



UNIVERSIDADE
Estácio de Sá

Universidade	Estácio de Sá
Campus	Polo de Cobilândia / Vila – Velha/ES
Nome do Curso	Desenvolvimento Full Stack
Nome da Disciplina	RPG0016 – BackEnd sem banco não tem
Turma	9001
Semestre	Primeiro Semestre de 2024
Integrantes do Grupo	Tiago de Jesus Pereira Furtado
Matrícula	202306189045

**VILA VELHA
2024**

Criação de aplicativo Java, com acesso ao banco de dados SQL Server através do middleware JDBC.

👉 1º Procedimento | Mapeamento Objeto-Relacional e DAO

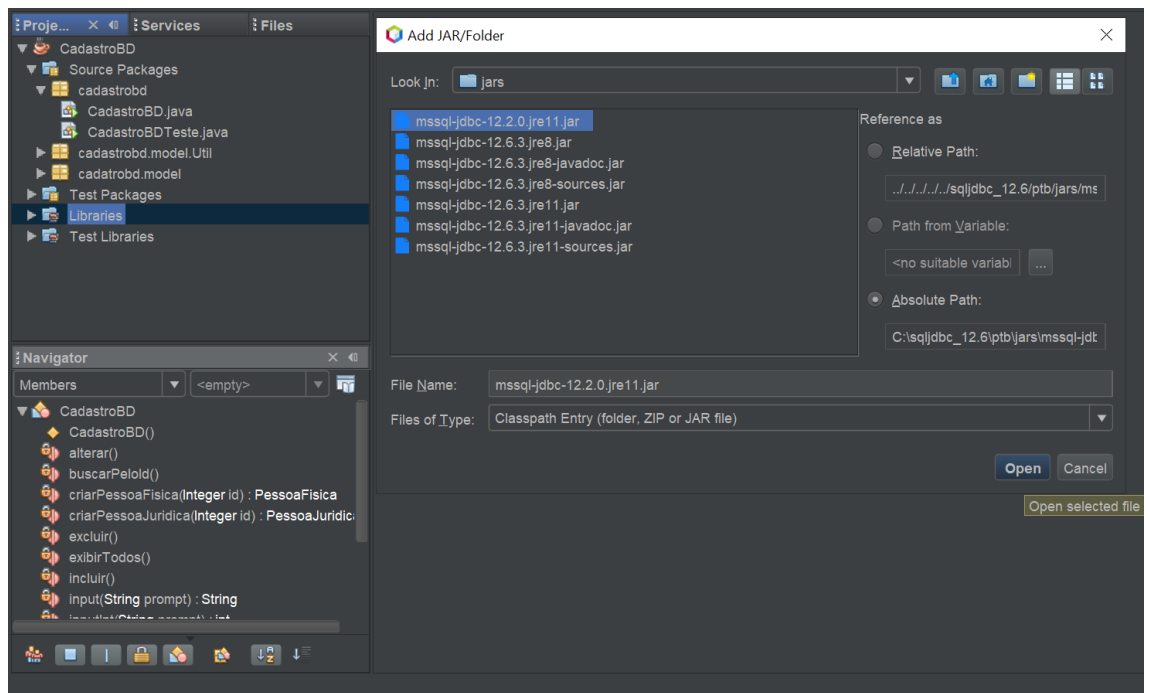
1- Objetivo da Prática:

- 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 SQL Server na persistência de dados.



1- Criar o projeto e configurar as bibliotecas necessárias:

- ✓ Criar um projeto no NetBeans, utilizando o nome CadastroBD, do tipo Aplicativo Java Padrão (modelo Ant).
- ✓ Adicionar o driver JDBC para SQL Server ao projeto, com o clique do botão direito sobre bibliotecas (libraries) e escolha da opção jar.

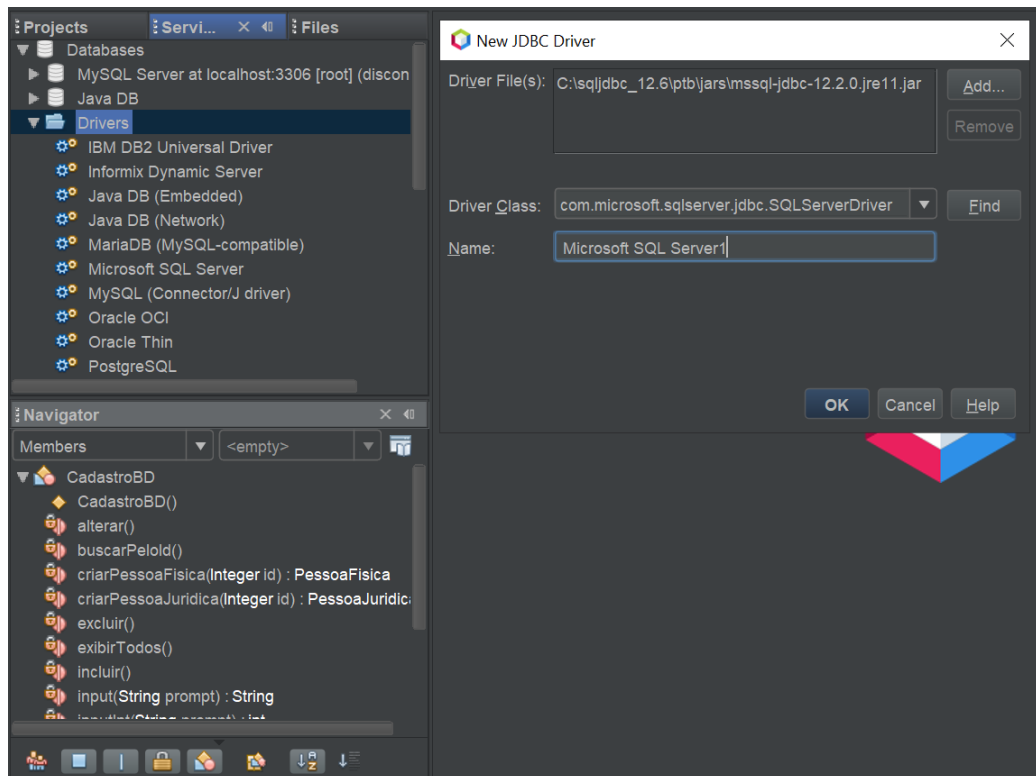


“Descrição do Código”

Adicionando a biblioteca “msql-jdbc-12.2.0.jre11.jar”

2- Configurar o acesso ao banco pela aba de serviços do NetBeans.

