Curso:

Desenvolvimento Full Stack

Campus:

UNIVERSIDADE ESTÁCIO DE SÁ

Disciplina:

BackEnd sem banco não tem

Turma:

2024.3

Aluno:

DARCI RODRIGUES FALCÃO NETO

Missão Prática | Nível 1 |

Mundo

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.**
5. **No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.**

Bom de modo geral eu juntei os dois códigos do procedimento, e resolvi trazer em apenas 1 código tudo junto. segue o mesmo abaixo.

```
package cadastrobd;

import cadastrobd.model.Pessoa;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.util.PessoaFisicaDAO;
import cadastrobd.model.util.PessoaJuridicaDAO;
import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;

public class CadastroBDTeste {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in, "UTF-8")) {
            PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
            PessoaJuridicaDAO pessoaJuridicaDAO = new

            PessoaJuridicaDAO(); int opcao = -1;

            while (opcao != 0) {
```

```
            System.out.println("Escolha 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 - Sair");

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

            switch (opcao) {
                case 1 -> {
                    System.out.println("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica):");
                    String tipo = scanner.nextLine().toUpperCase();
                    System.out.println("Digite o ID:");
                    int id = scanner.nextInt();
                    scanner.nextLine(); // Limpar o buffer
```

```
System.out.println("Digite o nome:");
String nome = scanner.nextLine();
System.out.println("Digite o logradouro:");
String logradouro = scanner.nextLine();
System.out.println("Digite a cidade:");
String cidade = scanner.nextLine();
System.out.println("Digite o estado:");
String estado = scanner.nextLine();
System.out.println("Digite o telefone:");
String telefone = scanner.nextLine();
System.out.println("Digite o email:");
String email = scanner.nextLine();

if ("F".equals(tipo)) {
    System.out.println("Digite o CPF:");
    String cpf = scanner.nextLine();
    PessoaFisica pessoaFisica = new PessoaFisica(
            id, nome, logradouro, cidade, estado, telefone, email, cpf
    );

    try {
        pessoaFisicaDAO.incluir(pessoaFisica);
        System.out.println("Pessoa Física incluída com sucesso!");
    } catch (SQLException e) {
        e.printStackTrace();
    }
                                                    } else if ("J".equals(tipo)) {
                                                    System.out.println("Digite o
                                                    CNPJ:");
                                                    String cnpj =
                                                    scanner.nextLine();
```

```
                    PessoaJuridica pessoaJuridica = new PessoaJuridica(
                            id, nome, logradouro, cidade, estado, telefone, email, cnpj
                    );

                    try {
                        pessoaJuridicaDAO.incluir(pessoaJuridica);
                        System.out.println("Pessoa Jurídica incluída com sucesso!");
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                } else {
                    System.out.println("Opção inválida.");
                }
            }
            case 2 -> {
                System.out.println("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica):");
                String tipo = scanner.nextLine().toUpperCase();
                System.out.println("Digite o ID:");
                int id = scanner.nextInt();
                scanner.nextLine(); // Limpar o buffer

                switch (tipo) {
                    case "F" -> {
                        try {
                            PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
                            if (pessoaFisica != null) {
                                System.out.println("Dados atuais:");
                                pessoaFisica.exibir();

                                System.out.println("Digite o novo telefone:");
                                String novoTelefone = scanner.nextLine();
                                pessoaFisica.setTelefone(novoTelefone);

                                pessoaFisicaDAO.alterar(pessoaFisica);
                                System.out.println("Pessoa Física alterada com sucesso!");
                            } else {
                                System.out.println("Pessoa Física não encontrada.");
                            }
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }
                                                        case "J" -> {
                                                        try {
```

```
                        PessoaJuridica pessoaJuridica =
pessoaJuridicaDAO.getPessoa(id);
                        if (pessoaJuridica != null) {
                            System.out.println("Dados atuais:");
                            pessoaJuridica.exibir();

                            System.out.println("Digite o novo telefone:");
                            String novoTelefone = scanner.nextLine();
                            pessoaJuridica.setTelefone(novoTelefone);

                            pessoaJuridicaDAO.alterar(pessoaJuridica);
                            System.out.println("Pessoa Jurídica alterada com sucesso!");
                        } else {
                            System.out.println("Pessoa Jurídica não encontrada.");
                        }
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
                default -> System.out.println("Opção inválida.");
