



BackEnd sem banco não tem

Ramon Barros Correa, 202208939635

Inserir aqui o Campus: Polo Sete lagoas (MG)
Nível 2 – Vamos manter informações, turma 2022.2, semestre 2024.1


Objetivo da Prática






Implementar persistência com base no middleware JDBC. Utilizar o padrão DAO (Data Access Object) no manuseio de dados. Implementar o mapeamento objeto-relacional em sistemas Java. Criar sistemas cadastrais com persistência em banco relacional. No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQLServer na persistência de dados.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

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:

Criando o pacote model

▼  cadastrobd.model

-  Pessoas.java
-  PessoasFisicas.java
-  PessoasFisicasDAO.java
-  PessoasJuridicas.java
-  PessoasJuridicasDAO.java

```
package cadastrobd.model;
```

```
public class Pessoas {  
    private int ID;  
    private String Nome;  
    private String Endereco;  
    private String Telefone;  
  
    // Construtor padrão  
    public Pessoas() {  
    }  
  
    // Construtor completo  
    public Pessoas(int id, String nome, String endereco, String telefone) {  
        this.ID = id;  
        this.Nome = nome;  
        this.Endereco = endereco;  
        this.Telefone = telefone;  
    }  
  
    // Método para exibir os dados  
    public void exibir() {  
        System.out.println("ID: " + ID);  
        System.out.println("Nome: " + Nome);  
        System.out.println("Logradouro: " + Endereco);  
        System.out.println("Telefone: " + Telefone);  
    }  
}
```

```
// Getters e setters
```

```
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 getEndereco() {  
    return Endereco;  
}
```

```
public void setEndereco(String Endereco) {  
    this.Endereco = Endereco;  
}
```

```
public String getTelefone() {  
    return Telefone;  
}
```

```
package cadastrobd.model;
```

```
public class PessoasFisicas extends Pessoas {
    private String CPF;

    // Construtor padrão
    public PessoasFisicas() {
    }

    // Construtor completo
    public PessoasFisicas(int ID, String Nome, String Endereco, String Telefone, String cpf) {
        super(ID, Nome, Endereco, Telefone);
        this.CPF = cpf;
    }

    // Método para exibir os dados
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + CPF);
    }

    // Getters e setters
    public String getCpf() {
        return CPF;
    }

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

```
package cadastrobd.model;
```

```
public class PessoasJuridicas extends Pessoas {
    private String CNPJ;

    // Construtor padrão
    public PessoasJuridicas() {
    }


    // Construtor completo
    public PessoasJuridicas(int ID, String Nome, String Endereco, String Telefone, String CNPJ) {
        super(ID, Nome, Endereco, Telefone);
        this.CNPJ = CNPJ;
    }



    // Método para exibir os dados
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + CNPJ);
    }

    // Getters e setters
    public String getCnpj() {
        return CNPJ;
    }

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

Criando pacote model.util

▼  cadastro.model.util

-  ConectorBD.java
-  SequenceManager.java

```
package cadastro.model.util;

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

public class ConectorBD {
    private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true";
    private static final String USER = "loja";
    private static final String PASSWORD = "loja";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

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

    public static ResultSet getSelect(String sql) throws SQLException {
        return getPrepared(sql).executeQuery();
    }
}
```

```
public static void close(Connection conn) {
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
        }
    }
}
```

```
public static void close(Statement stmt) {
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
        }
    }
}
```

```
public static void close(ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
        }
    }
}
```

