



MISSÃO PRÁTICA – NÍVEL 3

Aluno: Daniel Kilzer Brasil Dias | **Matrícula:** 202310422026

Campus: Polo Bezerra de Menezes – Fortaleza/CE

Disciplina: BackEnd Sem Banco Não Tem – **Turma:** 9001 – **Semestre:** 2024.4

Objetivo da Prática

<Descreva nessa seção qual o objetivo da sua prática. Todos os Relatórios de Práticas deverão ser confeccionados em arquivo no formato PDF, com a **Logo da Universidade, nome do Campus, nome do Curso, nome da Disciplina, número da Turma, semestre letivo**. Além disso, o projeto deve ser armazenado em um repositório no GIT e o respectivo endereço deve constar na documentação e essa documentação deve estar no no GIT. O código deve estar versionado no GIT de forma organizada.>

<Lembre-se que a organização contará pontos. Lembre-se também de DELETAR os textos e escrever O SEU RELATÓRIO>

<Esse template é um modelo a ser seguido. O aluno pode optar por seguir outro modelo, desde que atenda a todas as etapas disponíveis na Missão Prática. O documento final deve estar em pdf.>

1º Procedimento | Mapeamento Objeto-Relacional e DAO

CÓDIGOS

Pacote cadastro.util

O pacote `cadastro.model.util` armazenará as classes `ConectorBD` e `SequenceManager`. Tais classes permitirão a conexão com o banco de dados.



A classe ConectorBD permite: conectar-se ao banco de dados, por meio do método `getConnection()`; realizar uma consulta SQL, por meio do método `getPrepared()`; e armazenar o dado tabular da consulta realizar numa estrutura de dados de objeto, por meio do método `getSelect()`. Além disso, as sobrecargas do método `close()` permitem fechar os recursos abertos para se comunicar com o banco de dados.

As constantes `url`, `usuario` e `senha` contêm as informações necessárias para acessar o banco de dados.

Classe ConectorBD

```
package cadastrobd.model.util;

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

public class ConectorBD {
    private static final String url =
"jdbc:sqlserver://localhost:1433;databaseName=Loja;encrypt=
true;trustServerCertificate=true";
    private static final String usuario = "loja";
    private static final String senha = "loja";

    public static Connection getConnection() throws
SQLException {
        return DriverManager.getConnection(url, usuario,
senha);
    }

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

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

    public static void close(Connection conexao) {
        try {
```



Estácio

```
        if (conexao != null) {
            conexao.close();
        }
    }
    catch (SQLException e) {
        System.err.println("Erro ao fechar a
conexao: " + e.getMessage());
    }
}

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

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

public static void close(Connection conexao,
PreparedStatement declaracao) throws SQLException {
    try {
        conexao.close();
        declaracao.close();
    }
    catch (SQLException e) {
        System.err.println("Erro ao tentar encerrar
recursos: " + e.getMessage());
    }
}
```



Estácio

```
public static void close(Connection conexao,
PreparedStatement declaracao, ResultSet conjuntoResultado)
throws SQLException {
    try {
        conexao.close();
        declaracao.close();
        conjuntoResultado.close();
    }
    catch (SQLException e) {
        System.err.println("Erro ao tentar encerrar
recursos: " + e.getMessage());
    }
}
```

A classe SequenceManager lidará com a sequência criada no banco de dados para gerar os identificadores dos registros inseridos, obtendo o próximo valor a ser atribuído a um novo registro por meio de seu método `getValue()`.

Classe SequenceManager

```
package cadastrobd.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 nomeSequencia) throws SQLException {
        String sql = "SELECT NEXT VALUE FOR " + nomeSequencia + "
AS proximoValor";

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao, sql);
            conjuntoResultado = declaracao.executeQuery();

            if (conjuntoResultado.next()) {
                return conjuntoResultado.getInt("proximoValor");
            }
            else {

```



```
        throw new SQLException("Erro ao obter o próximo valor da
sequência: " + nomeSequencia);
    }
}
finally {
    ConectorBD.close(conexao, declaracao,
conjuntoResultado);
}
}
```

Pacote cadastrobd.model

No pacote cadastrobd.model estarão as classes Pessoa, PessoaFisica, PessoaJuridica, PessoaFisicaDAO e PessoaJuridicaDAO.