            }
        }
        case 3 -> {
            System.out.println("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica):");
            String tipo = scanner.nextLine().toUpperCase();
            System.out.println("Digite o ID:");
            int id = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer

            switch (tipo) {
                case "F" -> {
                    try {
                        pessoaFisicaDAO.excluir(id);
                        System.out.println("Pessoa Física excluída com sucesso!");
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
                case "J" -> {
                    try {
                        pessoaJuridicaDAO.excluir(id);
                        System.out.println("Pessoa Jurídica excluída com sucesso!");
                    } catch (SQLException e) {
                                                    e.printStackTrace();
                                                    }
                                                    }
                                                        default ->
```

```
                        System.out.println("Opção inválida.");
                    }
                }
                case 4 -> {
                    System.out.println("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica):");
                    String tipo = scanner.nextLine().toUpperCase();
                    System.out.println("Digite o ID:");
                    int id = scanner.nextInt();
                    scanner.nextLine(); // Limpar o buffer

                    switch (tipo) {
                        case "F" -> {
                            try {
                                PessoaFisica pessoaFisica = pessoaFisicaDAO.getPessoa(id);
                                if (pessoaFisica != null) {
                                    System.out.println("Dados da Pessoa Física:");
                                    pessoaFisica.exibir();
                                } else {
                                    System.out.println("Pessoa Física não encontrada.");
                                }
                            } catch (SQLException e) {
                                e.printStackTrace();
                            }
                        }
                        case "J" -> {
                            try {
                                PessoaJuridica pessoaJuridica =
pessoaJuridicaDAO.getPessoa(id);
                                if (pessoaJuridica != null) {
                                    System.out.println("Dados da Pessoa Jurídica:");
                                    pessoaJuridica.exibir();
                                } else {
                                    System.out.println("Pessoa Jurídica não encontrada.");
                                }
                            } catch (SQLException e) {
                                e.printStackTrace();
                            }
                        }
                        default -> System.out.println("Opção inválida.");
                    }
                }
                case 5 -> {
```

```
                System.out.println("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica):");
                String tipo = scanner.nextLine().toUpperCase();

                switch (tipo) {
                    case "F" -> {
                        try {
                            List<PessoaFisica> pessoasFisicas =
pessoaFisicaDAO.getPessoas();
                            if (!pessoasFisicas.isEmpty()) {
                                System.out.println("Lista de Pessoas Físicas:");
                                for (Pessoa pessoa : pessoasFisicas) {
                                    pessoa.exibir();
                                    System.out.println();
                                }
                            } else {
                                System.out.println("Nenhuma Pessoa Física cadastrada.");
                            }
                        } catch (SQLException e) {
                            e.printStackTrace();
```

```
                    }
                }
                case "J" -> {
                    try {
                        List<PessoaJuridica> pessoasJuridicas =
pessoaJuridicaDAO.getPessoas();
                        if (!pessoasJuridicas.isEmpty()) {
                            System.out.println("Lista de Pessoas Jurídicas:");
                            for (Pessoa pessoa : pessoasJuridicas) {
                                pessoa.exibir();
                                System.out.println();
                            }
                        } else {
                            System.out.println("Nenhuma Pessoa Jurídica cadastrada.");
                        }
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                }
                default -> System.out.println("Opção inválida.");
            }
        }
        case 0 -> System.out.println("Saindo...");
        default -> System.out.println("Opção inválida. Digite um número de 0 a
    5."); }
}
```

```
}
}
}

// Classe Pessoa
package
cadastrobd.model;

public class
Pessoa {
private int id;
private String
nome;
private String
logradouro;
private String
cidade;
private String
estado;
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;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getLogradouro() {
        return logradouro;
    }
```

```
public String
getCidade() {
return cidade;
}

public String
getEstado() {
return estado;
}

public String
getTelefone() {
return telefone;
}

public String
getEmail() {
return email;
}

public void
exibir() {
```