Classes no padrao DAO

```
package cadastrabd.model;

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

public class PessoasFisicasDAO {
    private final Connection conexao;

    public PessoasFisicasDAO() throws SQLException {
        this.conexao = ConectorBD.getConnection();
    }

    public PessoasFisicas getPessoas(int id) throws SQLException {
        String sql = "SELECT * FROM Pessoas INNER JOIN PessoasFisicas ON Pessoas.ID = PessoasFisicas.ID WHERE Pessoas.ID = ?";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return criarPessoaFisica(rs);
            }
        }
        return null;
    }

    public List<PessoasFisicas> getPessoas() throws SQLException {
        List<PessoasFisicas> pessoas = new ArrayList<>(255);
        String sql = "SELECT * FROM Pessoas INNER JOIN PessoasFisicas ON Pessoas.ID = PessoasFisicas.ID";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                pessoas.add(criarPessoaFisica(rs));
            }
        }
        return pessoas;
    }

    public int getNextPessoaId() throws SQLException {
        String sql = "SELECT NEXT VALUE FOR Seq_Pessoa_ID";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return rs.getInt(1);
            } else {
                throw new SQLException("Failed to get next Pessoa ID from sequence");
            }
        }
    }

    public void incluir(PessoasFisicas pessoa) throws SQLException {
        int id = getNextPessoaId();
        String sqlPessoa = "INSERT INTO Pessoas (ID, Nome, Endereco, Telefone) VALUES (?, ?, ?, ?)";
        String sqlPessoaFisica = "INSERT INTO PessoasFisicas (ID, CPF) VALUES (?, ?)";
        try (PreparedStatement stmtPessoa = conexao.prepareStatement(sqlPessoa);
            PreparedStatement stmtPessoaFisica = conexao.prepareStatement(sqlPessoaFisica)) {
            stmtPessoa.setInt(1, id);
            stmtPessoa.setString(2, pessoa.getNome());
            stmtPessoa.setString(3, pessoa.getEndereco());
            stmtPessoa.setString(4, pessoa.getTelefone());
            stmtPessoa.executeUpdate();
            stmtPessoaFisica.setInt(1, id);
            stmtPessoaFisica.setString(2, pessoa.getCpf());
            stmtPessoaFisica.executeUpdate();
        }
    }
}
```

```

public void alterar(PessoasFisicas pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoas SET Nome=?, Endereco=?, Telefone=? WHERE ID=?";
    String sqlPessoaFisica = "UPDATE PessoasFisicas SET CPF=? WHERE ID=?";
    try (PreparedStatement stmtPessoa = conexao.prepareStatement(sqlPessoa);

```

```

package cadastrobd.model;

```

```

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

```

```

public class PessoasJuridicasDAO {
    private final Connection conexao;

```

```

    public PessoasJuridicasDAO() throws SQLException {
        this.conexao = ConectorBD.getConnection();
    }

```

```

    public PessoasJuridicas getPessoa(int id) throws SQLException {
        String sql = "SELECT * FROM Pessoas INNER JOIN PessoasJuridicas ON Pessoas.ID = PessoasJuridicas.ID WHERE Pessoas.ID = ?";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return criarPessoaJuridica(rs);
            }
        }
        return null;
    }

```

```

    private PessoasJuridicas criarPessoaJuridica(ResultSet rs) throws SQLException {

```

```

    }

```

```

    private PessoasFisicas criarPessoaFisica(ResultSet rs) throws SQLException {
        PessoasFisicas pessoa = new PessoasFisicas();
        pessoa.setId(rs.getInt("ID"));
        pessoa.setNome(rs.getString("Nome"));
        pessoa.setEndereco(rs.getString("Endereco"));
        pessoa.setTelefone(rs.getString("Telefone"));
        pessoa.setCpf(rs.getString("CPF"));
        return pessoa;
    }

```

```

package cadastrobd.model;

```

```

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

```

```

public class PessoasJuridicasDAO {
    private final Connection conexao;

```

```

    public PessoasJuridicasDAO() throws SQLException {
        this.conexao = ConectorBD.getConnection();
    }

```

```

    public PessoasJuridicas getPessoa(int id) throws SQLException {
        String sql = "SELECT * FROM Pessoas INNER JOIN PessoasJuridicas ON Pessoas.ID = PessoasJuridicas.ID WHERE Pessoas.ID = ?";
        try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
            stmt.setInt(1, id);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                return criarPessoaJuridica(rs);
            }
        }
        return null;
    }

```

```

public List<PessoasJuridicas> getPessoas() throws SQLException {
    List<PessoasJuridicas> pessoas = new ArrayList<>();
    String sql = "SELECT * FROM Pessoas INNER JOIN PessoasJuridicas ON Pessoas.ID = PessoasJuridicas.ID";
    try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            pessoas.add(criarPessoaJuridica(rs));
        }
    }
    return pessoas;
}

public int getNextPessoaId() throws SQLException {
    String sql = "SELECT NEXT VALUE FOR Seq_Pessoa_ID";
    try (PreparedStatement stmt = conexao.prepareStatement(sql)) {
        ResultSet rs = stmt.executeQuery() {
            if (rs.next()) {
                return rs.getInt(1);
            } else {
                throw new SQLException("Failed to get next Pessoa ID from sequence");
            }
        }
    }
}