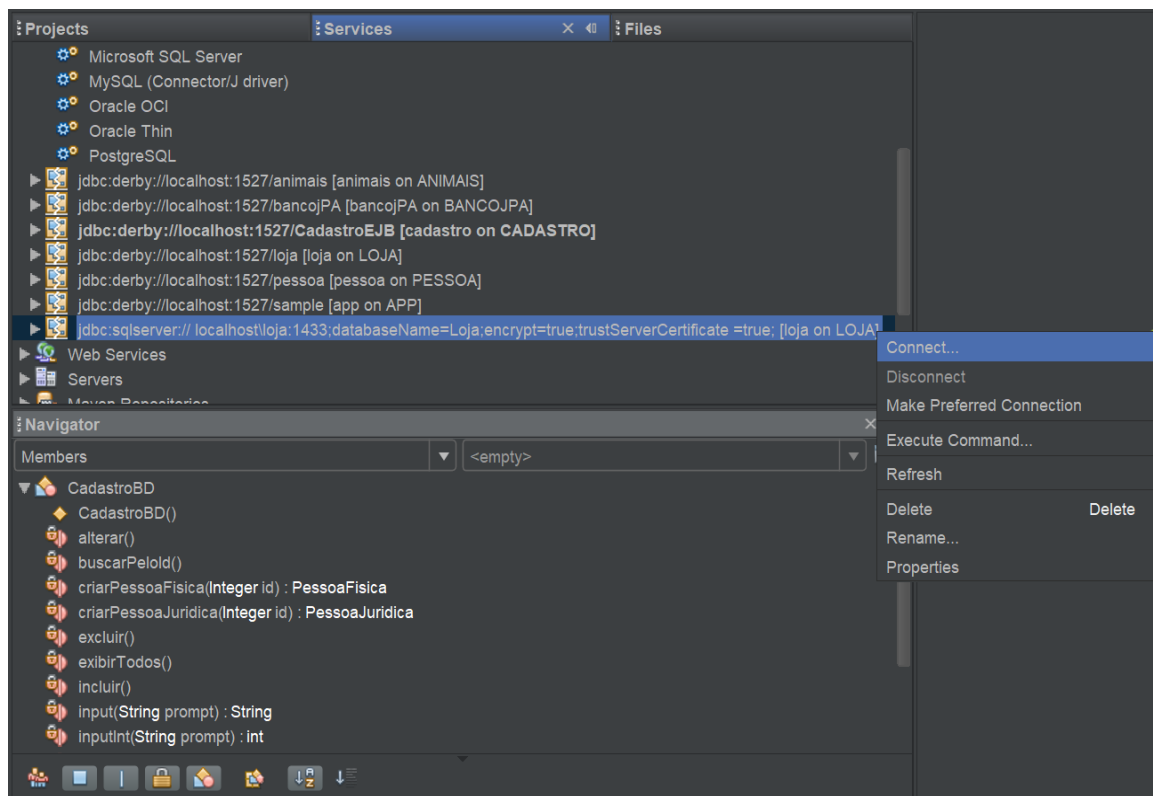
- ✓ Na aba de Serviços, divisão Banco de Dados, clique com o botão direito em Drivers e escolha Novo Driver.
- ✓ Na janela que se abrirá, clicar em Add (Adicionar), escolher o arquivo jar utilizado no passo anterior e finalizar com Ok.
- ✓ O reconhecimento será automático, e podemos definir uma conexão com o clique do botão direito sobre o driver e escolha de conectar utilizando.



“Descrição do Código”

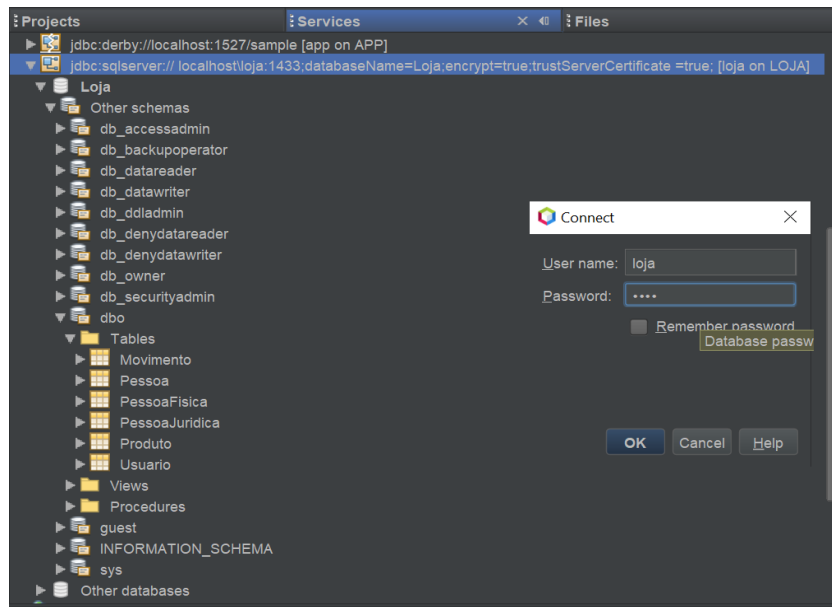
Adicionando em Drivers a nossa biblioteca “msql-jdbc-12.2.0.jre11.jar”

- ✓ Para os campos database, user e password, utilizar o valor loja, de acordo com os elementos criados em exercício anterior sobre a criação do banco de dados de exemplo, marcando também a opção Lembrar Senha.
- ✓ Para o campo JDBC URL deve ser utilizada a seguinte expressão:
`jdbc:sqlserver://localhost:1433; databaseName=loja; encrypt=true; trustServerCertificate=true;`
- ✓ Clicar em Testar Conexão e, estando tudo certo, Finalizar.
- ✓ Ao clicar duas vezes na nova conexão, os objetos do banco estarão todos disponíveis na árvore de navegação.

