



## **RPG0016 - BackEnd sem banco não tem**

**Jéssica Maria de Carvalho Matrícula: 202209187939**

**POLO JARDIM SÃO BERNARDO - SÃO PAULO - SP**

**Nível 2: Vamos Manter as Informações? – 9003 – 3º**

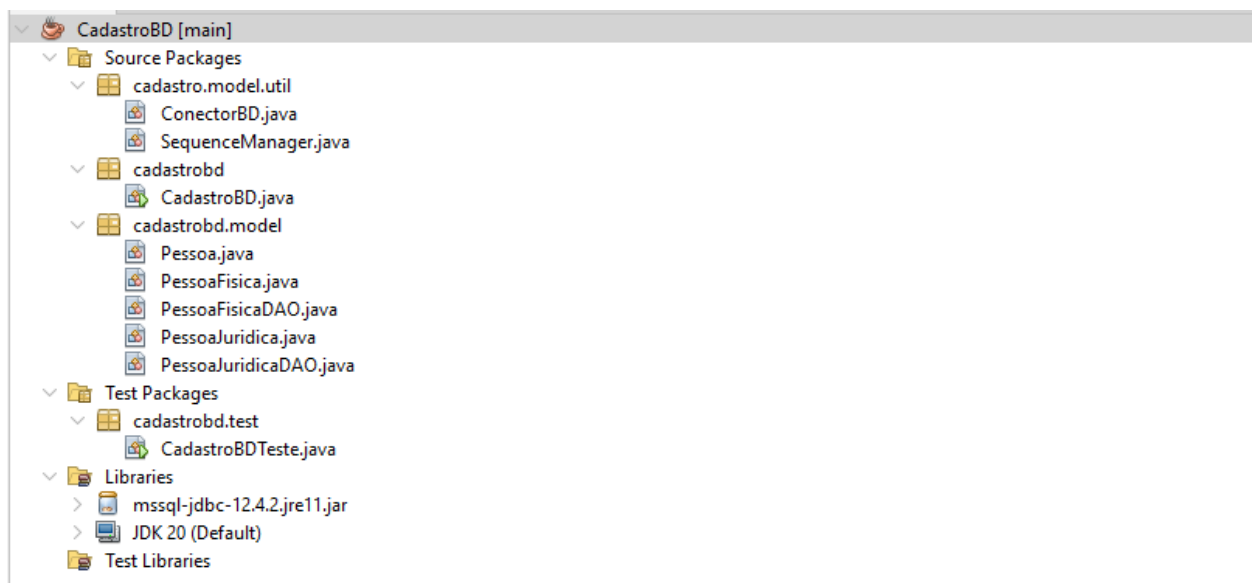
**Endereço do Repositório GIT:**

**<https://github.com/Jessicac30/CadastroBD/tree/main/CadastroBD>**

### **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
6. SQL Server na persistência de dados.

## 1º Procedimento | Mapeamento Objeto-Relacional e DAO



## Arquivo ConectorBD.java

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

```
import java.sql.*;
```

```
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(Connection conn, String sql) throws  
    SQLException {  
        return conn.prepareStatement(sql);  
    }
```

```
    public static ResultSet getSelect(Connection conn, String sql) throws SQLException {  
        Statement stmt = conn.createStatement();  
        return stmt.executeQuery(sql);  
    }
```

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

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

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

## Arquivo SequenceManager.java

```
package cadastro.model.util;

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

public class SequenceManager {

    public static int getNextValue(Connection conn, String sequenceName) throws
    SQLException {
        String sql = "SELECT NEXT VALUE FOR " + sequenceName;
        try (PreparedStatement pstmt = ConectorBD.getPrepared(conn, sql);
            ResultSet rs = pstmt.executeQuery()) {

            if (rs.next()) {
                return rs.getInt(1);
            } else {
                throw new SQLException("Não foi possível obter o próximo valor da sequência.");
            }
        }
    }
}
```

## Arquivo Pessoa.java

```
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;  
        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("Id: " + getId());  
    System.out.println("Nome: " + getNome());  
    System.out.println("Logradouro: " + getLogradouro());  
    System.out.println("Cidade: " + getCidade());  
    System.out.println("Estado: " + getEstado());  
    System.out.println("Telefone: " + getTelefone());  
    System.out.println("E-mail: " + getEmail());  
}  
}
```



## Arquivo PessoaFisica.java

```
package cadastrbd.model;
```

```
public class PessoaFisica extends Pessoa {
```

```
    private String cpf;
```

```
    public PessoaFisica(String joão, String rua_12_casa_3, String riacho_do_Sul, String pa, String  
string, String joaoriachocom, String string1) {
```

```
        super();
```

```
    }
```

```
    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 int getId() {
```

```
        return super.getId();
```

```
    }
```

```
    @Override
```

```
    public void setId(int id) {
```

```
        super.setId(id);
```

```
    }
```

```
    @Override
```

```
public String getNome() {  
    return super.getNome();  
}
```

```
@Override  
public void setNome(String nome) {  
    super.setNome(nome);  
}
```

```
@Override  
public String getLogradouro() {  
    return super.getLogradouro();  
}
```

```
@Override  
public void setLogradouro(String logradouro) {  
    super.setLogradouro(logradouro);  
}
```

```
@Override  
public String getCidade() {  
    return super.getCidade();  
}
```

```
@Override  
public void setCidade(String cidade) {  
    super.setCidade(cidade);  
}
```

```
@Override  
public String getEstado() {  
    return super.getEstado();  
}
```

```
@Override  
public void setEstado(String estado) {  
    super.setEstado(estado);  
}
```

```
}
```

```
@Override  
public String getTelefone() {  
    return super.getTelefone();  
}
```

```
@Override  
public void setTelefone(String telefone) {  
    super.setTelefone(telefone);  
}
```

```
@Override  
public String getEmail() {  
    return super.getEmail();  
}
```

```
@Override  
public void setEmail(String email) {  
    super.setEmail(email);  
}
```