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

}

// Classe PessoaFisica
package cadastrobd.model;

public class PessoaFisica extends Pessoa {
    private String cpf;

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

    public String getCpf() {
        return cpf;
    }

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

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

```
        System.out.println("CPF: " + cpf);
    }

    public void setId(int idPessoa) {
        throw new UnsupportedOperationException("Not supported yet."); // Generated
from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
}

    public void setTelefone(String telefone) {
        this.telefone = telefone;
    }
}
```

```
package cadastrobd.model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

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

    public String getCnpj() {
        return cnpj;
    }

    public void setTelefone(String telefone) {
    this.telefone = telefone;
}

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

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

                                                                    public void
                                                                    setId(int
                                                                    idPessoa) {
                                                                    throw new
```

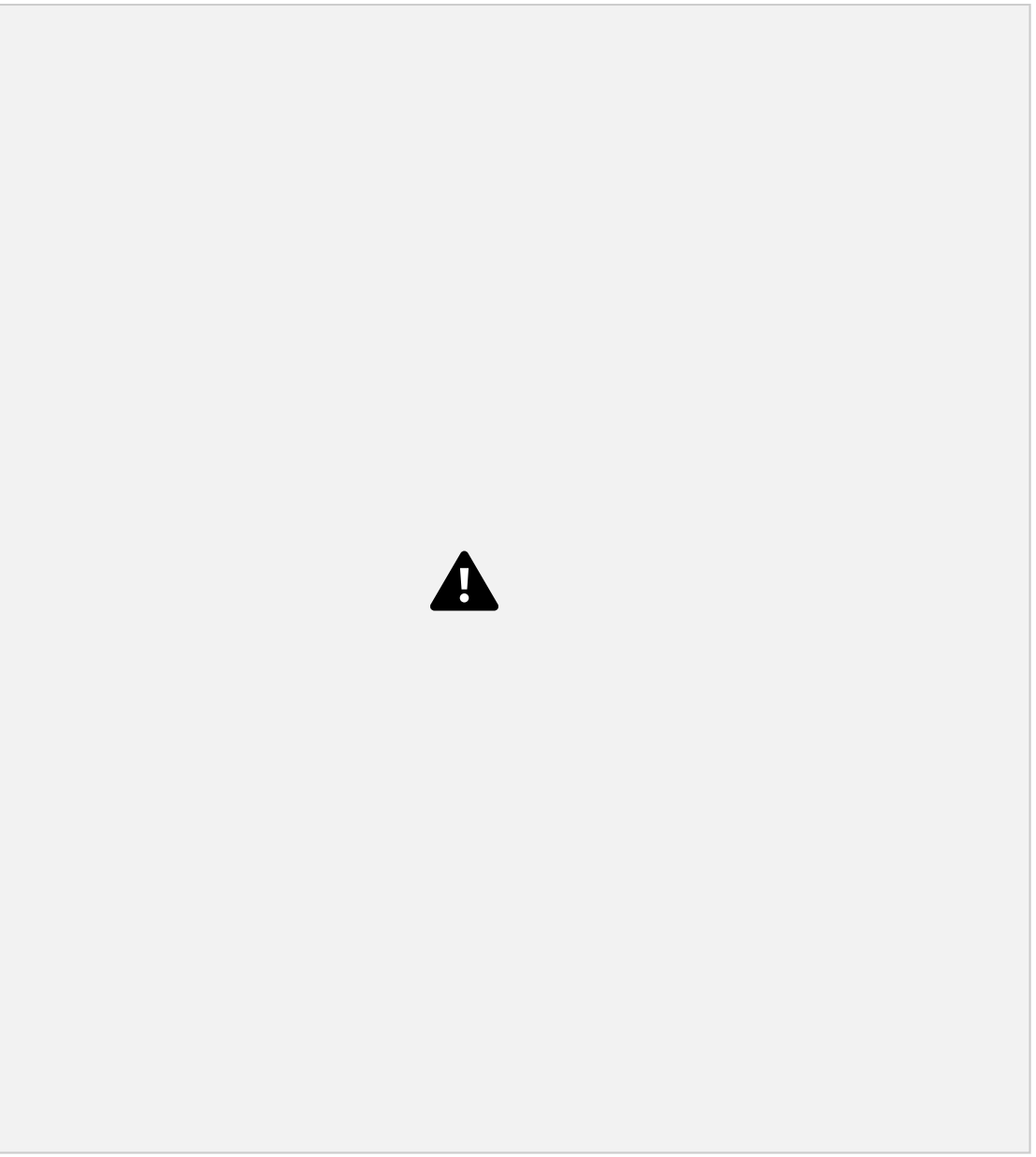
```
UnsupportedOperationException("Not supported yet."); // Generated from
nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody }
}

package cadastrobd.model.util;

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

public class ConectorBD {
    private static final String URL =
"jdbc:sqlserver://localhost:1433;databaseName=SistemaLoja;encrypt=true;trustServerC
ertificate=true";
    private static final String USUARIO = "sa";
    private static final String SENHA = "loja";

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

    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(PreparedStatement ps) throws SQLException
        { if (ps != null) {
            ps.close();
        }
    }

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

public static
void

```
    close(Connection con) throws SQLException {
        if (con != null) {
            con.close();
        }
    }
}

// Classe PessoaFisicaDAO
package cadastrobd.model.util;

import cadastrobd.model.PessoaFisica;

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 PessoaFisicaDAO {

    public void incluir(PessoaFisica pessoaFisica) throws SQLException { String
sqlPessoa = "INSERT INTO Pessoa(nome, logradouro, cidade, estado, telefone,
email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaFisica = "INSERT INTO PessoaFisica(idPessoa, cpf) VALUES
(?, ?)";