```

```

public void incluir(PessoasJuridicas pessoa) throws SQLException {
    int id = getNextPessoaId();
    String sqlPessoa = "INSERT INTO Pessoas (ID, Nome, Endereco, Telefone) VALUES (?, ?, ?, ?)";
    String sqlPessoaJuridica = "INSERT INTO PessoasJuridicas (ID, CNPJ) VALUES (?, ?)";
    try (PreparedStatement stmtPessoa = conexao.prepareStatement(sqlPessoa);
        PreparedStatement stmtPessoaJuridica = conexao.prepareStatement(sqlPessoaJuridica)) {
        stmtPessoa.setInt(1, id);
        stmtPessoa.setString(2, pessoa.getNome());
        stmtPessoa.setString(3, pessoa.getEndereco());
        stmtPessoa.setString(4, pessoa.getTelefone());
        stmtPessoa.executeUpdate();

        stmtPessoaJuridica.setInt(1, id);
        stmtPessoaJuridica.setString(2, pessoa.getCnpj());
        stmtPessoaJuridica.executeUpdate();
    }
}

```

```

public void alterar(PessoasJuridicas pessoa) throws SQLException {
    String sqlPessoa = "UPDATE Pessoas SET Nome=?, Endereco=?, Telefone=? WHERE ID=?";
    String sqlPessoaJuridica = "UPDATE PessoasJuridicas SET CNPJ=? WHERE id=?";
    try (PreparedStatement stmtPessoa = conexao.prepareStatement(sqlPessoa);
        PreparedStatement stmtPessoaJuridica = conexao.prepareStatement(sqlPessoaJuridica)) {
        stmtPessoa.setString(1, pessoa.getNome());
        stmtPessoa.setString(2, pessoa.getEndereco());
        stmtPessoa.setString(3, pessoa.getTelefone());
        stmtPessoa.setInt(4, pessoa.getId());
        stmtPessoaJuridica.setString(1, pessoa.getCnpj());
        stmtPessoaJuridica.setInt(2, pessoa.getId());
        stmtPessoa.executeUpdate();
        stmtPessoaJuridica.executeUpdate();
    }
}

```

```

public void excluir(int id) throws SQLException {
    String sqlPessoaJuridica = "DELETE FROM PessoasJuridicas WHERE ID=?";
    String sqlPessoa = "DELETE FROM Pessoas WHERE ID=?";
    try (PreparedStatement stmtPessoaJuridica = conexao.prepareStatement(sqlPessoaJuridica);
        PreparedStatement stmtPessoa = conexao.prepareStatement(sqlPessoa)) {
        stmtPessoaJuridica.setInt(1, id);
        stmtPessoa.setInt(1, id);
        stmtPessoaJuridica.executeUpdate();
        stmtPessoa.executeUpdate();
    }
}

```

```

private PessoasJuridicas criarPessoaJuridica(ResultSet rs) throws SQLException {
    PessoasJuridicas pessoa = new PessoasJuridicas();
    pessoa.setId(rs.getInt("id"));
    pessoa.setNome(rs.getString("nome"));
    pessoa.setEndereco(rs.getString("Endereco"));
    pessoa.setTelefone(rs.getString("telefone"));
    pessoa.setCnpj(rs.getString("cnpj"));
    return pessoa;
}

```

```

}

```

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

Abstração de Banco de Dados, Conectividade de Dados, Gerenciamento de Conexões, Suporte a Transações, Portabilidade e etc.

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

o PreparedStatement é geralmente preferido sobre o Statement devido à sua segurança aprimorada contra injeção de SQL, melhor desempenho em operações repetitivas e melhor legibilidade e manutenção do código. No entanto, em situações onde a consulta SQL é simples e não envolve entrada do usuário, o Statement pode ser adequado.

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

Separação de Responsabilidades, Abstração de Implementação de Banco de Dados, Reutilização de Código, Facilita Testes Unitários, Padronização das Operações de Acesso a Dados e etc.

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

Tabelas Separadas (Concrete Table Inheritance):

Cada classe concreta no modelo de classes tem sua própria tabela correspondente no banco de dados. As tabelas filhas contêm todas as colunas associadas à classe filha, além de quaisquer colunas adicionais específicas dessa classe.