A classe Pessoa (superclasse) e as classes PessoaFisica e PessoaJuridica (subclasses) instanciarão objetos a serem armazenados em memória quando da execução da aplicação. As classes PessoaFisicaDAO e PessoaJuridicaDAO realizarão a comunicação com o banco de dados, ora recuperando dados, ora inserindo dados e ora alterando dados.

Classe Pessoa

```
package cadastrobd.model;

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

    public Pessoa() {}

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



Estácio

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

public void exibir() {
    System.out.println("-----");
    System.out.println("ID: " + getId());
    System.out.println("Nome: " + getNome());
    System.out.println("Logradouro: " + getLogradouro() + ", " + getCidade() + "/" + getEstado());
    System.out.println("Telefone: " + getTelefone());
    System.out.println("E-mail: " + getEmail());
}
}
```

Classe PessoaFisica

```
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

    public PessoaFisica() {}

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

    public String getCpf() {
        return cpf;
    }

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



```
@Override
public void exhibir() {
    super.exibir();
    System.out.println("CPF: " + getCpf());
}
}
```

Classe PessoaJuridica

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    public PessoaJuridica() {}

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

    public String getCnpj() {
        return cnpj;
    }

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

    @Override
    public void exhibir() {
        super.exibir();
        System.out.println("CNPJ: " + getCnpj());
    }
}
```

Classe PessoaFisicaDAO

```
package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;
import cadastrobd.model.util.SequenceManager;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```




Estácio

```
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {
    public PessoaFisica getPessoa(int id) throws
SQLException {
        String sql = "SELECT p.*, pf.cpf FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa
WHERE idPessoa = ?";

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao,
sql);
            declaracao.setInt(1, id);
            conjuntoResultado =
ConectorBD.getSelect(declaracao);

            if (conjuntoResultado.next()) {
                return new PessoaFisica(

conjuntoResultado.getInt("idPessoa"),
conjuntoResultado.getString("nome"),
conjuntoResultado.getString("logradouro"),
conjuntoResultado.getString("cidade"),
conjuntoResultado.getString("estado"),
conjuntoResultado.getString("telefone"),
conjuntoResultado.getString("email"),
conjuntoResultado.getString("cpf")
                );
            }
        } catch (SQLException e) {
            System.err.println("Erro ao recuperar pessoa
física: " + e.getMessage());
        }
    }
}
```



Estácio

```
        finally {
            ConectorBD.close(conexao,          declaracao,
conjuntoResultado);
        }

        return null;
    }

    public List<PessoaFisica> getPessoas() throws
SQLException {
        List<PessoaFisica> pessoas = new ArrayList<>();

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

        String sql = "SELECT p.*, pf.cpf FROM Pessoa p
INNER JOIN PessoaFisica pf ON p.idPessoa = pf.idPessoa";

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao,
sql);
            conjuntoResultado =
ConectorBD.getSelect(declaracao);

            while (conjuntoResultado.next()) {
                PessoaFisica pessoa = new PessoaFisica(
conjuntoResultado.getInt("idPessoa"),
conjuntoResultado.getString("nome"),
conjuntoResultado.getString("logradouro"),
conjuntoResultado.getString("cidade"),
conjuntoResultado.getString("estado"),
conjuntoResultado.getString("telefone"),
conjuntoResultado.getString("email"),
conjuntoResultado.getString("cpf")
                );
                pessoas.add(pessoa);
            }
        }
```



Estácio

```
        System.out.println("\n=====");
        System.out.println("PESSOAS FÍSICAS CADASTRADAS:");
        for (PessoaFisica pessoa : pessoas) {
            pessoa.exibir();
        }

        System.out.println("=====\\n");
    }
    catch (SQLException e) {
        System.err.println("Erro ao recuperar pessoas físicas: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracao, conjuntoResultado);
    }

    return pessoas;
}

public void incluir(PessoaFisica pessoa) throws SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaFisica = null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        int id = SequenceManager.getValue("SeqIdPessoa");
        pessoa.setId(id);