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement psPessoa =
connection.prepareStatement(sqlPessoa,
PreparedStatement.RETURN_GENERATED_KEYS);
             PreparedStatement psPessoaFisica =
connection.prepareStatement(sqlPessoaFisica)) {

            psPessoa.setString(1, pessoaFisica.getNome());
            psPessoa.setString(2, pessoaFisica.getLogradouro());
            psPessoa.setString(3, pessoaFisica.getCidade());
            psPessoa.setString(4, pessoaFisica.getEstado());
            psPessoa.setString(5, pessoaFisica.getTelefone());
            psPessoa.setString(6, pessoaFisica.getEmail());

            int rowsAffected = psPessoa.executeUpdate();

            if (rowsAffected == 0) {
                throw new SQLException("Falha ao inserir pessoa.");
            }

                                                        ResultSet
                                                    generatedKeys =
```

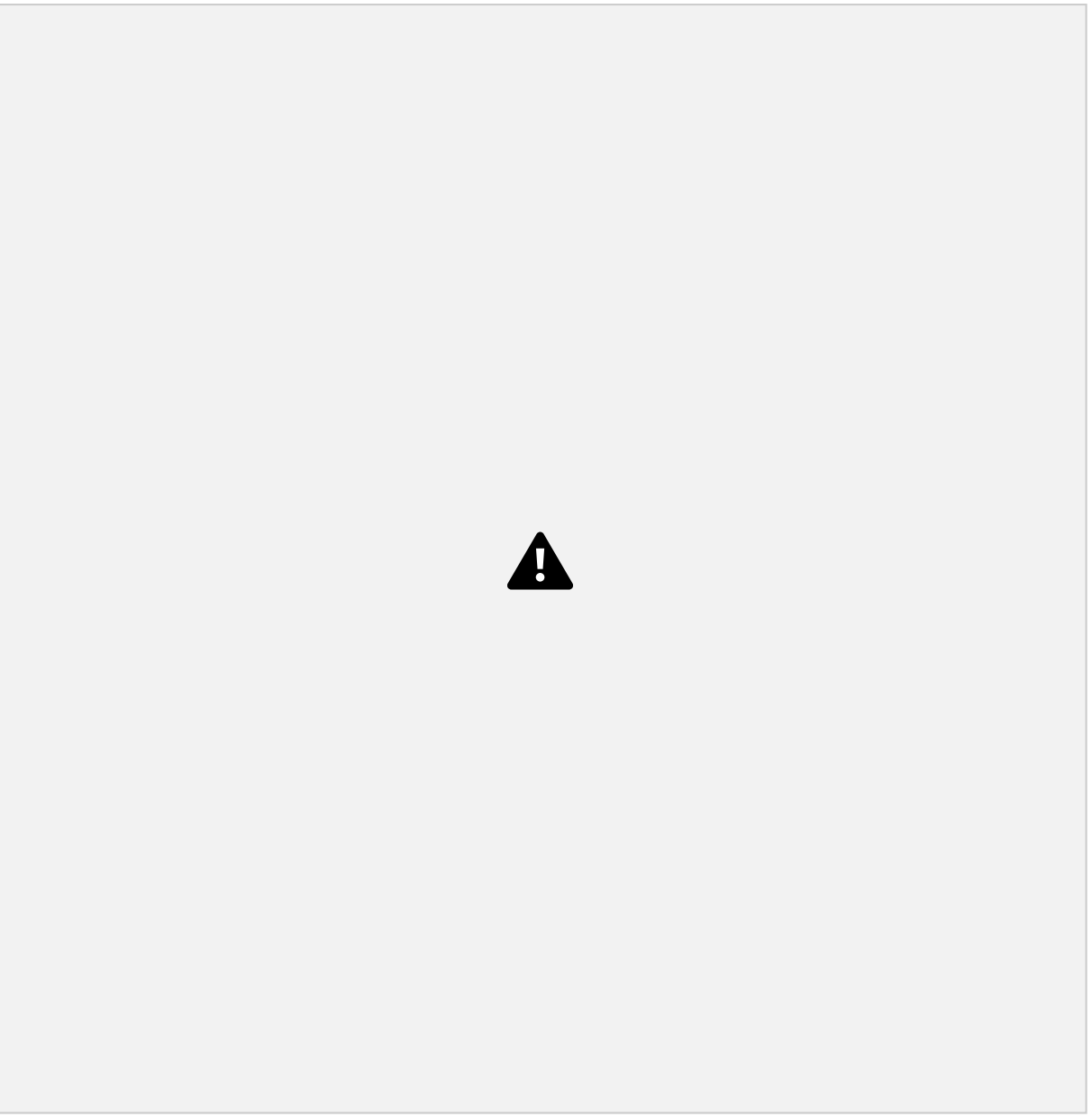
```
                        psPessoa.getGeneratedKeys();
            if (generatedKeys.next()) {
                int idPessoa = generatedKeys.getInt(1);
                pessoaFisica.setId(idPessoa);

                psPessoaFisica.setInt(1, idPessoa);
                psPessoaFisica.setString(2, pessoaFisica.getCpf());
                psPessoaFisica.executeUpdate();
            } else {
                throw new SQLException("Falha ao obter ID gerado.");
            }
        }
    }

    public PessoaFisica getPessoa(int id) throws SQLException { String sql =
"SELECT p.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.idPessoa " +
                    "WHERE p.idPessoa = ?";