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

```
@Override  
public String toString() {  
    return "PessoaFisica{" +  
        "id=" + getId() +  
        ", nome=" + getNome() + "\" +  
        ", logradouro=" + getLogradouro() + "\" +  
        ", cidade=" + getCidade() + "\" +  
        ", estado=" + getEstado() + "\" +  
        ", telefone=" + getTelefone() + "\" +
```

```
    ", email=" + getEmail() + "\" +  
    ", cpf=" + cpf + "\" +  
    '}'  
  }  
}
```

## Arquivo PessoaFisicaDAO.java

```
package cadastrbd.model;

import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaDAO {

    public PessoaFisica getPessoa(int id) {
        PessoaFisica pessoaFisica = null;

        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.Pessoa_idPessoa WHERE Pessoa.idPessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, id);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    pessoaFisica = new PessoaFisica(
                        rs.getInt("idPessoa"),
                        rs.getString("nome"),
                        rs.getString("logradouro"),
                        rs.getString("cidade"),
                        rs.getString("estado"),
                        rs.getString("telefone"),
                        rs.getString("email"),
                        rs.getString("CPF")
                    );
                }
            }
        } catch (SQLException e) {
```

```

    }
    return pessoaFisica;
}

```

```

public List<PessoaFisica> getPessoas() {
    List<PessoaFisica> pessoasFisicas = new ArrayList<>();
    String sql = "SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.Pessoa_idPessoa";

```

```

    try (Connection conn = ConectorBD.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {

```

```

        while (rs.next()) {
            PessoaFisica pessoaFisica = new PessoaFisica(
                rs.getInt("idPessoa"),
                rs.getString("nome"),
                rs.getString("logradouro"),
                rs.getString("cidade"),
                rs.getString("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("CPF")
            );
            pessoasFisicas.add(pessoaFisica);
        }
    } catch (SQLException e) {
    }
    return pessoasFisicas;
}

```

```

public boolean incluir(PessoaFisica pessoaFisica) {
    Connection conn = null;
    PreparedStatement pstmtPessoa = null;
    PreparedStatement pstmtPessoaFisica = null;

```

```

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

    int idPessoa = SequenceManager.getNextValue(conn, "Seq_idPessoa");

    String sqlPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado,
telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
    pstmtPessoa = conn.prepareStatement(sqlPessoa);
    pstmtPessoa.setInt(1, idPessoa);
    pstmtPessoa.setString(2, pessoaFisica.getNome());
    pstmtPessoa.setString(3, pessoaFisica.getLogradouro());
    pstmtPessoa.setString(4, pessoaFisica.getCidade());
    pstmtPessoa.setString(5, pessoaFisica.getEstado());
    pstmtPessoa.setString(6, pessoaFisica.getTelefone());
    pstmtPessoa.setString(7, pessoaFisica.getEmail());
    pstmtPessoa.executeUpdate();

    String sqlPessoaFisica = "INSERT INTO PessoaFisica (Pessoa_idPessoa, CPF)
VALUES (?, ?)";
    pstmtPessoaFisica = conn.prepareStatement(sqlPessoaFisica);
    pstmtPessoaFisica.setInt(1, idPessoa);
    pstmtPessoaFisica.setString(2, pessoaFisica.getCpf());
    pstmtPessoaFisica.executeUpdate();

    conn.commit();
return true;
} catch (SQLException e) {
    e.printStackTrace();
    return false;

} finally {
    Statement generatedKeys = null;
    ConectorBD.close(generatedKeys);
    ConectorBD.close(pstmtPessoa);
    ConectorBD.close(pstmtPessoaFisica);
    ConectorBD.close(conn);
}

```

```
}  
}
```

```
public boolean alterar(PessoaFisica pessoaFisica) {  
    String sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado =  
    ?, telefone = ?, email = ? WHERE idPessoa = ?";  
    String sqlPessoaFisica = "UPDATE PessoaFisica SET CPF = ? WHERE Pessoa_idPessoa  
    = ?";
```

```
    try (Connection conn = ConectorBD.getConnection();  
        PreparedStatement pstmtPessoa = conn.prepareStatement(sqlPessoa);  
        PreparedStatement pstmtPessoaFisica = conn.prepareStatement(sqlPessoaFisica)) {
```

```
        conn.setAutoCommit(false);
```