Existe uma relação de um-para-um entre as tabelas filhas e a tabela pai.

Esta abordagem é simples e intuitiva, mas pode levar a redundância de dados e dificuldade em garantir a integridade referencial.

Tabela Única (Single Table Inheritance):

Todas as classes concretas e a classe pai são mapeadas para uma única tabela no banco de dados.

A tabela única contém todas as colunas associadas a todas as classes no modelo de classes, incluindo colunas adicionais para identificar a classe concreta de cada registro.

Esta abordagem reduz a redundância de dados, mas pode resultar em uma tabela grande e complexa, especialmente se houver muitos atributos exclusivos para cada classe.

Tabelas de Associação (Class Table Inheritance):

Cada classe no modelo de classes é mapeada para sua própria tabela no banco de dados.

A tabela pai contém os atributos comuns a todas as classes, enquanto as tabelas filhas contêm apenas os atributos exclusivos de cada classe.

As tabelas filhas têm uma relação de um-para-um com a tabela pai e podem ter uma relação de um-para-um entre si, se houver herança múltipla.

Esta abordagem combina os benefícios das abordagens anteriores, reduzindo a redundância de dados e mantendo uma estrutura mais organizada.

A escolha da abordagem depende dos requisitos específicos do aplicativo, das características da hierarquia de classes e das preferências de design do banco de dados. Cada abordagem tem suas vantagens e desvantagens em termos de desempenho, complexidade e facilidade de manutenção.

2º Procedimento | Alimentando a Base

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:

Alterando o Main

```
package cadastrobd;

import cadastrobd.model.PessoasFisicas;
import cadastrobd.model.PessoasFisicasDAO;
import cadastrobd.model.PessoasJuridicas;
import cadastrobd.model.PessoasJuridicasDAO;
import java.sql.SQLException;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

public class CadastroBDPrincipal {

    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            int opcao;
            do {
                System.out.println("Selecione uma opção:");
                System.out.println("1. Incluir");
                System.out.println("2. Alterar");
                System.out.println("3. Excluir");
                System.out.println("4. Exibir pelo ID");
                System.out.println("5. Exibir todos");
                System.out.println("0. Finalizar");

                opcao = scanner.nextInt();
                scanner.nextLine(); // Limpar o buffer de entrada

                switch (opcao) {
                    case 1 ->
                        incluir(scanner);
                    case 2 ->
                        alterar(scanner);
                    case 3 ->
                        excluir(scanner);
                    case 4 ->
                        exibirPorId(scanner);
                    case 5 ->
                        exibirTodos();
                    case 0 ->
                        System.out.println("Finalizando o programa...");
                    default ->
                        System.out.println("Opção inválida. Tente novamente.");
                }
            } while (opcao != 0);

        } catch (InputMismatchException e) {
            System.out.println("Entrada inválida. O programa será encerrado.");
        }
    }
}
```

```

private static void incluir(Scanner scanner) {
    System.out.println("Selecione o tipo de pessoa (1 para Física, 2 para Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada
    switch (tipo) {
        case 1 ->
            incluirPessoaFisica(scanner);
        case 2 ->
            incluirPessoaJuridica(scanner);
        default ->
            System.out.println("Tipo de pessoa inválido.");
    }
}

private static void incluirPessoaFisica(Scanner scanner) {
    // Obter dados da pessoa física do usuário
    System.out.println("Digite o nome da pessoa física:");
    String nome = scanner.nextLine();

    System.out.println("Digite o CPF da pessoa física:");
    String cpf = scanner.nextLine();

    System.out.println("Digite o endereço da pessoa física:");
    String endereco = scanner.nextLine();

    System.out.println("Digite o telefone da pessoa física:");
    String telefone = scanner.nextLine();

    // Criar objeto PessoasFisicas e persistir no banco de dados
    PessoasFisicas pessoaFisica = new PessoasFisicas();
    pessoaFisica.setNome(nome);
    pessoaFisica.setCpf(cpf);
    pessoaFisica.setEndereco(endereco);
    pessoaFisica.setTelefone(telefone);

    try {
        PessoasFisicasDAO pessoaFisicaDAO = new PessoasFisicasDAO();
        pessoaFisicaDAO.incluir(pessoaFisica);
        System.out.println("Pessoa física incluída com sucesso!");
    } catch (SQLException e) {
        System.out.println("Erro ao incluir pessoa física: " + e.getMessage());
    }
}