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement ps = connection.prepareStatement(sql))

            { ps.setInt(1, id);

            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return extractPessoaFisicaFromResultSet(rs);
                }
            }
        }

        return null;
    }

    public List<PessoaFisica> getPessoas() throws SQLException { String sql =
"SELECT p.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.idPessoa";

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

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement ps = connection.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {

                                                        while (rs.next()) {
```

```
            pessoas.add(extractPessoaFisicaFromResultSet(rs));
        }
    }

    return pessoas;
  }

  public void alterar(PessoaFisica pessoaFisica) throws SQLException { String
sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado =
?, telefone = ?, email = ? WHERE idPessoa = ?";
    String sqlPessoaFisica = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa =
?";

    try (Connection connection = ConectorBD.getConnection();
        PreparedStatement psPessoa =
        connection.prepareStatement(sqlPessoa); PreparedStatement
        psPessoaFisica =
connection.prepareStatement(sqlPessoaFisica)) {

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

        psPessoaFisica.setString(1, pessoaFisica.getCpf());
        psPessoaFisica.setInt(2, pessoaFisica.getId());

        psPessoa.executeUpdate();
        psPessoaFisica.executeUpdate();
    }
  }

  public void excluir(int id) throws SQLException {
    String sqlPessoaFisica = "DELETE FROM PessoaFisica WHERE idPessoa =
    ?"; String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";

    try (Connection connection = ConectorBD.getConnection();
        PreparedStatement psPessoaFisica =
connection.prepareStatement(sqlPessoaFisica);
        PreparedStatement psPessoa = connection.prepareStatement(sqlPessoa))

        { psPessoaFisica.setInt(1, id);
```



```
            psPessoaFisica.executeUpdate();

            psPessoa.setInt(1, id);
            psPessoa.executeUpdate();
        }
    }

    private PessoaFisica extractPessoaFisicaFromResultSet(ResultSet rs) throws
SQLException {
        int id = rs.getInt("idPessoa");
        String nome = rs.getString("nome");
        String logradouro = rs.getString("logradouro");
        String cidade = rs.getString("cidade");
        String estado = rs.getString("estado");
        String telefone = rs.getString("telefone");
        String email = rs.getString("email");
        String cpf = rs.getString("cpf");

        return new PessoaFisica(id, nome, logradouro, cidade, estado, telefone,
email, cpf);
```

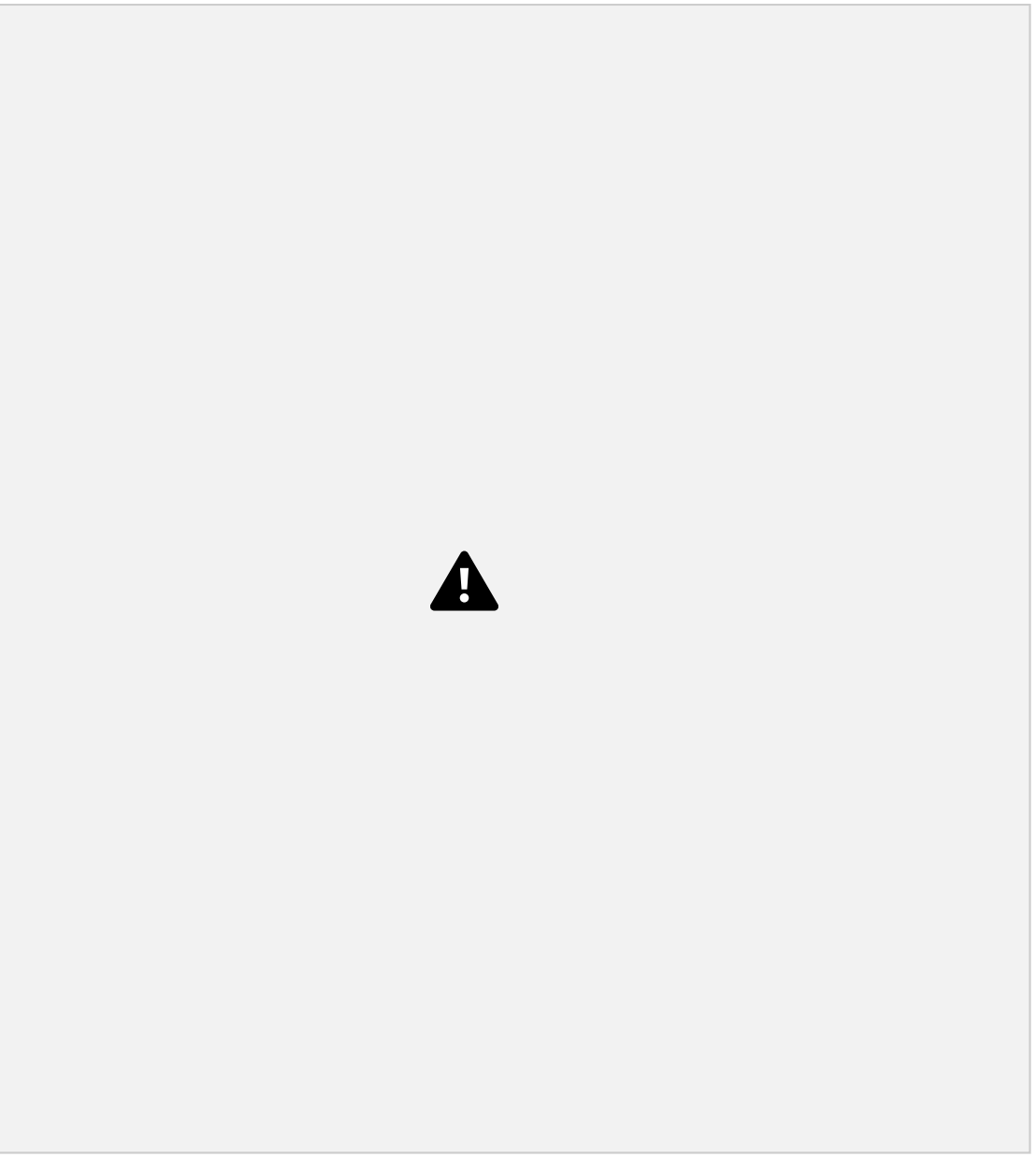
```
    }