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

```
        pstmtPessoaFisica.setString(1, pessoaFisica.getCpf());  
        pstmtPessoaFisica.setInt(2, pessoaFisica.getId());  
        pstmtPessoaFisica.executeUpdate();
```

```
        conn.commit();  
        return true;  
    } catch (SQLException e) {  
        e.printStackTrace();  
        return false;  
    }  
}
```



```

public boolean excluir(int id) {
    String sqlDeleteMovimentos = "DELETE FROM Movimento WHERE Pessoa_idPessoa =
?";
    String sqlDeletePessoaFisica = "DELETE FROM PessoaFisica WHERE Pessoa_idPessoa =
?";
    String sqlDeletePessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";

    try (Connection conn = ConectorBD.getConnection());
        PreparedStatement pstmtDeleteMovimentos =
conn.prepareStatement(sqlDeleteMovimentos);
        PreparedStatement pstmtDeletePessoaFisica =
conn.prepareStatement(sqlDeletePessoaFisica);
        PreparedStatement pstmtDeletePessoa = conn.prepareStatement(sqlDeletePessoa)) {

        conn.setAutoCommit(false);

        pstmtDeleteMovimentos.setInt(1, id);
        pstmtDeleteMovimentos.executeUpdate();

        pstmtDeletePessoaFisica.setInt(1, id);
        pstmtDeletePessoaFisica.executeUpdate();

        pstmtDeletePessoa.setInt(1, id);
        pstmtDeletePessoa.executeUpdate();

        conn.commit();
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    } finally {
    }
}

public PessoaFisica getPessoaPorCpf(String cpf) {
    PessoaFisica pessoaFisica = null;

```

```
String sql = "SELECT * FROM Pessoa INNER JOIN PessoaFisica ON Pessoa.idPessoa =  
PessoaFisica.Pessoa_idPessoa WHERE PessoaFisica.CPF = ?";
```

```
try (Connection conn = ConectorBD.getConnection();  
    PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
    pstmt.setString(1, cpf);  
    ResultSet rs = pstmt.executeQuery();  
  
    if (rs.next()) {  
        pessoaFisica = new PessoaFisica(  
            rs.getInt("idPessoa"),  
            rs.getString("nome"),  
            rs.getString("logradouro"),  
            rs.getString("cidade"),  
            rs.getString("estado"),  
            rs.getString("telefone"),  
            rs.getString("email"),  
            rs.getString("CPF")  
        );  
    }  
    rs.close();  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
return pessoaFisica;  
}  
}
```

## Arquivo PessoaJuridica.java

```
package cadastrobd.model;
```

```
public class PessoaJuridica extends Pessoa {  
    private String cnpj;
```

```
  
    public PessoaJuridica() {  
        super();  
    }
```

```
  
    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 int getId() {  
        return super.getId();  
    }
```

```
  
    @Override  
    public void setId(int id) {  
        super.setId(id);  
    }
```

```
  
    @Override
```

```
public String getNome() {  
    return super.getNome();  
}
```

```
@Override  
public void setNome(String nome) {  
    super.setNome(nome);  
}
```

```
@Override  
public String getLogradouro() {  
    return super.getLogradouro();  
}
```

```
@Override  
public void setLogradouro(String logradouro) {  
    super.setLogradouro(logradouro);  
}
```

```
@Override  
public String getCidade() {  
    return super.getCidade();  
}
```

```
@Override  
public void setCidade(String cidade) {  
    super.setCidade(cidade);  
}
```

```
@Override  
public String getEstado() {  
    return super.getEstado();  
}
```

```
@Override  
public void setEstado(String estado) {  
    super.setEstado(estado);  
}
```

```
}
```

```
@Override  
public String getTelefone() {  
    return super.getTelefone();  
}
```

```
@Override  
public void setTelefone(String telefone) {  
    super.setTelefone(telefone);  
}
```

```
@Override  
public String getEmail() {  
    return super.getEmail();  
}
```

```
@Override  
public void setEmail(String email) {  
    super.setEmail(email);  
}
```

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

```
@Override  
public String toString() {  
    return "PessoaJuridica{" +  
        "id=" + getId() +  
        ", nome=" + getNome() + "\" +  
        ", logradouro=" + getLogradouro() + "\" +  
        ", cidade=" + getCidade() + "\" +  
        ", estado=" + getEstado() + "\" +  
        ", telefone=" + getTelefone() + "\" +
```

```
    ", email=" + getEmail() + "\" +  
    ", cnpj=" + cnpj + "\" +  
    '}'  
  }  
}
```

## Arquivo PessoaJuridicaDAO.java

```
package cadastrbd.model;

import cadastro.model.util.ConectorBD;
import cadastro.model.util.SequenceManager;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaDAO {

    public PessoaJuridica getPessoa(int id) {
        PessoaJuridica pessoaJuridica = null;

        String sql = "SELECT * FROM Pessoa INNER JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.Pessoa_idPessoa WHERE Pessoa.idPessoa = ?";

        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                pessoaJuridica = new PessoaJuridica(
                    rs.getInt("idPessoa"),
                    rs.getString("nome"),
                    rs.getString("logradouro"),
                    rs.getString("cidade"),
                    rs.getString("estado"),
                    rs.getString("telefone"),
                    rs.getString("email"),
                    rs.getString("CNPJ")
                );
            }
            rs.close();
        }
    }
}
```

```

    } catch (SQLException e) {
        e.printStackTrace();
    }
    return pessoaJuridica;
}

public List<PessoaJuridica> getPessoas() {
    List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

    String sql = "SELECT * FROM Pessoa INNER JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.Pessoa_idPessoa";

    try (Connection conn = ConectorBD.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            PessoaJuridica pessoaJuridica = new PessoaJuridica(
                rs.getInt("idPessoa"),
                rs.getString("nome"),
                rs.getString("logradouro"),
                rs.getString("cidade"),
                rs.getString("estado"),
                rs.getString("telefone"),
                rs.getString("email"),
                rs.getString("CNPJ")
            );
            pessoasJuridicas.add(pessoaJuridica);
        }
    } catch (SQLException e) {
    }

    return pessoasJuridicas;
}

public boolean incluir(PessoaJuridica pessoaJuridica) {
    Connection conn = null;
    PreparedStatement pstmtPessoa = null;
    PreparedStatement pstmtPessoaJuridica = null;