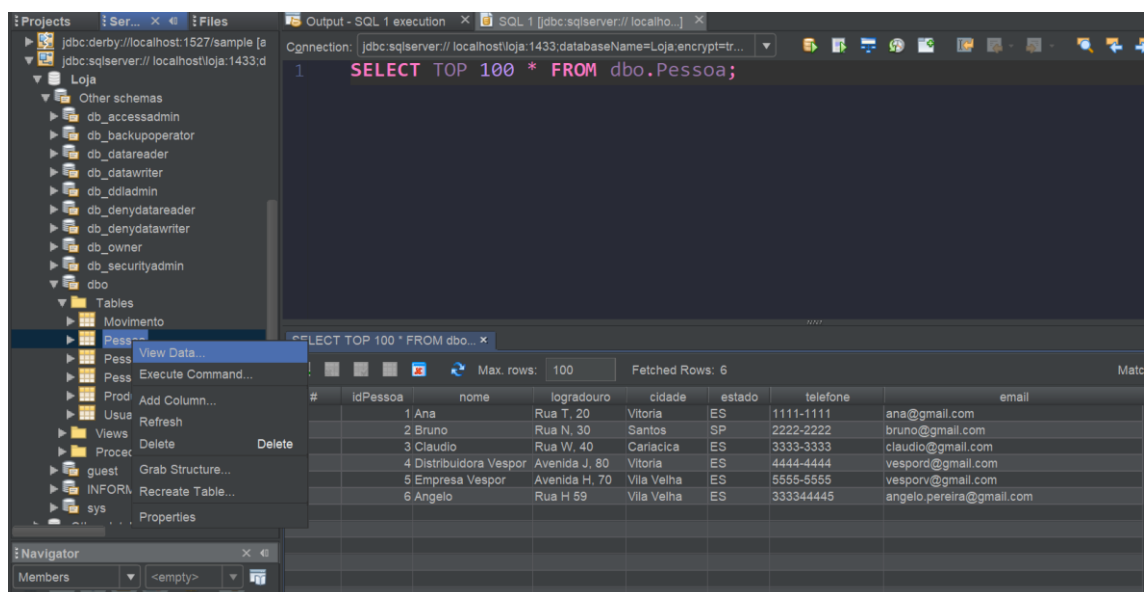


“Descrição do código”
Conectando no nosso sql serve

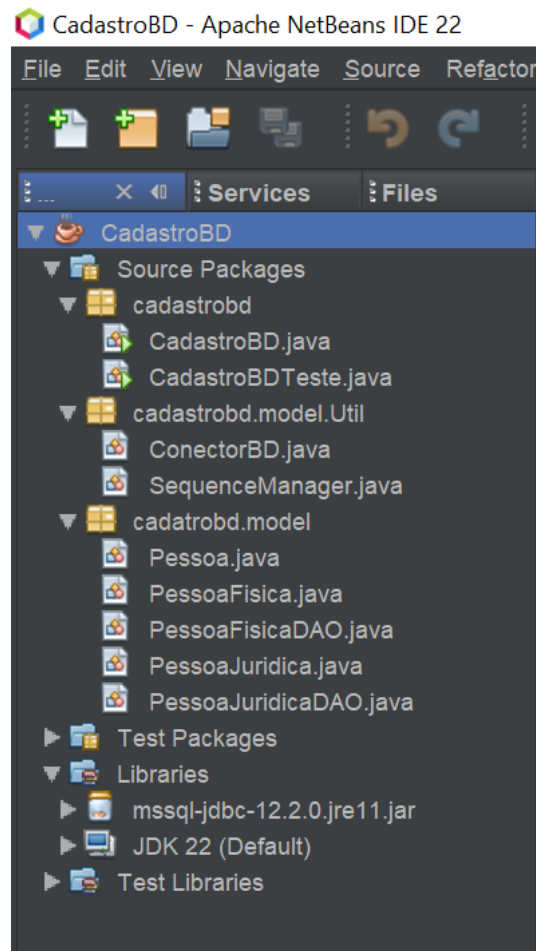
Conectando o Sql serve utilizando a o usuário “loja” e a senha “loja”



Utilizando a opção “View Data ” verificando as informações das tabelas



3 - Todos os códigos solicitados neste roteiro de aula:

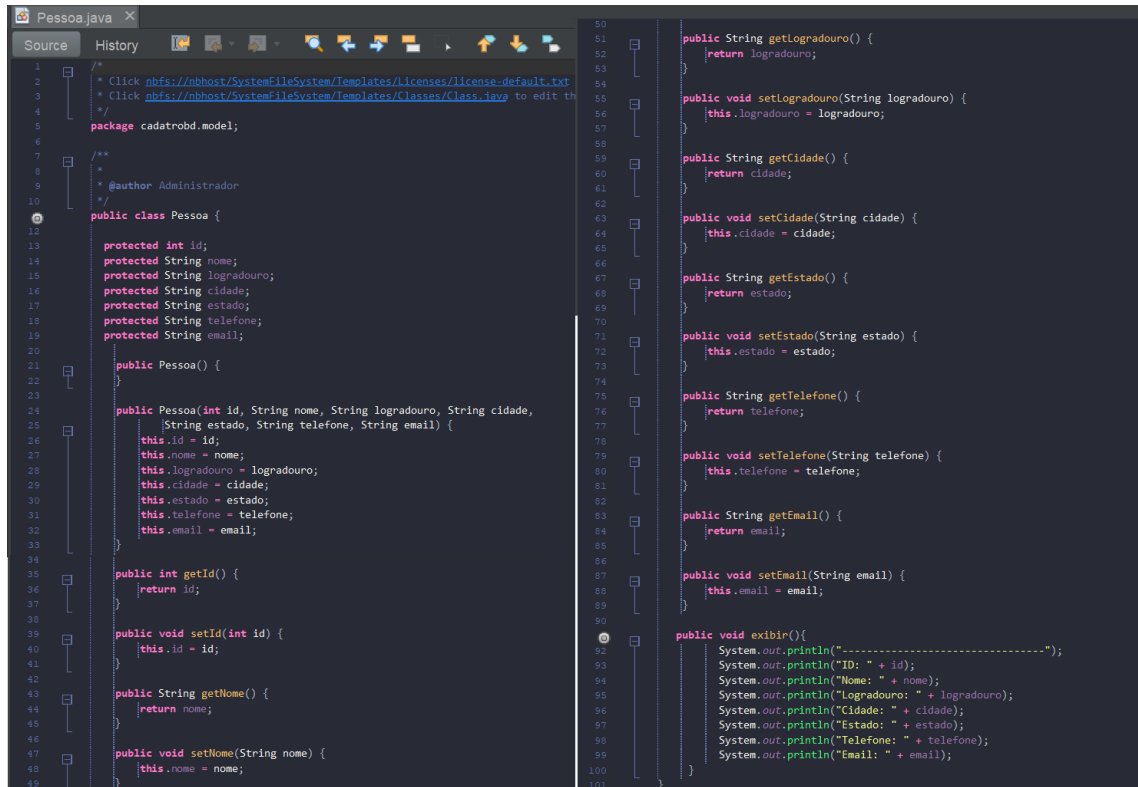


“Descrição”

A imagem mostra a estrutura final do projeto CadastroBD
Podemos ver os packages juntamente com as classes solicitadas no material separado de acordo com suas funções.

Voltando ao projeto, criar o pacote cadastrobd. model, e nele criar as classes apresentadas a seguir:

Criando a Classe Pessoa:

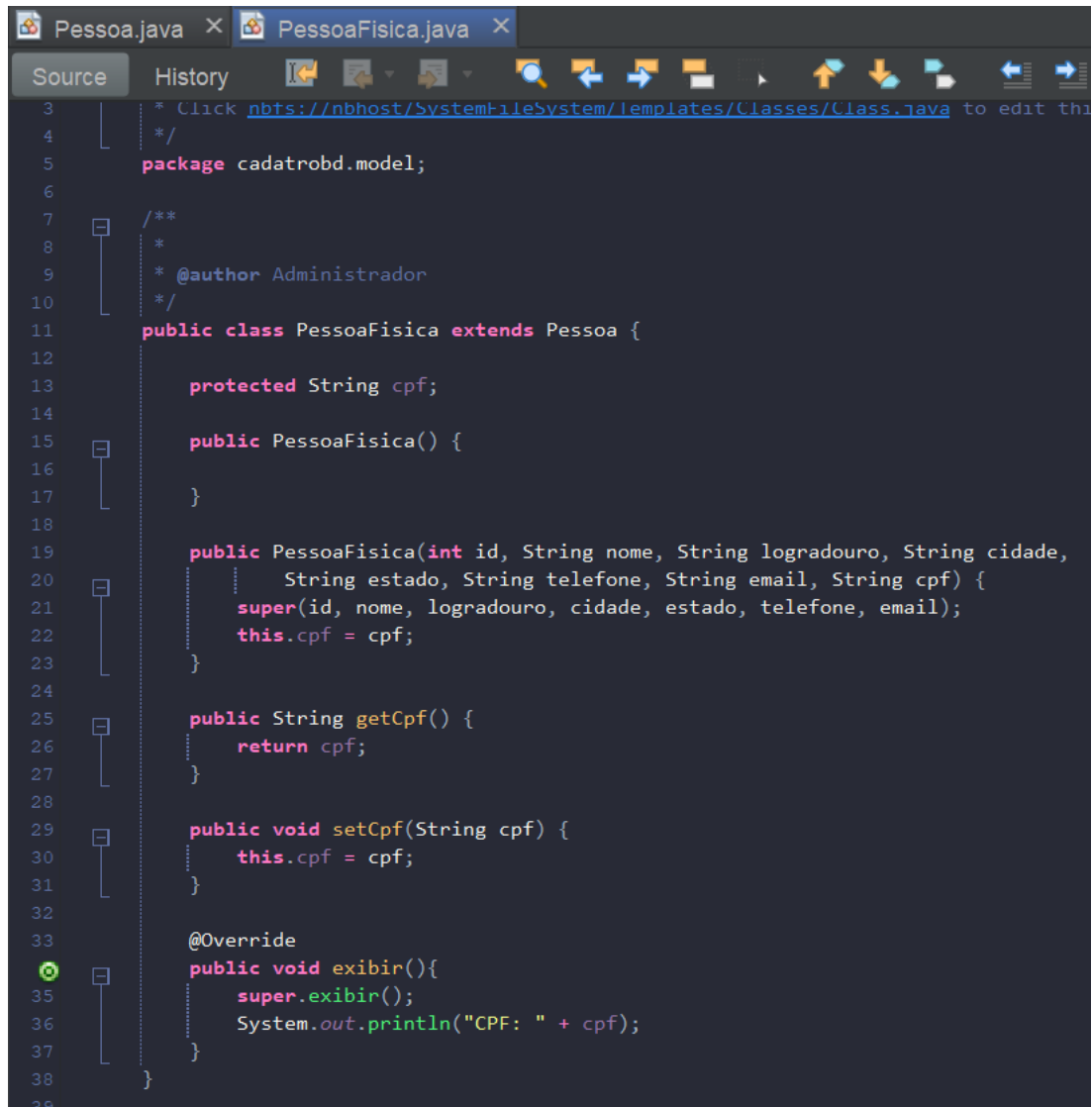


```
1  /**
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit th
4   */
5   package cadastrobd.model;
6
7
8   /**
9    * @author Administrador
10   */
11  public class Pessoa {
12
13      protected int id;
14      protected String nome;
15      protected String logradouro;
16      protected String cidade;
17      protected String estado;
18      protected String telefone;
19      protected String email;
20
21      public Pessoa() {
22
23      }
24
25      public Pessoa(int id, String nome, String logradouro, String cidade,
26                  String estado, String telefone, String email) {
27          this.id = id;
28          this.nome = nome;
29          this.logradouro = logradouro;
30          this.cidade = cidade;
31          this.estado = estado;
32          this.telefone = telefone;
33          this.email = email;
34      }
35
36      public int getId() {
37          return id;
38      }
39
40      public void setId(int id) {
41          this.id = id;
42      }
43
44      public String getNome() {
45          return nome;
46      }
47
48      public void setNome(String nome) {
49          this.nome = nome;
50      }
51
52      public String getLogradouro() {
53          return logradouro;
54      }
55
56      public void setLogradouro(String logradouro) {
57          this.logradouro = logradouro;
58      }
59
60      public String getCidade() {
61          return cidade;
62      }
63
64      public void setCidade(String cidade) {
65          this.cidade = cidade;
66      }
67
68      public String getEstado() {
69          return estado;
70      }
71
72      public void setEstado(String estado) {
73          this.estado = estado;
74      }
75
76      public String getTelefone() {
77          return telefone;
78      }
79
80      public void setTelefone(String telefone) {
81          this.telefone = telefone;
82      }
83
84      public String getEmail() {
85          return email;
86      }
87
88      public void setEmail(String email) {
89          this.email = email;
90      }
91
92      public void exhibit(){
93          System.out.println("-----");
94          System.out.println("ID: " + id);
95          System.out.println("Nome: " + nome);
96          System.out.println("Logradouro: " + logradouro);
97          System.out.println("Cidade: " + cidade);
98          System.out.println("Estado: " + estado);
99          System.out.println("Telefone: " + telefone);
100         System.out.println("Email: " + email);
101     }
```