}

// Classe PessoaJuridicaDAO
package cadastrobd.model.util;

import cadastrobd.model.PessoaJuridica;

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 void incluir(PessoaJuridica pessoaJuridica) throws SQLException { String
sqlPessoa = "INSERT INTO Pessoa(nome, logradouro, cidade, estado, telefone,
email) VALUES (?, ?, ?, ?, ?, ?)";
        String sqlPessoaJuridica = "INSERT INTO PessoaJuridica(idPessoa, cnpj)
VALUES (?, ?)";

        try (Connection connection = ConectorBD.getConnection();
```



```
            PreparedStatement psPessoa =
connection.prepareStatement(sqlPessoa,
PreparedStatement.RETURN_GENERATED_KEYS);
            PreparedStatement psPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {

            psPessoa.setString(1, pessoaJuridica.getNome());
            psPessoa.setString(2, pessoaJuridica.getLogradouro());
            psPessoa.setString(3, pessoaJuridica.getCidade());
            psPessoa.setString(4, pessoaJuridica.getEstado());
            psPessoa.setString(5, pessoaJuridica.getTelefone());
            psPessoa.setString(6, pessoaJuridica.getEmail());

            int rowsAffected = psPessoa.executeUpdate();

            if (rowsAffected == 0) {
                throw new SQLException("Falha ao inserir pessoa.");
            }

            ResultSet generatedKeys = psPessoa.getGeneratedKeys();
```

```
            if (generatedKeys.next()) {
                int idPessoa = generatedKeys.getInt(1);
                pessoaJuridica.setId(idPessoa);

                psPessoaJuridica.setInt(1, idPessoa);
                psPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
                psPessoaJuridica.executeUpdate();
            } else {
                throw new SQLException("Falha ao obter ID gerado.");
            }
        }
    }

    public PessoaJuridica getPessoa(int id) throws SQLException { String sql =
"SELECT p.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.idPessoa " +
                    "WHERE p.idPessoa = ?";

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement ps = connection.prepareStatement(sql)) {

            ps.setInt(1, id);

            try (ResultSet rs = ps.executeQuery()) {
                if (rs.next()) {
                    return extractPessoaJuridicaFromResultSet(rs);
                }
            }
        }

        return null;
    }

    public
```

```
List<PessoaJuridica> getPessoas() throws SQLException { String sql = "SELECT
p.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.idPessoa";

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

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement ps = connection.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {

            while (rs.next()) {
                pessoas.add(extractPessoaJuridicaFromResultSet(rs));
            }
        }

        return pessoas;
    }

    public void alterar(PessoaJuridica pessoaJuridica) throws SQLException { String
sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado =
?, telefone = ?, email = ? WHERE idPessoa = ?";
        String sqlPessoaJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE
idPessoa = ?";