```



```

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

    int idPessoa = SequenceManager.getNextValue(conn, "NomeDaSequenciaPessoa");

    String sqlPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, estado,
telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
    pstmtPessoa = conn.prepareStatement(sqlPessoa);
    pstmtPessoa.setInt(1, idPessoa);
    pstmtPessoa.setString(2, pessoaJuridica.getNome());
    pstmtPessoa.setString(3, pessoaJuridica.getLogradouro());
    pstmtPessoa.setString(4, pessoaJuridica.getCidade());
    pstmtPessoa.setString(5, pessoaJuridica.getEstado());
    pstmtPessoa.setString(6, pessoaJuridica.getTelefone());
    pstmtPessoa.setString(7, pessoaJuridica.getEmail());
    pstmtPessoa.executeUpdate();

    String sqlPessoaJuridica = "INSERT INTO PessoaJuridica (Pessoa_idPessoa, CNPJ)
VALUES (?, ?)";
    pstmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica);
    pstmtPessoaJuridica.setInt(1, idPessoa);
    pstmtPessoaJuridica.setString(2, pessoaJuridica.getCnpj());
    pstmtPessoaJuridica.executeUpdate();

    conn.commit();
    return true;
} catch (SQLException e) {
    return false;
}
}

public boolean alterar(PessoaJuridica pessoaJuridica) {

```

```
String sqlPessoa = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, estado = ?,  
telefone = ?, email = ? WHERE idPessoa = ?";
```

```
String sqlPessoaJuridica = "UPDATE PessoaJuridica SET CNPJ = ? WHERE  
Pessoa_idPessoa = ?";
```

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

```
    PreparedStatement pstmtPessoa = conn.prepareStatement(sqlPessoa);
```

```
    PreparedStatement pstmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica)) {
```

```
    conn.setAutoCommit(false);
```

```
    pstmtPessoa.setString(1, pessoaJuridica.getNome());
```

```
    pstmtPessoa.setString(2, pessoaJuridica.getLogradouro());
```

```
    pstmtPessoa.setString(3, pessoaJuridica.getCidade());
```

```
    pstmtPessoa.setString(4, pessoaJuridica.getEstado());
```

```
    pstmtPessoa.setString(5, pessoaJuridica.getTelefone());
```

```
    pstmtPessoa.setString(6, pessoaJuridica.getEmail());
```

```
    pstmtPessoa.setInt(7, pessoaJuridica.getId());
```

```
    pstmtPessoa.executeUpdate();
```

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

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

```
    pstmtPessoaJuridica.executeUpdate();
```

```
    conn.commit();
```

```
    return true;
```

```
} catch (SQLException e) {
```

```
    return false;
```

```
}
```

```
}
```

```
public boolean excluir(int id) {
```

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

```
    String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
```

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

```
    PreparedStatement pstmtPessoaJuridica = conn.prepareStatement(sqlPessoaJuridica);
```

```
PreparedStatement pstmtPessoa = conn.prepareStatement(sqlPessoa)) {  
  
    conn.setAutoCommit(false);  
  
    pstmtPessoaJuridica.setInt(1, id);  
    pstmtPessoaJuridica.executeUpdate();  
  
    pstmtPessoa.setInt(1, id);  
    pstmtPessoa.executeUpdate();  
  
    conn.commit();  
    return true;  
} catch (SQLException e) {  
    return false;  
}  
}  
}
```