```

```

private static void incluirPessoaJuridica(Scanner scanner) {
    // Obter dados da pessoa jurídica do usuário
    System.out.println("Digite o nome da pessoa jurídica:");
    String nome = scanner.nextLine();

    System.out.println("Digite o CNPJ da pessoa jurídica:");
    String cnpj = scanner.nextLine();

    System.out.println("Digite o endereço da pessoa jurídica:");
    String endereco = scanner.nextLine();

    System.out.println("Digite o telefone da pessoa jurídica:");
    String telefone = scanner.nextLine();

    // Criar objeto PessoasJuridicas e persistir no banco de dados
    PessoasJuridicas pessoaJuridica = new PessoasJuridicas();
    pessoaJuridica.setNome(nome);
    pessoaJuridica.setCnpj(cnpj);
    pessoaJuridica.setEndereco(endereco);
    pessoaJuridica.setTelefone(telefone);

    // Chamar o método para incluir a pessoa jurídica no banco de dados
    try {
        PessoasJuridicasDAO pessoaJuridicaDAO = new PessoasJuridicasDAO();
        pessoaJuridicaDAO.incluir(pessoaJuridica);
        System.out.println("Pessoa jurídica incluída com sucesso!");
    } catch (SQLException ex) {
        System.out.println("Erro ao incluir pessoa jurídica: " + ex.getMessage());
    }
}

```

```

private static void alterar(Scanner scanner) {
    System.out.println("Selecione o tipo de pessoa para alterar (1 para Física, 2 para Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada

    System.out.println("Digite o ID da pessoa:");
    int id = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada
    switch (tipo) {
        case 1 ->
            alterarPessoaFisica(scanner, id);
        case 2 ->
            alterarPessoaJuridica(scanner, id);
        default ->
            System.out.println("Tipo de pessoa inválido.");
    }
}

```

```

private static void alterarPessoaFisica(Scanner scanner, int id) {
    try {
        // Instanciar o DAO para acessar os métodos de acesso ao banco de dados
        PessoasFisicasDAO pessoaFisicaDAO = new PessoasFisicasDAO();

        // Obter a pessoa física com o ID informado
        PessoasFisicas pessoaFisica = pessoaFisicaDAO.getPessoas(id);

        if (pessoaFisica != null) {
            // Exibir os dados atuais da pessoa física
            System.out.println("Nome atual: " + pessoaFisica.getNome());
            System.out.println("CPF atual: " + pessoaFisica.getCpf());
            System.out.println("Endereço atual: " + pessoaFisica.getEndereco());
            System.out.println("Telefone atual: " + pessoaFisica.getTelefone());

            // Solicitar os novos dados da pessoa física ao usuário
            System.out.println("Digite o novo nome:");
            String novoNome = scanner.nextLine();
            System.out.println("Digite o novo CPF:");
            String novoCpf = scanner.nextLine();
            System.out.println("Digite o novo endereço:");
            String novoEndereco = scanner.nextLine();
            System.out.println("Digite o novo telefone:");
            String novoTelefone = scanner.nextLine();

            // Atualizar os dados da pessoa física
            pessoaFisica.setNome(novoNome);
            pessoaFisica.setCpf(novoCpf);
            pessoaFisica.setEndereco(novoEndereco);
            pessoaFisica.setTelefone(novoTelefone);

            // Chamar o método para alterar a pessoa física no banco de dados
            pessoaFisicaDAO.alterar(pessoaFisica);
            System.out.println("Pessoa física alterada com sucesso!");
        } else {
            System.out.println("Pessoa física não encontrada com o ID informado.");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao alterar pessoa física: " + e.getMessage());
    }
}