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement psPessoa =
             connection.prepareStatement(sqlPessoa); PreparedStatement
             psPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica)) {

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

            psPessoaJuridica.setString(1, pessoaJuridica.getCnpj());
```



```
            psPessoaJuridica.setInt(2, pessoaJuridica.getId());

            psPessoa.executeUpdate();
            psPessoaJuridica.executeUpdate();
        }
    }

    public void excluir(int id) throws SQLException {
        String sqlPessoaJuridica = "DELETE FROM PessoaJuridica WHERE idPessoa =
?";
        String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";

        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement psPessoaJuridica =
connection.prepareStatement(sqlPessoaJuridica);
             PreparedStatement psPessoa = connection.prepareStatement(sqlPessoa)) {

            psPessoaJuridica.setInt(1, id);
            psPessoaJuridica.executeUpdate();
```

```
            psPessoa.setInt(1, id);
            psPessoa.executeUpdate();
        }
    }

    private PessoaJuridica extractPessoaJuridicaFromResultSet(ResultSet rs) throws
SQLException {
        int id = rs.getInt("idPessoa");
        String nome = rs.getString("nome");
        String logradouro = rs.getString("logradouro");
        String cidade = rs.getString("cidade");
        String estado = rs.getString("estado");
        String telefone = rs.getString("telefone");
        String email = rs.getString("email");
        String cnpj = rs.getString("cnpj");

        return new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone,
email, cnpj);
    }
}



// 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 sequenceName) {
        try (Connection connection = ConectorBD.getConnection();
             PreparedStatement ps = connection.prepareStatement("SELECT NEXTVAL('" +
sequenceName + "')");
             ResultSet rs = ps.executeQuery()) {

            if (rs.next()) {
                return rs.getInt(1);
            }
        } catch (SQLException e) {
        }
        return 0;
    }
}
```

Análise e

Conclusão:

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

A persistência em arquivo e a persistência em banco de dados são duas abordagens diferentes para armazenar e recuperar dados em um sistema de software. Cada uma delas possui características distintas e é mais adequada para cenários específicos. Vamos explorar as diferenças entre elas:

1. **Armazenamento de Dados:**
   - **Persistência em Arquivo:** Os dados são armazenados em arquivos no sistema de arquivos do computador ou em algum sistema de armazenamento externo, como um servidor de arquivos. Cada arquivo geralmente contém uma representação dos dados, e os dados são lidos ou gravados diretamente nesses arquivos.

   - **Persistência em Banco de Dados:** Os dados são armazenados em um sistema de gerenciamento de banco de dados (SGBD), como MySQL, PostgreSQL, MongoDB, entre outros. Os dados são organizados em tabelas, documentos ou outros tipos de estruturas de dados definidas pelo SGBD.

2. **Recuperação e Manipulação de Dados:**
   - **Persistência em Arquivo:** A recuperação e manipulação de dados geralmente envolvem a leitura e gravação direta nos arquivos. Isso pode exigir código personalizado para interpretar e modificar os dados nos arquivos.

   - **Persistência em Banco de Dados:** O acesso aos dados é feito por meio de consultas SQL (em sistemas de bancos de dados relacionais) ou consultas específicas da API (em bancos de dados NoSQL). O SGBD cuida da tradução das consultas em operações de leitura e gravação nos dados.

3. **Consistência e Integridade dos Dados:**
   - **Persistência em Arquivo:** A responsabilidade pela consistência e integridade dos dados recai em grande parte sobre o desenvolvedor. Não há mecanismos automáticos de garantia de consistência.

   - **Persistência em Banco de Dados:** Os SGBDs oferecem mecanismos para garantir a integridade referencial, transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e outras características que ajudam a manter a integridade dos dados.

4. **Concorrência e Escalabilidade:**
   - **Persistência em Arquivo:** O controle de concorrência pode ser complexo de implementar em sistemas de arquivos simples. A escalabilidade pode ser limitada, especialmente em sistemas de arquivos locais.

-

Persistência em Banco de Dados: Os SGBDs geralmente incluem recursos para controle de concorrência e escalabilidade, permitindo que várias conexões de clientes acessem os dados simultaneamente.

5. **Segurança:**
   - **Persistência em Arquivo:** A segurança depende da configuração do sistema de arquivos e das permissões de acesso. Pode ser mais difícil de controlar e auditar o acesso aos dados.

   - **Persistência em Banco de Dados:** Os SGBDs oferecem recursos avançados de segurança, como autenticação, autorização e auditoria, para proteger os dados contra acessos não autorizados.

6. **Backup e Recuperação:**
   - **Persistência em Arquivo:** A responsabilidade pelo backup e recuperação de dados recai em grande parte sobre o desenvolvedor, o que pode ser propenso a erros.

   - **Persistência em Banco de Dados:** Os SGBDs geralmente oferecem facilidades para backup e recuperação de dados, simplificando o processo.

Em resumo, a escolha entre persistência em arquivo e persistência em banco de dados depende das necessidades específicas do seu aplicativo. Os bancos de dados são mais adequados para aplicativos que requerem consistência, segurança, escalabilidade e recursos avançados de consulta. A persistência em arquivo pode ser mais simples e adequada para pequenos aplicativos ou cenários em que a estrutura dos dados é simples e os requisitos de segurança e escalabilidade são menos críticos.

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 significativamente a impressão de valores contidos em entidades (objetos) ao permitir que você escreva código mais conciso e legível para tarefas como a iteração em coleções de objetos. A introdução de expressões lambda no Java a partir da versão 8 trouxe melhorias significativas para lidar com esse tipo de operação.

Antes do Java 8, para imprimir valores contidos em entidades, você frequentemente usaria loops, como o `for` ou o `foreach`, para iterar sobre as coleções de objetos e, em seguida, usaria uma estrutura de código mais elaborada para acessar e imprimir os valores. Isso frequentemente resultava em código mais verboso e menos legível.

Com a introdução dos operadores lambda e a API Streams no Java 8, você pode realizar operações de iteração e transformação de maneira mais elegante e concisa.

Aqui está um

exemplo simplificado de como os operadores lambda simplificaram a impressão de valores em coleções de entidades:

Antes do Java 8:
```java
List<Entidade> entidades = // uma lista de objetos Entidade
for (Entidade entidade : entidades) {
    System.out.println(entidade.getValor());
}
```

Com operadores lambda (Java 8 em diante):
```java
List<Entidade> entidades = // uma lista de objetos Entidade
entidades.forEach(entidade ->
System.out.println(entidade.getValor())); ```