        String sqlPessoa = "INSERT INTO Pessoa(idPessoa, nome, logradouro, cidade, estado, telefone, email) VALUES(?, ?, ?, ?, ?, ?, ?)";
        declaracaoPessoa = ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2, pessoa.getNome());
        declaracaoPessoa.setString(3, pessoa.getLogradouro());
```



Estácio

```
        declaracaoPessoa.setString(4,
        pessoa.getCidade());
        declaracaoPessoa.setString(5,
        pessoa.getEstado());
        declaracaoPessoa.setString(6,
        pessoa.getTelefone());
        declaracaoPessoa.setString(7,
        pessoa.getEmail());
        declaracaoPessoa.executeUpdate();

        String sqlPessoaFisica = "INSERT INTO
        PessoaFisica(idPessoa, cpf) VALUES(?, ?)";
        declaracaoPessoaFisica =
        ConectorBD.getPrepared(conexao, sqlPessoaFisica);
        declaracaoPessoaFisica.setInt(1, id);
        declaracaoPessoaFisica.setString(2,
        pessoa.getCpf());
        declaracaoPessoaFisica.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa física incluída
        com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
                encerrar conexão: " + ex.getMessage());
            }
        }

        System.err.println("Erro ao tentar incluir
        pessoa física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaFisica);
    }
}

public void alterar(PessoaFisica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
```



Estácio

```
PreparedStatement declaracaoPessoaFisica = null;

try {
    conexao = ConectorBD.getConnection();
    conexao.setAutoCommit(false);

    String sqlPessoa = "UPDATE Pessoa SET nome =
?, logradouro = ?, cidade = ?, estado = ?, telefone = ?,
email = ? WHERE idPessoa = ?";
    declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
    declaracaoPessoa.setString(1,
pessoa.getNome());
    declaracaoPessoa.setString(2,
pessoa.getLogradouro());
    declaracaoPessoa.setString(3,
pessoa.getCidade());
    declaracaoPessoa.setString(4,
pessoa.getEstado());
    declaracaoPessoa.setString(5,
pessoa.getTelefone());
    declaracaoPessoa.setString(6,
pessoa.getEmail());
    declaracaoPessoa.setInt(7, pessoa.getId());
    declaracaoPessoa.executeUpdate();

    String sqlPessoaFisica = "UPDATE
PessoaFisica SET cpf = ? WHERE idPessoa = ?";
    declaracaoPessoaFisica =
ConectorBD.getPrepared(conexao, sqlPessoaFisica);
    declaracaoPessoaFisica.setString(1,
pessoa.getCpf());
    declaracaoPessoaFisica.setInt(2,
pessoa.getId());
    declaracaoPessoaFisica.executeUpdate();

    conexao.commit();

    System.out.println("Pessoa física alterada
com sucesso!");
}
catch (SQLException e) {
    if (conexao != null) {
        try {
            conexao.rollback();
        }
        catch (SQLException ex) {
```



Estácio

```
        System.err.println("Erro ao tentar
encerrar conexão: " + ex.getMessage());
    }
}

        System.err.println("Erro ao tentar alterar a
pessoa física: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaFisica);
    }
}

    public void excluir(int id) throws SQLException {
        Connection conexao = null;
        PreparedStatement declaracaoPessoa = null;
        PreparedStatement declaracaoPessoaFisica = null;

        try {
            conexao = ConectorBD.getConnection();
            conexao.setAutoCommit(false);

            String sqlPessoaFisica = "DELETE FROM
PessoaFisica WHERE idPessoa = ?";
            declaracaoPessoaFisica =
ConectorBD.getPrepared(conexao, sqlPessoaFisica);
            declaracaoPessoaFisica.setInt(1, id);
            declaracaoPessoaFisica.executeUpdate();

            String sqlPessoa = "DELETE FROM Pessoa WHERE
idPessoa = ?";
            declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
            declaracaoPessoa.setInt(1, id);
            declaracaoPessoa.executeUpdate();

            conexao.commit();

            System.out.println("Pessoa física excluída
com sucesso!");
        }
        catch (SQLException e) {
            if (conexao != null) {
                try {
                    conexao.rollback();
                }
            }
        }
    }
}
```



Estácio