## **CadastroBDTeste.java**

```
package cadastrobd.test;

import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.util.List;

public class CadastroBDTeste {

    public static void main(String[] args) {
        PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();

        PessoaFisica pessoaExistente = pessoaFisicaDAO.getPessoaPorCpf("11111111111");
        if (pessoaExistente != null) {
            pessoaFisicaDAO.excluir(pessoaExistente.getId());
        }

        PessoaFisica novaPessoa = new PessoaFisica(0, "João", "Rua 12, casa 3, Quintanda",
            "Riacho do Sul", "PA", "1111-1111", "joao@riacho.com", "11111111111");
        boolean sucessoInclusao = pessoaFisicaDAO.incluir(novaPessoa);

        novaPessoa.setNome("Pedro");
        novaPessoa.setLogradouro("Rua nova, 456");
        novaPessoa.setCidade("Nova Cidade");
        novaPessoa.setEstado("SP");
        novaPessoa.setTelefone("8765-4321");
        novaPessoa.setEmail("pedro@email.com");
        boolean sucessoAlteracao = pessoaFisicaDAO.alterar(novaPessoa);

        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        System.out.println("Listando todas as pessoas fisicas:");
        for (PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
        }
    }
}
```

```

    }

    boolean sucessoExclusao = pessoaFisicaDAO.excluir(novaPessoa.getId());

    PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    PessoaJuridica novaPessoaJuridica = new PessoaJuridica(0, "JJC", "Rua 11, Centro",
    "Riacho do Norte", "PA", "1212-1212", "jjc@riacho.com", "11111111111111");

    boolean sucessoInclusaoPJ = pessoaJuridicaDAO.incluir(novaPessoaJuridica);

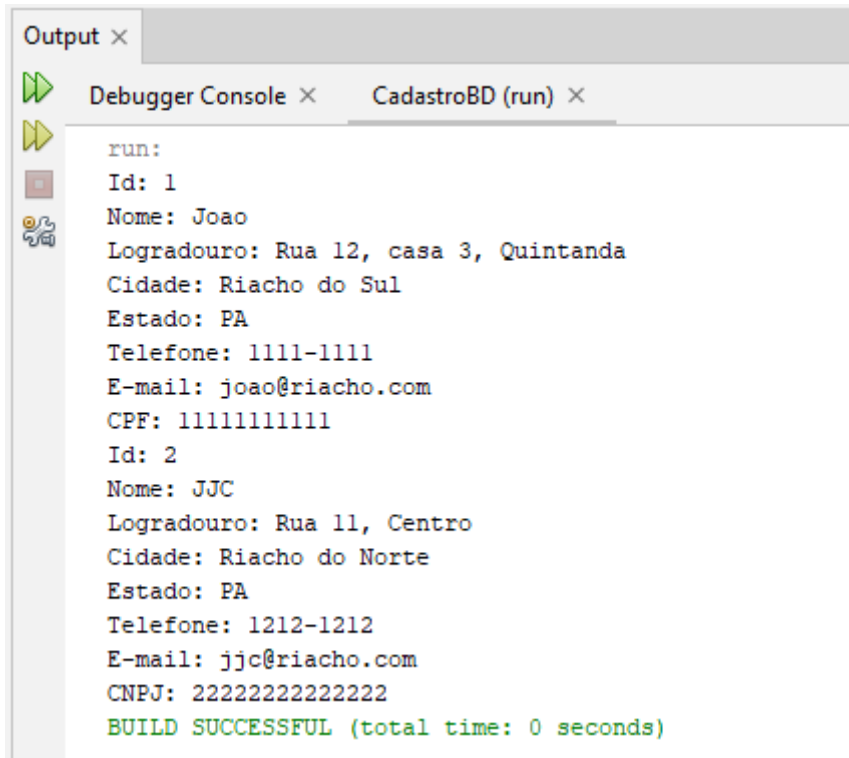
    novaPessoaJuridica.setNome("CCJ");
    novaPessoaJuridica.setLogradouro("Rua nova, 12");
    novaPessoaJuridica.setCidade("Cidade Nova");
    novaPessoaJuridica.setEstado("SP");
    novaPessoaJuridica.setTelefone("8765-1123");
    novaPessoaJuridica.setEmail("CCJ@email.com");
    boolean sucessoAlteracaoPJ = pessoaJuridicaDAO.alterar(novaPessoaJuridica);

    List<PessoaJuridica> todasPessoasJuridicas = pessoaJuridicaDAO.getPessoas();
    System.out.println("Listando todas as pessoas jurídicas:");
    for (PessoaJuridica pj : todasPessoasJuridicas) {
        pj.exibir();
    }

    boolean sucessoExclusaoPJ = pessoaJuridicaDAO.excluir(novaPessoaJuridica.getId());
}
}

```

## Resultado



```
run:
Id: 1
Nome: Joao
Logradouro: Rua 12, casa 3, Quintanda
Cidade: Riacho do Sul
Estado: PA
Telefone: 1111-1111
E-mail: joao@riacho.com
CPF: 11111111111
Id: 2
Nome: JJC
Logradouro: Rua 11, Centro
Cidade: Riacho do Norte
Estado: PA
Telefone: 1212-1212
E-mail: jjc@riacho.com
CNPJ: 22222222222222
BUILD SUCCESSFUL (total time: 0 seconds)
```

## **Análise e Conclusão:**

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

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel fundamental na arquitetura de software, atuando como intermediários entre as aplicações e os recursos de dados. Eles oferecem uma camada de abstração que permite que as aplicações interajam com o banco de dados de maneira uniforme e eficiente, sem se preocupar com detalhes específicos de implementação do banco de dados.

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

No contexto do JDBC, *Statement* e *PreparedStatement* são duas interfaces para executar comandos SQL, mas têm algumas diferenças importantes:

#### **Statement**

- É usado para executar consultas SQL estáticas sem parâmetros.
- A consulta é compilada no banco de dados cada vez que é executada.
- Pode ser menos eficiente se a mesma consulta SQL for executada várias vezes, pois precisa ser compilada sempre.
- É mais vulnerável a ataques de injeção de SQL, pois as consultas são concatenadas com strings, o que pode ser explorado se as entradas do usuário não forem sanitizadas corretamente.

#### **PreparedStatement**

- É usado para executar consultas SQL parametrizadas.
- A consulta é pré-compilada no banco de dados na primeira execução, o que pode melhorar o desempenho, especialmente para consultas executadas repetidamente.

- Reduz o risco de injeção de SQL, pois os parâmetros são definidos separadamente do código SQL e são automaticamente escapados pelo JDBC.
- Permite o uso de métodos setX (onde X é o tipo de dado) para definir parâmetros, tornando o código mais legível e seguro.

