPj.setInt(1, chave);
            stmtPj.executeUpdate();
            stmt.setInt(1, chave);
            stmt.executeUpdate();
```

```
        }catch(Exception e){
            System.err.println("Exj: " + e);
        }finally{
            connector.close(stmt);
            connector.close(stmtPj);
            connector.close(con);
        }
    }
}
package cadastrobd;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.sql.SQLException;
public class CadastroBDTeste {
    private final PessoaFisicaDAO pessoafisicaDAO;
    private final PessoaJuridicaDAO pessoajuridicaDAO;
    public CadastroBDTeste() {
        pessoafisicaDAO = new PessoaFisicaDAO();
        pessoajuridicaDAO = new PessoaJuridicaDAO();
    }
    private void run() {
        PessoaFisica pf = new PessoaFisica(null, "Lucia", "There", "Loob", "RJ", "333355556",
"FredT@gmail.com", "35353535961");
        try {
            pessoafisicaDAO.incluir(pf);
            System.out.println("==============================");
            System.out.println("Pessoa Fisica id: " + pf.getId() + " adicionada.");
```

```
            System.out.println("==============================");
            pf.exibir();
            System.out.println("==============================");
            pf.setCidade("Uberlandia");
            pf.setLogradouro("Rua Veneza");
            pessoafisicaDAO.alterar(pf);
            System.out.println("Pessoa Fisica id: " + pf.getId() + " dados alterados.");
            System.out.println("==============================");
            pf.exibir();
            System.out.println("==============================");
            System.out.println("Exibindo todas Pessoas Fisicas.");
            System.out.println("==============================");
            pessoafisicaDAO.getPessoas().forEach(pessoaF -> pessoaF.exibir());
            System.out.println("==============================");
            pessoafisicaDAO.excluir(pf.getId());
            System.out.println("Pessoa Fisica id: " + pf.getId() + " excluida.");
        } catch (Exception e) {
            System.err.println("F: " + e);
        }
        PessoaJuridica pj = new PessoaJuridica(null, "Jase", "Priet", "Itabira", "RJ", "38365542",
"JakeHern@gmail.com", "36235965821256");
        try {
            pessoajuridicaDAO.incluir(pj);
            System.out.println("==============================");
            System.out.println("Pessoa Juridica id: " + pj.getId() + " adicionada.");
            System.out.println("==============================");
            pj.exibir();
            System.out.println("==============================");
            pj.setCidade("Ouro Preto");
```

```
            pj.setLogradouro("Rua Direita");
            pessoajuridicaDAO.alterar(pj);
            System.out.println("Pessoa Juridica id: " + pj.getId() + " Dados alterados.");
            System.out.println("============================");
            pj.exibir();
            System.out.println("============================");
            pessoajuridicaDAO.getPessoas().forEach(pessoaJ -> pessoaJ.exibir());
            System.out.println("============================");
            pessoajuridicaDAO.excluir(pj.getId());
            System.out.println("Pessoa Juridica id: " + pj.getId() + " excluida.");
        } catch (Exception e) {
            System.err.println("J: " + e);
        }
    }
    public static void main(String[] args) throws SQLException, Exception {
        new CadastroBDTeste().run();
    }
}
```

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 1º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

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

Os componentes de middleware são essenciais para o desenvolvimento de sistemas de software modernos, pois facilitam a integração, a interoperabilidade e a escalabilidade das aplicações. Ao abstrair a complexidade da infraestrutura, o middleware permite que os desenvolvedores se concentrem na lógica de negócios e acelerem o desenvolvimento de software.

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

A principal diferença entre o Statement e o PreparedStatement é que o PreparedStatement é "preparado" no banco de dados, tornando-o mais rápido. Ou seja, se você fizer diversas consultas parecidas, onde só mudam alguns valores, ele executa mais rapidamente do que se você fizer várias consultas usando o Statement.

Isso acontece porque quando uma consulta chega no banco de dados para ser executada, ela tem que passar por diversos passos de preparação até ser realmente processada. Ao usar o Statement, esses passos são feitos a cada consulta. Já no PreparedStatement, esses passos são feitos apenas uma vez.

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

DAO é a camada do sistema (pacotes, classes e métodos) que abstrai todo o acesso ao banco de dados separadamente da lógica de negócio da aplicação. É no DAO que implementamos os métodos do CRUD (Create – Read – Update – Delete). Bem como é por meio do DAO que as operações no banco de dados são realizadas.

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

Cada hierárquica mapeada reflete uma tabela, relacionadas através do mecanismo de especialização padrão do banco de dados relacional (utilização de chaves estrangeiras). Segunda esta modalidade de mapeamento, tenta-se ao máximo manter a normalização de dados, de forma que a estrutura final das tabelas fica bastante parecida com a hierarquia das classes representada na UML. Esta é a técnica que mais naturalmente mapeia objetos para banco de dados relacionais.

2º Procedimento | Alimentando a Base

Inserir neste campo, **de forma organizada**, todos os códigos do roteiro do 2º Procedimento da Atividade Prática, os resultados da execução do código e a Análise e Conclusão:

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

A persistência em arquivo é uma forma simples de persistir dados em sistemas de computação. Os arquivos são amplamente utilizados para armazenar configurações, logs e outros tipos de informações que não requerem um banco de dados completo, eles são legíveis por humanos, enquanto os arquivos binários são mais eficientes em termos de espaço e desempenho.

A persistência em bancos de dados utiliza sistemas de gerenciamento de banco de dados para armazenar, recuperar e gerenciar dados de maneira estruturada, com uso de tabelas, índices, suporta operações complexas e transações. Também oferece recursos avançados como atomicidade, consistência, isolamento, durabilidade, além de segurança, backup e otimizações para acessos simultâneos.

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

Em Java, a expressão lambda é uma nova característica introduzida na versão 8 da linguagem. Ela permite criar funções anônimas com sintaxe mais concisa e expressiva.

Uma expressão lambda em Java é composta por três partes principais:

- Argumentos: Uma lista de argumentos separados por vírgulas, que especifica os parâmetros que a expressão lambda recebe. Esses parâmetros podem ter tipos explícitos ou podem ser inferidos pelo compilador.
- Setas: O operador "->" separa os argumentos da expressão lambda do corpo da função.
- Corpo: O corpo da função contém a expressão que define a ação a ser realizada pela função. Esse corpo pode ser uma única expressão ou um bloco de código encapsulado em chaves "{}".