```
        catch (SQLException ex) {
            System.err.println("Erro ao tentar
desfazer a transação: " + ex.getMessage());
        }
    }

    System.err.println("Erro ao tentar excluir
pessoa física: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracaoPessoa);
    ConectorBD.close(declaracaoPessoaFisica);
}
}
```

Classe PessoaJuridicaDAO

```
package cadastrobd.model;

import cadastrobd.model.util.ConectorBD;
import cadastrobd.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 {
    public PessoaJuridica getPessoa(int id) throws
SQLException {
        String sql = "SELECT p.*, pj.cnpj FROM Pessoa p
INNER JOIN PessoaJuridica pj ON p.idPessoa = pj.idPessoa
WHERE idPessoa = ?";

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao,
sql);
            declaracao.setInt(1, id);
            conjuntoResultado =
ConectorBD.getSelect(declaracao);

            if (conjuntoResultado.next()) {
```



Estácio

```
        return new PessoaJuridica(
            conjuntoResultado.getInt("idPessoa"),
            conjuntoResultado.getString("nome"),
            conjuntoResultado.getString("logradouro"),
            conjuntoResultado.getString("cidade"),
            conjuntoResultado.getString("estado"),
            conjuntoResultado.getString("telefone"),
            conjuntoResultado.getString("email"),
            conjuntoResultado.getString("cnpj")
        );
    }
    catch (SQLException e) {
        System.err.println("Erro ao tentar recuperar
        pessoa jurídica: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao,          declaracao,
        conjuntoResultado);
    }

    return null;
}

    public List<PessoaJuridica> getPessoas() throws
    SQLException {
        List<PessoaJuridica> pessoas = new ArrayList<>();

        Connection conexao = null;
        PreparedStatement declaracao = null;
        ResultSet conjuntoResultado = null;

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

        try {
            conexao = ConectorBD.getConnection();
            declaracao = ConectorBD.getPrepared(conexao,
            sql);
```




Estácio

```
        conjuntoResultado =
ConectorBD.getSelect(declaracao);

        while (conjuntoResultado.next()) {
            PessoaJuridica pessoa = new
PessoaJuridica(
            conjuntoResultado.getInt("idPessoa"),
            conjuntoResultado.getString("nome"),
            conjuntoResultado.getString("logradouro"),
            conjuntoResultado.getString("cidade"),
            conjuntoResultado.getString("estado"),
            conjuntoResultado.getString("telefone"),
            conjuntoResultado.getString("email"),
            conjuntoResultado.getString("cnpj")
            );

            pessoas.add(pessoa);
        }

        for (PessoaJuridica pessoa : pessoas) {
            pessoa.exibir();
        }
    }
    catch (SQLException e) {
        System.err.println("Erro ao recuperar
pessoas jurídicas: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracao,
conjuntoResultado);
    }

    return pessoas;
}

public void incluir(PessoaJuridica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
```



Estácio

```
PreparedStatement declaracaoPessoaJuridica =
null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        int id =
SequenceManager.getValue("SeqIdPessoa");
        pessoa.setId(id); // O ID gerado pelo banco
de dados é atribuído ao objeto carregado em memória.

        String sqlPessoa = "INSERT INTO
Pessoa(idPessoa, nome, logradouro, cidade, estado,
telefone, email) VALUES(?, ?, ?, ?, ?, ?, ?)";
        declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.setString(2,
pessoa.getNome());
        declaracaoPessoa.setString(3,
pessoa.getLogradouro());
        declaracaoPessoa.setString(4,
pessoa.getCidade());
        declaracaoPessoa.setString(5,
pessoa.getEstado());
        declaracaoPessoa.setString(6,
pessoa.getTelefone());
        declaracaoPessoa.setString(7,
pessoa.getEmail());
        declaracaoPessoa.executeUpdate();

        String sqlPessoaJuridica = "INSERT INTO
PessoaJuridica(idPessoa, cnpj) VALUES(?, ?)";
        declaracaoPessoaJuridica =
ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
        declaracaoPessoaJuridica.setInt(1, id);
        declaracaoPessoaJuridica.setString(2,
pessoa.getCnpj());
        declaracaoPessoaJuridica.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa jurídica incluída
com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
```



Estácio