```

```

private static void alterarPessoaJuridica(Scanner scanner, int id) {
    try {
        // Instanciar o DAO para acessar os métodos de acesso ao banco de dados
        PessoasJuridicasDAO pessoaJuridicaDAO = new PessoasJuridicasDAO();

        // Obter a pessoa jurídica com o ID informado
        PessoasJuridicas pessoaJuridica = pessoaJuridicaDAO.getPessoa(id);

        if (pessoaJuridica != null) {
            // Exibir os dados atuais da pessoa jurídica
            System.out.println("Nome atual: " + pessoaJuridica.getNome());
            System.out.println("CNPJ atual: " + pessoaJuridica.getCnpj());
            System.out.println("Endereço atual: " + pessoaJuridica.getEndereco());
            System.out.println("Telefone atual: " + pessoaJuridica.getTelefone());

            // Solicitar os novos dados da pessoa jurídica ao usuário
            System.out.println("Digite o novo nome:");
            String novoNome = scanner.nextLine();
            System.out.println("Digite o novo CNPJ:");
            String novoCnpj = scanner.nextLine();
            System.out.println("Digite o novo endereço:");
            String novoEndereco = scanner.nextLine();
            System.out.println("Digite o novo telefone:");
            String novoTelefone = scanner.nextLine();

            // Atualizar os dados da pessoa jurídica
            pessoaJuridica.setNome(novoNome);
            pessoaJuridica.setCnpj(novoCnpj);
            pessoaJuridica.setEndereco(novoEndereco);
            pessoaJuridica.setTelefone(novoTelefone);

            // Chamar o método para alterar a pessoa jurídica no banco de dados
            pessoaJuridicaDAO.alterar(pessoaJuridica);
            System.out.println("Pessoa jurídica alterada com sucesso!");
        } else {
            System.out.println("Pessoa jurídica não encontrada com o ID informado.");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao alterar pessoa jurídica: " + e.getMessage());
    }
}

```

```

private static void excluir(Scanner scanner) {
    System.out.println("Selecione o tipo de pessoa para excluir (1 para Física, 2 para Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada

    System.out.println("Digite o ID da pessoa:");
    int id = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada

    switch (tipo) {
        case 1 ->
            excluirPessoaFisica(id);
        case 2 ->
            excluirPessoaJuridica(id);
        default ->
            System.out.println("Tipo de pessoa inválido.");
    }
}

```

```

private static void excluirPessoaFisica(int id) {
    try {
        PessoasFisicasDAO pessoaFisicaDAO = new PessoasFisicasDAO();
        pessoaFisicaDAO.excluir(id);
        System.out.println("Pessoa física excluída com sucesso!");
    } catch (SQLException e) {
        System.out.println("Erro ao excluir pessoa física: " + e.getMessage());
    }
}

```

```

private static void excluirPessoaJuridica(int id) {
    try {
        PessoasJuridicasDAO pessoaJuridicaDAO = new PessoasJuridicasDAO();
        pessoaJuridicaDAO.excluir(id);
        System.out.println("Pessoa jurídica excluída com sucesso!");
    } catch (SQLException e) {
        System.out.println("Erro ao excluir pessoa jurídica: " + e.getMessage());
    }
}

```

```

private static void exibirPorId(Scanner scanner) {
    System.out.println("Selecione o tipo de pessoa para exibir pelo ID (1 para Física, 2 para Jurídica):");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada

    System.out.println("Digite o ID da pessoa:");
    int id = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer de entrada
    switch (tipo) {
        case 1 ->
            exibirPessoaFisicaPorId(id);
        case 2 ->
            exibirPessoaJuridicaPorId(id);
        default ->
            System.out.println("Tipo de pessoa inválido.");
    }
}

private static void exibirPessoaFisicaPorId(int id) {
    try {
        PessoasFisicasDAO pessoaFisicaDAO = new PessoasFisicasDAO();
        PessoasFisicas pessoaFisica = pessoaFisicaDAO.getPessoas(id);

        if (pessoaFisica != null) {
            System.out.println("Dados da pessoa física:");
            System.out.println("ID: " + pessoaFisica.getId());
            System.out.println("Nome: " + pessoaFisica.getNome());
            System.out.println("CPF: " + pessoaFisica.getCpf());
        } else {
            System.out.println("Pessoa física não encontrada com o ID informado.");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao exibir pessoa física por ID: " + e.getMessage());
    }
}