“Descrição”

Classe Pessoa, com os campos id, nome, logradouro, cidade, estado, telefone e Email, construtor padrão e completo, além de método exhibit, para impressão dos dados no console.

Criando a Classe PessoaFisica:



```
3  * Click https://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadatrobd.model;
6
7  /**
8   *
9   * @author Administrador
10  */
11  public class PessoaFisica extends Pessoa {
12
13      protected String cpf;
14
15      public PessoaFisica() {
16
17      }
18
19      public PessoaFisica(int id, String nome, String logradouro, String cidade,
20          String estado, String telefone, String email, String cpf) {
21          super(id, nome, logradouro, cidade, estado, telefone, email);
22          this.cpf = cpf;
23      }
24
25      public String getCpf() {
26          return cpf;
27      }
28
29      public void setCpf(String cpf) {
30          this.cpf = cpf;
31      }
32
33      @Override
34      public void exibir(){
35          super.exibir();
36          System.out.println("CPF: " + cpf);
37      }
38  }
```

“Descrição”

Classe PessoaFisica, herdando de Pessoa, com acréscimo do campo cpf, além da reescrita dos construtores e uso de polimorfismo em exibir.

Criando a Classe PessoaJuridica:

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this
4  */
5  package cadastrobd.model;
6
7  /**
8   *
9   * @author Administrador
10  */
11  public class PessoaJuridica extends Pessoa {
12
13      protected String cnpj;
14
15      public PessoaJuridica() {
16      }
17
18      public PessoaJuridica(int id, String nome, String logradouro, String cidade,
19                          String estado, String telefone, String email, String cnpj) {
20          super(id, nome, logradouro, cidade, estado, telefone, email);
21          this.cnpj = cnpj;
22      }
23
24      public String getCnpj() {
25          return cnpj;
26      }
27
28      public void setCnpj(String cnpj) {
29          this.cnpj = cnpj;
30      }
31
32      @Override
33      public void exibir(){
34          super.exibir();
35          System.out.println("CPF: " + cnpj);
36      }
37  }
```

“Descrição”

Classe PessoaJuridica, herdando de Pessoa, com acréscimo do campo cnpj, além da reescrita dos construtores e uso de polimorfismo em exibir

Criar o pacotes cadastro.model.util, para inclusão das classes utilitárias que são apresentadas a seguir:

Criando a Classe ConectorBD:

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change th
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastrobd.model.Util;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.sql.Statement;
13
14 /**
15 *
16 * @author Administrador
17 */
18 public class ConectorBD {
19
20     private static final String URL = "jdbc:sqlserver://localhost:1433;databaseName=Loja;"
21     + "encrypt=true;trustServerCertificate=true";
22     private static final String USER = "loja";
23     private static final String PASSWORD = "loja";
24
25     public static Connection getConnection() throws SQLException {
26         return DriverManager.getConnection(URL, USER, PASSWORD);
27     }
28
29     public static PreparedStatement getPrepared(String sql) throws SQLException {
30         return getConnection().prepareStatement(sql);
31     }
32
33     public static ResultSet getSelect(PreparedStatement stmt) throws SQLException {
34         return stmt.executeQuery();
35     }
36
37     public static void close(Connection conn) throws SQLException {
38         if (conn != null) {
39             conn.close();
40         }
41     }
42
43     public static void close(Statement stmt) throws SQLException {
44         if (stmt != null) {
45             stmt.close();
46         }
47     }
48
49     public static void close(ResultSet rs) throws SQLException {
50         if (rs != null) {
51             rs.close();
52         }
53     }
54 }
55
56
```

“Descrição”

Classe ConectorBD, com os métodos getConnection, para retornar uma conexão com o banco de dados, getPrepared, para retornar um objeto do tipo PreparedStatement a partir de um SQL fornecido com parâmetro, e getSelect, para retornar o ResultSet relacionado a uma consulta.