```
        try {
            conexao.rollback();
        }
        catch (SQLException ex) {
            System.err.println("Erro ao tentar
encerrar a conexão: " + ex.getMessage());
        }
    }

    System.err.println("Erro ao tentar incluir
pessoa jurídica: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracaoPessoa);
    ConectorBD.close(declaracaoPessoaJuridica);
}

}

public void alterar(PessoaJuridica pessoa) throws
SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica =
null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        String sqlPessoa = "UPDATE Pessoa SET nome =
?, logradouro = ?, cidade = ?, estado = ?, telefone = ?,
email = ? WHERE idPessoa = ?";
        declaracaoPessoa =
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setString(1,
pessoa.getNome());
        declaracaoPessoa.setString(2,
pessoa.getLogradouro());
        declaracaoPessoa.setString(3,
pessoa.getCidade());
        declaracaoPessoa.setString(4,
pessoa.getEstado());
        declaracaoPessoa.setString(5,
pessoa.getTelefone());
        declaracaoPessoa.setString(6,
pessoa.getEmail());
        declaracaoPessoa.setInt(7, pessoa.getId());
```



Estácio

```
String sqlPessoaJuridica = "UPDATE
PessoaJuridica SET cnpj = ? WHERE idPessoa = ?";
declaracaoPessoaJuridica =
ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
declaracaoPessoaJuridica.setString(1,
pessoa.getCnpj());
declaracaoPessoaJuridica.setInt(2,
pessoa.getId());

conexao.commit();

System.out.println("Pessoa jurídica alterada
com sucesso!");
}
catch (SQLException e) {
    if (conexao != null) {
        try {
            conexao.rollback();
        }
        catch (SQLException ex) {
            System.err.println("Erro ao tentar
desfazer a transação: " + ex.getMessage());
        }
    }

    System.err.println("Erro ao tentar alterar a
pessoa jurídica: " + e.getMessage());
}
finally {
    ConectorBD.close(conexao, declaracaoPessoa);
    ConectorBD.close(declaracaoPessoaJuridica);
}

}

public void excluir(int id) throws SQLException {
    Connection conexao = null;
    PreparedStatement declaracaoPessoa = null;
    PreparedStatement declaracaoPessoaJuridica =
null;

    try {
        conexao = ConectorBD.getConnection();
        conexao.setAutoCommit(false);

        String sqlPessoaJuridica = "DELETE FROM
PessoaJuridica WHERE idPessoa = ?";
```



Estácio

```
        declaracaoPessoaJuridica
ConectorBD.getPrepared(conexao, sqlPessoaJuridica);
        declaracaoPessoaJuridica.setInt(1, id);
        declaracaoPessoaJuridica.executeUpdate();

        String sqlPessoa = "DELETE FROM Pessoa WHERE
idPessoa = ?";
        declaracaoPessoa
ConectorBD.getPrepared(conexao, sqlPessoa);
        declaracaoPessoa.setInt(1, id);
        declaracaoPessoa.executeUpdate();

        conexao.commit();

        System.out.println("Pessoa jurídica excluída
com sucesso!");
    }
    catch (SQLException e) {
        if (conexao != null) {
            try {
                conexao.rollback();
            }
            catch (SQLException ex) {
                System.err.println("Erro ao tentar
desfazer a transação: " + ex.getMessage());
            }
        }

        System.err.println("Erro ao tentar excluir pessoa
jurídica: " + e.getMessage());
    }
    finally {
        ConectorBD.close(conexao, declaracaoPessoa);
        ConectorBD.close(declaracaoPessoaJuridica);
    }
}
```

Pacote cadastrobd

O pacote cadastrobd armazena a classe CadastroBDTeste, responsável pelo método main, porta de entrada para a aplicação Java.

Aqui serão realizados testes de execução do código escrito até o momento, que corresponde ao código do 1º Procedimento. Os testes envolvem



instanciar objetos de pessoas físicas e jurídicas, incluir, alterar e excluir suas informações no banco de dados.

Classe CadastroDBTeste