Nesse exemplo, o uso de `forEach` com uma expressão lambda permite que você itere sobre a lista de entidades e imprima o valor de cada entidade de forma muito mais concisa e legível.

Além disso, você pode usar operações de streaming para realizar filtragem, mapeamento e outras transformações de maneira mais simples e eficiente. Por exemplo, você pode filtrar entidades com base em algum critério e imprimir apenas os valores que atendem a esse critério usando operações de stream:

```java
List<Entidade> entidades = // uma lista de objetos Entidade
entidades.stream()
        .filter(entidade -> entidade.getValor() > 10)
        .forEach(entidade -> System.out.println(entidade.getValor()));
```

Em resumo, os operadores lambda no Java, juntamente com a API Streams, simplificaram a impressão de valores em entidades, bem como muitas outras operações de manipulação de coleções, tornando o código mais legível e expressivo. Isso é especialmente útil em cenários onde a manipulação de dados é uma parte fundamental do seu programa.

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

Em Java, os métodos acionados diretamente pelo método
```

`main` sem o uso de um objeto precisam ser marcados como `static` porque o método `main` também é `static`. Isso está relacionado à natureza da execução de um programa Java.

Aqui estão algumas razões pelas quais o método `main` e outros métodos acionados diretamente por ele devem ser `static`:

1. **Ponto de Entrada:** O método `main` é o ponto de entrada para a execução de um programa Java. Ele é chamado pelo ambiente de execução Java (a máquina virtual Java - JVM) sem a criação de uma instância da classe que contém o método `main`. Isso significa que o método `main` deve ser acessível mesmo antes que qualquer objeto da classe seja criado.

2. **Escopo Global:** Quando um método é `static`, ele pertence à classe em vez de pertencer a instâncias específicas dessa classe. Isso torna o método acessível globalmente, permitindo que seja chamado sem a necessidade de criar uma instância da classe. Isso é fundamental para a chamada do método `main` porque, como mencionado anteriormente, ele é o ponto de entrada para o programa.

3. **Convenção:** O uso de `static` em métodos que são chamados diretamente pelo `main` é uma convenção em Java. Essa convenção facilita a identificação dos métodos que servem como pontos de partida para a execução do programa e ajuda os desenvolvedores a entenderem que esses métodos não dependem do estado de instâncias da classe.

4. **Eficiência:** Métodos `static` não têm acesso ao estado de instância e, portanto, não podem acessar campos de instância diretamente. Isso elimina a necessidade de criar uma instância da classe apenas para chamar esses métodos, o que pode ser mais eficiente em termos de recursos.

Portanto, para que um método seja invocado diretamente a partir do método `main`, ele deve ser `static`. A assinatura do método `main` padrão é a seguinte:

```java
public static void main(String[] args)
```

Isso permite que a JVM chame o método `main` sem criar uma instância da classe que o contém, tornando-o adequado como ponto de entrada para a execução do programa Java.