Criando a Classe SequenceManager:

```
5 package cadastrobd.model.Util;
6
7 import com.microsoft.sqlserver.jdbc.SQLServerException;
8 import java.sql.Connection;
9 import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12
13 /**
14  *
15  * @author Administrador
16  */
17 public class SequenceManager {
18
19     public static int getValue(String sequenceName) throws SQLException {
20         String sql = "SELECT NEXT VALUE FOR " + sequenceName + " AS nextval";
21         try (Connection conn = ConectorBD.getConnection();
22             PreparedStatement stmt = conn.prepareStatement(sql);
23             ResultSet rs = stmt.executeQuery()) {
24             if (rs.next()) {
25                 return rs.getInt("nextval");
26             } else {
27                 throw new SQLException("Não foi possível obter o próximo valor da sequência "
28                                         + sequenceName);
29             }
30         }
31     }
32 }
```

"Descrição"

Classe SequenceManager, que terá o método getValue, recebendo o nome da sequência como parâmetro e retornando o próximo valor.

Codificar as classes no padrão DAO, no pacote cadastro.model:

Criando a Classe PessoaFisicaDAO:

```
package cadastro.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;

/**
 * @author Administrador
 */
public class PessoaFisicaDAO {

    public PessoaFisica getPessoa(int id) throws SQLException {
        String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
            "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaFisica.cpf " +
            "FROM Pessoa " +
            "JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa " +
            "WHERE Pessoa.idPessoa = ?";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = conn.createStatement().executeQuery()) {
                if (rs.next()) {
                    PessoaFisica pessoa = new PessoaFisica();
                    pessoa.setId(rs.getInt("idPessoa"));
                    pessoa.setNome(rs.getString("nome"));
                    pessoa.setLogradouro(rs.getString("logradouro"));
                    pessoa.setCidade(rs.getString("cidade"));
                    pessoa.setEstado(rs.getString("estado"));
                    pessoa.setTelefone(rs.getString("telefone"));
                    pessoa.setEmail(rs.getString("email"));
                    pessoa.setCpf(rs.getString("cpf"));
                    return pessoa;
                }
            }
        }
        return null;
    }

    public List<PessoaFisica> getPessoas() throws SQLException {
        List<PessoaFisica> pessoas = new ArrayList<>();
        String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
            "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaFisica.cpf " +
            "FROM Pessoa " +
            "JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rs = conn.createStatement().executeQuery()) {
            while (rs.next()) {
                PessoaFisica pessoa = new PessoaFisica();
                pessoa.setId(rs.getInt("idPessoa"));
                pessoa.setNome(rs.getString("nome"));
                pessoa.setLogradouro(rs.getString("logradouro"));
                pessoa.setCidade(rs.getString("cidade"));
                pessoa.setEstado(rs.getString("estado"));
                pessoa.setTelefone(rs.getString("telefone"));
                pessoa.setEmail(rs.getString("email"));
                pessoa.setCpf(rs.getString("cpf"));
                pessoas.add(pessoa);
            }
        }
        return pessoas;
    }

    public void incluir(PessoaFisica pessoa) throws SQLException {
        String sqlInsertPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, " +
            "estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
        String sqlInsertPessoaFisica = "INSERT INTO PessoaFisica (idPessoa, cpf) " +
            "VALUES (?, ?)";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtInsertPessoa = conn.prepareStatement(sqlInsertPessoa);
            PreparedStatement stmtInsertPessoaFisica = conn.prepareStatement(sqlInsertPessoaFisica)) {
            stmtInsertPessoa.setInt(1, pessoa.getId());
            stmtInsertPessoa.setString(2, pessoa.getNome());
            stmtInsertPessoa.setString(3, pessoa.getLogradouro());
            stmtInsertPessoa.setString(4, pessoa.getCidade());
            stmtInsertPessoa.setString(5, pessoa.getEstado());
            stmtInsertPessoa.setString(6, pessoa.getTelefone());
            stmtInsertPessoa.setString(7, pessoa.getEmail());
            stmtInsertPessoa.executeUpdate();

            stmtInsertPessoaFisica.setInt(1, pessoa.getId());
            stmtInsertPessoaFisica.setString(2, pessoa.getCpf());
            stmtInsertPessoaFisica.executeUpdate();
        }
    }

    public void alterar(PessoaFisica pessoa, String novoNome, String novoLogradouro, String novaCidade,
        String novoEstado, String novoTelefone, String novoEmail, String novoCpf) throws SQLException {
        String sql = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, " +
            "estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
        String sqlFisica = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa = ?";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql);
            PreparedStatement stmtFisica = conn.prepareStatement(sqlFisica)) {
            stmt.setString(1, novoNome);
            stmt.setString(2, novoLogradouro);
            stmt.setString(3, novaCidade);
            stmt.setString(4, novoEstado);
            stmt.setString(5, novoTelefone);
            stmt.setString(6, novoEmail);
            stmt.setInt(7, pessoa.getId());
            stmt.executeUpdate();

            stmtFisica.setString(1, novoCpf);
            stmtFisica.setInt(2, pessoa.getId());
            stmtFisica.executeUpdate();
        }
    }

    public void excluir(int id) throws SQLException {
        String sql = "DELETE FROM PessoaFisica WHERE idPessoa = ?";
        String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
        try (Connection conn = ConectorBD.getConnection();
            PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {
            stmtPessoa.setInt(1, id);
            stmtPessoa.executeUpdate();

            Pessoa pessoa = new Pessoa();
            pessoa.setId(id);
            stmtPessoa.setInt(1, id);
            stmtPessoa.executeUpdate();

            System.out.println("Pessoa fisica excluida com ID: " + id);
        }
    }
}
```

"Descrição"

Classe PessoaFisicaDAO, com os métodos getPessoa, retornando uma pessoa física a partir do seu id, getPessoas, para retorno de todas as pessoas físicas do banco de dados, incluir, para inclusão de uma pessoa física, fornecida como parâmetro, nas tabelas Pessoa e PessoaFisica, alterar, para alteração dos dados de uma pessoa física, e excluir, para remoção da pessoa do banco em ambas as tabelas.