```
package cadastrobd;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.sql.SQLException;

public class CadastroDBTeste {
    public static void main(String[] args) throws
SQLException {
        PessoaFisica pessoaFisica = new PessoaFisica(
            0,
            "João Silva",
            "Rua das Flores, nº 123",
            "São Paulo",
            "SP",
            "(11) 98765-4321",
            "joao@email.com",
            "12345678900"
        );

        PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();

        pfDAO.incluir(pessoaFisica);

        pessoaFisica.setLogradouro("Rua dos Cravos, nº
456");

        pfDAO.alterar(pessoaFisica);

        pfDAO.getPessoas();

        pfDAO.excluir(pessoaFisica.getId());

        PessoaJuridica pessoaJuridica = new
PessoaJuridica(
            0,
            "Tech Solutions Ltda.",
            "Rua das Empresas, nº 789",
            "Rio de Janeiro",
            "RJ",
            "(21) 98123-4567",
```



Estácio

```
        "contato@techsolutions.com",
        "12345678000199"
    );

    PessoaJuridicaDAO pjDAO = new
PessoaJuridicaDAO();

    pjDAO.incluir(pessoaJuridica);

    pessoaJuridica.setNome("Tech Solutions S.A.");
    pessoaJuridica.setLogradouro("Avenida      das
Empresas, nº 321");

    pjDAO.alterar(pessoaJuridica);

    pjDAO.getPessoas();

    pjDAO.excluir(pessoaJuridica.getId());
    }
}
```



RESULTADOS

Após a execução do código escrito até aqui, temos a criação de uma pessoa física e uma pessoa jurídica, bem como suas respectivas inclusão, alteração e exclusão do banco de dados, além da exibição de todas as pessoas físicas e jurídicas presentes no banco de dados.

Na figura a seguir, podemos ver o resultado da execução do código na saída do console do NetBeans 24.

```
Output x
CadastroBD (run) x daniel.dias - C:\Users\daniel.dias x

run:
Pessoa fisica incluida com sucesso!
Pessoa fisica alterada com sucesso!

=====
PESSOAS FÍSICAS CADASTRADAS:
=====
ID: 1
Nome: João Silva
Logradouro: Rua dos Cravos, n° 456, São Paulo/SP
Telefone: (11) 98765-4321
E-mail: joao@email.com
CPF: 12345678900
=====

Pessoa fisica excluida com sucesso!
Pessoa juridica incluida com sucesso!
Pessoa juridica alterada com sucesso!

=====
PESSOAS JURÍDICAS CADASTRADAS:
=====
ID: 2
Nome: Tech Solutions Ltda.
Logradouro: Rua das Empresas, n° 789, Rio de Janeiro/RJ
Telefone: (21) 98123-4567
E-mail: contato@techsolutions.com
CNPJ: 12345678000199
=====