### **Como o padrão DAO melhora a manutenibilidade do software?**

O padrão Data Access Object (DAO) é uma estratégia de design que abstrai e encapsula o acesso a dados da lógica de negócios da aplicação. Ele atua como um intermediário entre a aplicação e a fonte de dados (por exemplo, um banco de dados SQL), permitindo que a aplicação interaja com a fonte de dados por meio de uma interface de alto nível.

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

Refletir herança em um modelo de banco de dados relacional, que por sua natureza não suporta diretamente conceitos de orientação a objetos como herança, envolve algumas abordagens padrão. Cada uma delas tem suas vantagens e desvantagens, e a escolha ideal geralmente depende das necessidades específicas da aplicação e das características do modelo de dados. As três principais estratégias são:

#### **1. Tabela Única (Single Table Inheritance):**

- **Como Funciona:** Uma única tabela contém colunas para todos os atributos de todas as classes na hierarquia de herança.
- **Vantagens:** Simplicidade, nenhuma necessidade de joins para consultas, e eficiência em operações de leitura.
- **Desvantagens:** A tabela pode ter muitas colunas nulas, especialmente se subclasses têm muitos atributos específicos, o que pode levar ao desperdício de espaço e tornar a manutenção mais complexa.



## 2. Tabela por Subclasse (Class Table Inheritance):

- **Como Funciona:** Cada classe na hierarquia (incluindo a superclasse) tem sua própria tabela. A tabela de uma subclasse armazena apenas os atributos específicos dessa subclasse e tem uma chave estrangeira que aponta para a tabela da superclasse.
- **Vantagens:** Evita colunas nulas e mantém a estrutura do banco de dados mais próxima à estrutura do modelo de objetos.
- **Desvantagens:** Pode ser menos eficiente em termos de desempenho devido à necessidade de joins para reconstruir um objeto completo. Também pode ser mais complexo para manter a integridade referencial.

## 3. Tabela por Classe Concreta (Concrete Table Inheritance):

- **Como Funciona:** Cada classe concreta na hierarquia de herança tem sua própria tabela, e estas tabelas não incluem dados de qualquer superclasse. Cada tabela de classe concreta contém todos os atributos dessa classe, incluindo os atributos herdados.
- **Vantagens:** Não há colunas nulas, e cada tabela de classe concreta pode ser otimizada de forma independente.
- **Desvantagens:** Pode resultar em duplicação de dados e maior complexidade em operações de atualização. Operações que envolvem a hierarquia inteira podem ser complexas, pois exigem a combinação de dados de várias tabelas.

## 2º Procedimento | Alimentando a Base

### Arquivo CadastroBD.java

```
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import java.util.Scanner;
import java.util.List;

public class CadastroBD {

    private static final Scanner scanner = new Scanner(System.in);
    private static final PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();
    private static final PessoaJuridicaDAO pessoaJuridicaDAO = new PessoaJuridicaDAO();

    public static void main(String[] args) {
        int opcao;
        do {
            System.out.println("=====");
            System.out.println("1 - Incluir Pessoa");
            System.out.println("2 - Alterar Pessoa");
            System.out.println("3 - Excluir Pessoa");
            System.out.println("4 - Buscar pelo Id");
            System.out.println("5 - Exibir Todos");
            System.out.println("0 - Finalizar Programa");
            System.out.println("=====");
            opcao = scanner.nextInt();
            scanner.nextLine();

            switch (opcao) {
                case 1:
                    incluir();
                    break;
                case 2:
                    alterar();
```

```

        break;
    case 3:
        excluir();
        break;
    case 4:
        exibirPorId();
        break;
    case 5:
        exibirTodos();
        break;
    case 0:
        System.out.println("Finalizando Programa.");
        break;
    default:
        System.out.println("Opção inválida.");
        break;
    }
} while (opcao != 0);

scanner.close();
}

private static void incluir() {
    System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
    String tipo = scanner.nextLine().trim().toUpperCase();

    if ("F".equals(tipo)) {
        System.out.println("Inserindo Pessoa Física...");
        System.out.print("Nome: ");
        String nome = scanner.nextLine();
        System.out.print("Logradouro: ");
        String logradouro = scanner.nextLine();
        System.out.print("Cidade: ");
        String cidade = scanner.nextLine();
        System.out.print("Estado: ");
        String estado = scanner.nextLine();
        System.out.print("Telefone: ");

```

```

String telefone = scanner.nextLine();
System.out.print("Email: ");
String email = scanner.nextLine();
System.out.print("CPF: ");
String cpf = scanner.nextLine();

PessoaFisica pf = new PessoaFisica(0, nome, logradouro, cidade, estado, telefone, email,
cpf);
boolean sucesso = pessoaFisicaDAO.incluir(pf);
if (sucesso) {
    System.out.println("Pessoa Física incluída com sucesso.");
    pf.exibir();
} else {
    System.out.println("Erro ao incluir Pessoa Física.");
}
} else if ("J".equals(tipo)) {

    System.out.println("Inserindo Pessoa Jurídica...");
    System.out.print("Nome: ");
    String nome = scanner.nextLine();
    System.out.print("Logradouro: ");
    String logradouro = scanner.nextLine();
    System.out.print("Cidade: ");
    String cidade = scanner.nextLine();
    System.out.print("Estado: ");
    String estado = scanner.nextLine();
    System.out.print("Telefone: ");
    String telefone = scanner.nextLine();
    System.out.print("Email: ");
    String email = scanner.nextLine();
    System.out.print("CNPJ: ");
    String cnpj = scanner.nextLine();

    PessoaJuridica pj = new PessoaJuridica(0, nome, logradouro, cidade, estado, telefone,
email, cnpj);
    boolean sucesso = pessoaJuridicaDAO.incluir(pj);
    if (sucesso) {