```

```

private static void exibirPessoaJuridicaPorId(int id) {
    try {
        PessoasJuridicasDAO pessoaJuridicaDAO = new PessoasJuridicasDAO();
        PessoasJuridicas pessoaJuridica = pessoaJuridicaDAO.getPessoa(id);

        if (pessoaJuridica != null) {
            System.out.println("Dados da pessoa jurídica:");
            System.out.println("ID: " + pessoaJuridica.getId());
            System.out.println("Nome: " + pessoaJuridica.getNome());
            System.out.println("CNPJ: " + pessoaJuridica.getCnpj());
        } else {
            System.out.println("Pessoa jurídica não encontrada com o ID informado.");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao exibir pessoa jurídica por ID: " + e.getMessage());
    }
}

private static void exibirTodos() {
    try {
        PessoasFisicasDAO pessoaFisicaDAO = new PessoasFisicasDAO();
        List<PessoasFisicas> pessoasFisicas = pessoaFisicaDAO.getPessoas();

        System.out.println("Pessoas físicas cadastradas:");
        for (PessoasFisicas pessoaFisica : pessoasFisicas) {
            System.out.println("ID: " + pessoaFisica.getId() + ", Nome: " + pessoaFisica.getNome() + ", CPF: " + pessoaFisica.getCpf());
        }

        PessoasJuridicasDAO pessoaJuridicaDAO = new PessoasJuridicasDAO();
        List<PessoasJuridicas> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();

        System.out.println("Pessoas jurídicas cadastradas:");
        for (PessoasJuridicas pessoaJuridica : pessoasJuridicas) {
            System.out.println("ID: " + pessoaJuridica.getId() + ", Nome: " + pessoaJuridica.getNome() + ", CNPJ: " + pessoaJuridica.getCnpj());
        }
    } catch (SQLException e) {
        System.out.println("Erro ao exibir todas as pessoas: " + e.getMessage());
    }
}

```

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

Formato de Armazenamento:

Persistência em Arquivo: Os dados são armazenados em um arquivo ou conjunto de arquivos no sistema de arquivos do sistema operacional. Os arquivos podem ser em formatos simples, como texto ou binário, e podem ser organizados de várias maneiras, como texto estruturado, XML, JSON, CSV, ou mesmo formatos binários específicos.

Persistência em Banco de Dados: Os dados são armazenados em tabelas dentro de um sistema de gerenciamento de banco de dados (SGBD). Os bancos de dados relacionais usam uma estrutura de tabelas com linhas e colunas, enquanto os bancos de dados NoSQL podem usar diferentes estruturas de armazenamento, como documentos, grafos ou pares chave-valor.

Recursos e Funcionalidades:

Persistência em Arquivo: Oferece menos recursos e funcionalidades em comparação com bancos de dados. Normalmente, não há suporte para consultas complexas, transações, controle de concorrência ou outras características avançadas de bancos de dados.

Persistência em Banco de Dados: Fornece uma ampla gama de recursos e funcionalidades, como suporte a consultas SQL, transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), controle de concorrência, índices, chaves estrangeiras e muito mais.

Desempenho:

Persistência em Arquivo: Em geral, pode oferecer desempenho mais rápido para operações simples de leitura e gravação, especialmente para conjuntos de dados menores, devido à simplicidade da operação de E/S de arquivos.

Persistência em Banco de Dados: Pode oferecer melhor desempenho para consultas complexas e manipulação de grandes volumes de dados devido à otimização interna do SGBD e à capacidade de indexação.

Escalabilidade e Concorrência:

Persistência em Arquivo: Pode ser mais difícil de escalar horizontalmente, pois requer compartilhamento de arquivos ou replicação de arquivos entre sistemas.

Persistência em Banco de Dados: Os bancos de dados geralmente oferecem melhores opções de escalabilidade e suporte a concorrência, permitindo que várias conexões acessem e modifiquem os dados simultaneamente com garantias de consistência e isolamento.

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

O uso de operadores lambda no Java simplificou a impressão dos valores contidos nas entidades ao introduzir uma maneira mais concisa e expressiva de iterar sobre coleções e realizar operações em seus elementos. Isso é especialmente útil ao lidar com coleções de entidades, como listas de objetos.

Antes da introdução de operadores lambda, a iteração sobre uma lista e a impressão dos valores de cada entidade geralmente exigia a utilização de um loop for convencional ou um loop foreach.

Com a introdução de operadores lambda e a API de Streams no Java 8, isso pode ser simplificado usando o método `forEach` da classe `Stream`.