Pessoa juridica excluida com sucesso!
BUILD SUCCESSFUL (total time: 0 seconds)
```

ANÁLISE

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

Os componentes de *middleware*, como o JDBC (*Java Database Connectivity*), são fundamentais para permitir a comunicação entre uma aplicação Java e um banco de dados relacional. Alguns pontos que tornam interessante o uso de uma API como essa são:



Estácio

- **Abstração do banco de dados:** oferece recursos importantes como gerenciamento de conexões, transações e tratamento de erros, que seriam muito complexos de implementar manualmente. Isso permite que o código Java se comunique com o banco de dados sem precisar conhecer detalhes específicos do protocolo de comunicação
- **Facilidade de uso e manutenção:** simplifica operações como conexão, execução de consultas SQL e manipulação de resultados. No nosso projeto, isso é evidente no ConectorBD, que encapsula toda complexidade de conexão em métodos simples como `getConnection()`.
- **Portabilidade:** como o JDBC é parte do Java SE (*Java Standard Edition*), ele funciona em qualquer plataforma que suporte Java. Isso permite trocar o banco de dados (SQL Server, MySQL, Oracle, etc.) com mínimas alterações no código.
- **Segurança:** oferece suporte a transações, *prepared statements* e outros mecanismos que ajudam a evitar vulnerabilidades como injeção de SQL.

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

O *Statement* é um objeto usado para a execução estática (sem parâmetros) de uma consulta SQL e retorna o resultado produzido. Já o *PreparedStatement* representa uma consulta SQL pré-compilada.

Na prática, se uma consulta SQL precisa ser executada várias vezes, pode ser mais interessante utilizar um *PreparedStatement*, que costuma reduzir o tempo de execução, pois a consulta já é pré-compilada, não sendo necessário que seja compilada pelo banco de dados a cada execução.

Em termos de segurança, o *PreparedStatement* mostra-se como uma melhor opção ao prevenir ataques de injeção de SQL. Isso porque no *Statement* os dados fornecidos pelo usuário são concatenados na *String* a ser



utilizada na consulta, enquanto o *PreparedStatement* utiliza marcadores de posição (*placeholders*) para consultas parametrizadas e os parâmetros são tratados de forma segura.

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

O DAO (*Data Access Object*) é um padrão de projeto que separa a lógica de acesso a dados (como consultas SQL) da lógica de negócios da aplicação.

A manutenibilidade do software é melhorada por:

- **Separação de responsabilidades:** o código de acesso ao banco de dados fica isolado em classes específicas (DAOs), facilitando a organização e o entendimento do código. Com isso, alterações na estrutura do banco ou na forma como os dados são manipulados podem ser feitas nas classes DAO sem impactar outras partes do sistema.
- **Reutilização de código:** operações comuns de CRUD (*Create, Read, Update, Delete*) podem ser reutilizadas em diferentes partes da aplicação.
- **Flexibilidade:** se o banco de dados for alterado, apenas as classes DAO precisam ser modificadas, sem afetar o restante da aplicação.
- **Testabilidade:** as classes DAO podem ser testadas unitariamente de forma mais fácil, pois não dependem da lógica de negócios.

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

Em bancos de dados relacionais, a herança (um conceito de orientação a objetos) não é diretamente suportada. No entanto, existem estratégias para mapear hierarquias de classes em tabelas:



Estácio

- **Tabela única (*single table*):** Todas as classes da hierarquia são mapeadas para uma única tabela.
 - Uma coluna discriminadora indica a classe específica de cada registro.
 - Em nosso projeto, isso seria implementado utilizando apenas a tabela Pessoa para todos os tipos de pessoa, utilizando uma coluna tipo para indicar se se trata de pessoa física ou jurídica.
 - Pode gerar colunas vazias para atributos que não se aplicam a todas as classes. No nosso caso, um registro de pessoa física ficaria sempre com o atributo do CNPJ vazio e um registro de pessoa jurídica ficaria sempre com o atributo do CPF vazio.
- **Tabela por classe (*table per class*):** cada classe concreta tem sua própria tabela.
 - As tabelas filhas contêm colunas para os atributos da classe pai.
 - Repetição de atributos comuns.
 - Em nosso projeto, isso seria implementado utilizando classes para as tabelas PessoaFisica e PessoaJuridica, sem uma tabela para a classe Pessoa. Isso faria com que campos comuns fossem repetidos em ambas tabelas, como os campos nome, logradouro, cidade etc.
- **Tabela por subclasse (*joined table*):** cada classe na hierarquia é mapeada para uma tabela, mas os atributos da classe pai são armazenados em uma tabela separada.
 - O relacionamento entre as tabelas se dá através de chaves estrangeiras (*foreign keys*).



Estácio

- Esta foi a abordagem escolhida para nosso projeto e apresenta vantagens e desvantagens.
 - Vantagens
 - Não há duplicação de campos comuns.
 - Integridade referencial garantida pelo banco de dados.
 - Facilidade para consultas específicas de cada tipo.
 - Desvantagens
 - Necessidade de *joins* para recuperar dados completos.
 - Operações de inclusão e exclusão precisam tratar múltiplas tabelas.
 - Transações são necessárias para manter consistência.

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?
- b) Como o uso de operador *lambda* simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?
- c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como *static*?

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.



Conclusão

<Elabore uma análise crítica da sua Missão Prática.>