Criando a Classe PessoaJuridicaDAO:

```
1  //
2  * Click http://localhost/System/Template/Classes/ClassesDefault.txt to change this license
3  * Click http://localhost/System/Template/Classes/Classes.java to edit this template
4
5  package cadastro.model;
6
7  import cadastro.model.Util.ConectorBD;
8  import java.sql.Connection;
9  import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.util.ArrayList;
13 import java.util.List;
14
15 /**
16  * @author Administrator
17  */
18 public class PessoaJuridicaDAO {
19
20     public PessoaJuridica getPessoa(int id) throws SQLException {
21         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
22             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaJuridica.cnpj " +
23             "FROM Pessoa " +
24             "JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa " +
25             "WHERE Pessoa.idPessoa = ?";
26         try (Connection conn = ConectorBD.getConnection();
27             PreparedStatement stmt = ConectorBD.getPrepared(sql)) {
28             stmt.setInt(1, id);
29             try (ResultSet rs = ConectorBD.getSelect(stmt)) {
30                 if (rs.next()) {
31                     PessoaJuridica pessoa = new PessoaJuridica();
32                     pessoa.setId(rs.getInt("idPessoa"));
33                     pessoa.setNome(rs.getString("nome"));
34                     pessoa.setLogradouro(rs.getString("logradouro"));
35                     pessoa.setCidade(rs.getString("cidade"));
36                     pessoa.setEstado(rs.getString("estado"));
37                     pessoa.setTelefone(rs.getString("telefone"));
38                     pessoa.setEmail(rs.getString("email"));
39                     pessoa.setCnpj(rs.getString("cnpj"));
40                     return pessoa;
41                 }
42             }
43             return null;
44         }
45     }
46
47     public List<PessoaJuridica> getPessoas() throws SQLException {
48         List<PessoaJuridica> pessoas = new ArrayList<>();
49         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
50             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaJuridica.cnpj " +
51             "FROM Pessoa " +
52             "JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa";
53         try (Connection conn = ConectorBD.getConnection();
54             PreparedStatement stmt = ConectorBD.getPrepared(sql);
55             ResultSet rs = ConectorBD.getSelect(stmt)) {
56             while (rs.next()) {
57                 PessoaJuridica pessoa = new PessoaJuridica();
58                 pessoa.setId(rs.getInt("idPessoa"));
59                 pessoa.setNome(rs.getString("nome"));
60                 pessoa.setLogradouro(rs.getString("logradouro"));
61                 pessoa.setCidade(rs.getString("cidade"));
62                 pessoa.setEstado(rs.getString("estado"));
63                 pessoa.setTelefone(rs.getString("telefone"));
64                 pessoa.setEmail(rs.getString("email"));
65                 pessoa.setCnpj(rs.getString("cnpj"));
66                 pessoas.add(pessoa);
67             }
68             return pessoas;
69         }
70     }
71
72     public void incluir(PessoaJuridica pessoa) throws SQLException {
73         String sqlInsertPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, " +
74             "estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
75         String sqlInsertPessoaJuridica = "INSERT INTO PessoaJuridica (idPessoa, cnpj) " +
76             "VALUES (?, ?)";
77         try (Connection conn = ConectorBD.getConnection();
78             PreparedStatement stmtInsertPessoa = conn.prepareStatement(sqlInsertPessoa);
79             PreparedStatement stmtInsertPessoaJuridica = conn.prepareStatement(sqlInsertPessoaJuridica)) {
80
81             stmtInsertPessoa.setInt(1, pessoa.getId());
82             stmtInsertPessoa.setString(2, pessoa.getNome());
83             stmtInsertPessoa.setString(3, pessoa.getLogradouro());
84             stmtInsertPessoa.setString(4, pessoa.getCidade());
85             stmtInsertPessoa.setString(5, pessoa.getEstado());
86             stmtInsertPessoa.setString(6, pessoa.getTelefone());
87             stmtInsertPessoa.setString(7, pessoa.getEmail());
88             stmtInsertPessoa.executeUpdate();
89
90             stmtInsertPessoaJuridica.setInt(1, pessoa.getId());
91             stmtInsertPessoaJuridica.setString(2, pessoa.getCnpj());
92             stmtInsertPessoaJuridica.executeUpdate();
93         }
94
95     public void alterar(PessoaJuridica pessoa, String novoNome, String novoLogradouro, String novaCidade,
96         String novoEstado, String novoTelefone, String novoEmail, String novoCnpj) throws SQLException {
97         String sql = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, " +
98             "estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
99         String sqlJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE idPessoa = ?";
100         try (Connection conn = ConectorBD.getConnection();
101             PreparedStatement stmt = conn.prepareStatement(sql);
102             PreparedStatement stmtJuridica = conn.prepareStatement(sqlJuridica)) {
103
104             stmt.setString(1, novoNome);
105             stmt.setString(2, novoLogradouro);
106             stmt.setString(3, novaCidade);
107             stmt.setString(4, novoEstado);
108             stmt.setString(5, novoTelefone);
109             stmt.setString(6, novoEmail);
110             stmt.setInt(7, pessoa.getId());
111             stmt.executeUpdate();
112
113             stmtJuridica.setString(1, novoCnpj);
114             stmtJuridica.setInt(2, pessoa.getId());
115             stmtJuridica.executeUpdate();
116         }
117
118     public void excluir(int id) throws SQLException {
119         String sql = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";
120         String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
121         try (Connection conn = ConectorBD.getConnection();
122             PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {
123             stmtPessoa.setInt(1, id);
124             stmtPessoa.executeUpdate();
125
126             stmtPessoa.setInt(1, id);
127             stmtPessoa.executeUpdate();
128
129             System.out.println("Pessoa jurídica excluída com ID: " + id);
130         }
131     }
132 }
```