```

```

        System.out.println("Pessoa Jurídica incluída com sucesso.");
        pj.exibir();
    } else {
        System.out.println("Erro ao incluir Pessoa Jurídica.");
    }
} else {
    System.out.println("Opção inválida. Por favor, digite 'F' para Pessoa Física ou 'J' para
Pessoa Jurídica.");
}
}

private static void alterar() {
    System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
    String tipo = scanner.nextLine().toUpperCase();

    System.out.println("Digite o ID da pessoa para alterar:");
    int id = scanner.nextInt();
    scanner.nextLine();

    if ("F".equals(tipo)) {
        PessoaFisica pf = pessoaFisicaDAO.getPessoa(id);
        if (pf != null) {
            System.out.println("Dados atuais:");
            System.out.println(pf);

            System.out.print("Novo Nome (deixe em branco para manter): ");
            String nome = scanner.nextLine();
            if (!nome.isEmpty()) pf.setNome(nome);

            System.out.print("Novo Logradouro (deixe em branco para manter): ");
            String logradouro = scanner.nextLine();
            if (!logradouro.isEmpty()) pf.setLogradouro(logradouro);

            System.out.print("Nova Cidade (deixe em branco para manter): ");
            String cidade = scanner.nextLine();
            if (!cidade.isEmpty()) pf.setCidade(cidade);

            System.out.print("Novo Estado (deixe em branco para manter): ");

```

```

String estado = scanner.nextLine();
if (!estado.isEmpty()) pf.setEstado(estado);

System.out.print("Novo Telefone (deixe em branco para manter): ");
String telefone = scanner.nextLine();
if (!telefone.isEmpty()) pf.setTelefone(telefone);

System.out.print("Novo Email (deixe em branco para manter): ");
String email = scanner.nextLine();
if (!email.isEmpty()) pf.setEmail(email);

System.out.print("Novo CPF (deixe em branco para manter): ");
String cpf = scanner.nextLine();
if (!cpf.isEmpty()) pf.setCpf(cpf);

boolean sucesso = pessoaFisicaDAO.alterar(pf);
if (sucesso) {
    System.out.println("Pessoa Física atualizada com sucesso.");
    pf.exibir();
} else {
    System.out.println("Erro ao atualizar Pessoa Física.");
}
} else {
    System.out.println("Pessoa Física não encontrada.");
}
} else if ("J".equals(tipo)) {
    PessoaJuridica pj = pessoaJuridicaDAO.getPessoa(id);
    if (pj != null) {
        System.out.println("Dados atuais:");
        System.out.println(pj);

        System.out.print("Novo Nome (deixe em branco para manter): ");
        String nome = scanner.nextLine();
        if (!nome.isEmpty()) pj.setNome(nome);

        System.out.print("Novo Logradouro (deixe em branco para manter): ");
        String logradouro = scanner.nextLine();

```

```

        if (!logradouro.isEmpty()) pj.setLogradouro(logradouro);

        System.out.print("Nova Cidade (deixe em branco para manter): ");
        String cidade = scanner.nextLine();
        if (!cidade.isEmpty()) pj.setCidade(cidade);

        System.out.print("Novo Estado (deixe em branco para manter): ");
        String estado = scanner.nextLine();
        if (!estado.isEmpty()) pj.setEstado(estado);

        System.out.print("Novo Telefone (deixe em branco para manter): ");
        String telefone = scanner.nextLine();
        if (!telefone.isEmpty()) pj.setTelefone(telefone);

        System.out.print("Novo Email (deixe em branco para manter): ");
        String email = scanner.nextLine();
        if (!email.isEmpty()) pj.setEmail(email);

        System.out.print("Novo CNPJ (deixe em branco para manter): ");
        String cnpj = scanner.nextLine();
        if (!cnpj.isEmpty()) pj.setCnpj(cnpj);

        boolean sucesso = pessoaJuridicaDAO.alterar(pj);
        if (sucesso) {
            System.out.println("Pessoa Jurídica atualizada com sucesso.");
            pj.exibir();
        } else {
            System.out.println("Erro ao atualizar Pessoa Jurídica.");
        }
    } else {
        System.out.println("Pessoa Jurídica não encontrada.");
    }
} else {
    System.out.println("Opção inválida.");
}
}

```

```

private static void excluir() {
    System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
    String tipo = scanner.nextLine();

    System.out.println("Digite o ID da pessoa para excluir:");
    int id = scanner.nextInt();
    scanner.nextLine();

    if (tipo.equalsIgnoreCase("F")) {
        boolean sucesso = pessoaFisicaDAO.excluir(id);
        if (sucesso) {
            System.out.println("Pessoa Física excluída com sucesso.");
        } else {
            System.out.println("Erro ao excluir Pessoa Física.");
        }
    } else if (tipo.equalsIgnoreCase("J")) {
        boolean sucesso = pessoaJuridicaDAO.excluir(id);
        if (sucesso) {
            System.out.println("Pessoa Jurídica excluída com sucesso.");
        } else {
            System.out.println("Erro ao excluir Pessoa Jurídica.");
        }
    } else {
        System.out.println("Opção inválida.");
    }
}

private static void exibirPorId() {
    System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
    String tipo = scanner.nextLine();
    System.out.println("Digite o ID da pessoa para exibir:");
    int id = scanner.nextInt();
    scanner.nextLine();

    if (tipo.equalsIgnoreCase("F")) {
        PessoaFisica pf = pessoaFisicaDAO.getPessoa(id);
        if (pf != null) {
            System.out.println("Exibindo dados de Pessoa Física:");

```



```

        pf.exibir();
    } else {
        System.out.println("Pessoa Fisica não encontrada.");
    }
} else if (tipo.equalsIgnoreCase("J")) {
    PessoaJuridica pj = pessoaJuridicaDAO.getPessoa(id);
    if (pj != null) {
        System.out.println("Exibindo dados de Pessoa Juridica:");
        pj.exibir();
    } else {
        System.out.println("Pessoa Jurídica não encontrada.");
    }
} else {
    System.out.println("Opção inválida.");
}
}

private static void exibirTodos() {
    System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
    String tipo = scanner.nextLine();

    if (tipo.equalsIgnoreCase("F")) {
        List<PessoaFisica> pessoasFisicas = pessoaFisicaDAO.getPessoas();
        System.out.println("Lista de Pessoas Físicas:");
        for (PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
        }
    } else if (tipo.equalsIgnoreCase("J")) {
        List<PessoaJuridica> pessoasJuridicas = pessoaJuridicaDAO.getPessoas();
        System.out.println("Lista de Pessoas Jurídicas:");
        for (PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
        }
    } else {
        System.out.println("Opção inválida.");
    }
}
}
}

```

## Resultado

```
run:
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
5
F - Pessoa Física | J - Pessoa Jurídica
f
Lista de Pessoas Físicas:
Id: 103
Nome: João
Logradouro: Rua 12, casa 3, Quintanda
Cidade: Riacho do Sul
Estado: PA
Telefone: 1111-1111
E-mail: joao@riacho.com
CPF: 11111111111
Id: 104
Nome: Jessy
Logradouro: Rua velha
Cidade: Cidade Velha
Estado: SP
Telefone: 112121212
E-mail: jessy@email.com
CPF: 13578235874
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
0 - Finalizar Programa
=====
```

## **Análise e Conclusão:**

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

A persistência em arquivo é uma forma simples de armazenar dados que se destaca pela facilidade com que os desenvolvedores podem criar, ler e escrever dados usando funções básicas de entrada e saída. É uma abordagem flexível que permite o armazenamento de dados em diversos formatos, como texto, binário ou formatos específicos. A portabilidade é outra vantagem dos arquivos, pois eles podem ser facilmente transferidos entre diferentes sistemas e plataformas. No entanto, quando se trata de gerenciar grandes volumes de dados, a persistência em arquivo pode se tornar ineficiente, principalmente para buscas e filtragens complexas. Além disso, essa abordagem não fornece mecanismos robustos para lidar com acessos concorrentes, integridade de dados ou segurança, que dependem da implementação de soluções específicas na aplicação.

Por outro lado, a persistência em banco de dados é gerenciada por sistemas de gerenciamento de banco de dados (DBMS), que oferecem uma série de funcionalidades projetadas para o manuseio eficiente de grandes quantidades de dados. Bancos de dados são altamente escaláveis e fornecem mecanismos avançados de consulta e indexação, o que permite realizar operações de busca e manipulação de dados de forma muito mais eficiente. Além disso, eles tratam de questões de concorrência e transações automaticamente, mantendo a consistência e a integridade dos dados. No que se refere à segurança, os bancos de dados oferecem controles de acesso refinados, com permissões detalhadas para usuários e operações específicas. Apesar dessas vantagens, bancos de dados são mais complexos e exigem manutenção regular, o que inclui otimização de desempenho, backups e atualizações de segurança.

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

As lambdas no Java trouxeram uma forma mais enxuta e direta de realizar operações que, antes, exigiam código mais extenso e complexo. Em essência, elas são como atalhos que permitem expressar conceitos de programação funcional com poucas linhas de código.

Por exemplo, para imprimir uma lista de objetos, antigamente você precisaria de um loop for para percorrer a lista e um comando de impressão para cada elemento. Agora, com as lambdas, você pode fazer tudo isso em uma linha só, utilizando o método `forEach` junto com uma expressão lambda que indica o que fazer com cada elemento - neste caso, imprimir.

Isso torna o código mais legível e mais focado no que você quer fazer, ao invés de como fazer. É uma forma mais moderna e fluída de programar, que facilita tanto a escrita quanto a leitura do código.

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

Métodos marcados como *static* pertencem à classe e não a uma instância específica (ou objeto) da classe. Isso significa que eles podem ser chamados diretamente no nome da classe, sem precisar criar um objeto dela.

Quando o Java executa um programa, ele começa pelo método main, que é um método estático. Como ainda não existem objetos quando o programa inicia, somente métodos estáticos podem ser chamados diretamente do main. Se tentasse chamar um método não estático, o Java não saberia a qual objeto esse método pertence, pois esse objeto ainda não foi criado. Por isso, dentro do main, só pode chamar diretamente outros métodos estáticos, a menos que você crie um objeto e chame métodos por meio dessa instância.