"Descrição"

Classe PessoaJuridicaDAO, com os métodos getPessoa, retornando uma pessoa jurídica a partir do seu id, getPessoas, para retorno de todas as pessoas jurídicas do banco de dados, incluir, para inclusão de uma pessoa jurídica, fornecida como parâmetro, nas tabelas Pessoa e PessoaJuridica, alterar, para alteração dos dados de uma pessoa jurídica, e excluir, para remoção da pessoa do banco em ambas as tabelas.

Criar uma classe principal de testes com o nome CadastroBDTeste, efetuando as operações seguintes no método main:

- ✓ Instanciar uma pessoa física e persistir no banco de dados.
- ✓ Alterar os dados da pessoa física no banco.
- ✓ Consultar todas as pessoas físicas do banco de dados e listar no console.
- ✓ Excluir a pessoa física criada anteriormente no banco.
- ✓ Instanciar uma pessoa jurídica e persistir no banco de dados.
- ✓ Alterar os dados da pessoa jurídica no banco.
- ✓ Consultar todas as pessoas jurídicas do banco e listar no console.
- ✓ Excluir a pessoa jurídica criada anteriormente no banco.

Criando a Classe CadastroBDTeste:

```
package cadastrobd;

import java.sql.SQLException;
import java.util.List;
import cadastrobd.model.PessoaFisica;
import cadastrobd.model.PessoaFisicaDAO;
import cadastrobd.model.PessoaJuridica;
import cadastrobd.model.PessoaJuridicaDAO;
import cadastrobd.model.Util.ConectorBD;
import java.sql.Connection;

/**
 * @author Administrador
 */
public class CadastroBDTeste {

    public static void main(String[] args) {
        try {
            Connection conn = ConectorBD.getConnection();
            PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
            PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();

            // Criando uma pessoa fisica
            PessoaFisica pf = new PessoaFisica(6, "Angelo", "Rua Z, 40", "Vitoria", "ES",
                ["1234-5678", "angelo@gmail.com", "12345678910"]);

            // Fixar uma pessoa fisica no banco de dados
            pfDAO.incluir(pf);
            System.out.println("Pessoa fisica criada:");
            pf.exibir();
            System.out.println();

            // Alterando os dados da pessoa fisica no banco
            pfDAO.alterar(pf, "Angelo Pereira", "Rua B, 11", "Salvador", "BA",
                ["9999-8888", "angelo.pereira@gmail.com", "12345678900"]);
            System.out.println("-----");
            System.out.println("Dados da pessoa fisica alterados.");
            System.out.println("-----");
            System.out.println();

            // Realizar Consulta de todas as pessoas fisicas do banco de dados e listar no console
            List<PessoaFisica> pessoasFisicas = pfDAO.getPessoas();
            System.out.println("Todas as pessoas fisicas:");
            for (PessoaFisica pessoaFisica : pessoasFisicas) {
                pessoaFisica.exibir();
            }
            System.out.println();

            // Excluindo a pessoa fisica criada anteriormente no banco
            System.out.println("-----");
            pfDAO.excluir(pf.getId());
            System.out.println("-----");
            System.out.println();

            // Criando uma pessoa juridica
            PessoaJuridica pj = new PessoaJuridica(7, "Empresa Vespor", "Av. Darlin Santos, 90",
                ["Vitoria", "ES", "1234-5678", "empresa@vespor.com", "12345678901234"]);

            // Fixar uma pessoa juridica no banco de dados
            pjDAO.incluir(pj);
            System.out.println("Pessoa juridica criada:");
            pj.exibir();
            System.out.println();

            // Alterar os dados da pessoa juridica no banco
            pjDAO.alterar(pj, "Companhia AutomotivaSA", "Av. Nova, 200", "Salvador", "BA",
                ["9876-5432", "companhia@automotivasa.com", "98765432109876"]);
            System.out.println("-----");
            System.out.println("Dados da pessoa juridica alterados.");
            System.out.println("-----");
            System.out.println();

            // Realizar Consulta de todas as pessoas juridicas do banco de dados e listar no console
            List<PessoaJuridica> pessoasJuridicas = pjDAO.getPessoas();
            System.out.println("Todas as pessoas juridicas:");
            for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
                pessoaJuridica.exibir();
            }
            System.out.println();

            // Excluir a pessoa juridica criada anteriormente no banco
            System.out.println("-----");
            pjDAO.excluir(pj.getId());
            System.out.println("-----");
            System.out.println();

            ConectorBD.close(conn);
        } catch (SQLException e) {
            System.out.println("Ocorreu um erro: " + e.getMessage());
        }
    }
}
```

4 - Os resultados da execução dos códigos também devem ser apresentados:

Resultado da Classe CadastroBDTeste com as devidas alterações:

```
run:

Pessoa fisica criada:
-----
ID: 6
Nome: Angelo
Logradouro: Rua Z, 40
Cidade: Vitoria
Estado: ES
Telefone: 1234-5678
Email: angelo@gmail.com
CPF: 12345678910

-----
Dados da pessoa fisica alterados.
-----

Todas as pessoas fisicas:
-----
ID: 1
Nome: Ana
Logradouro: Rua T, 20
Cidade: Vitoria
Estado: ES
Telefone: 1111-1111
Email: ana@gmail.com
CPF: 11111111111
-----
ID: 2
Nome: Bruno
Logradouro: Rua N, 30
Cidade: Santos
Estado: SP
Telefone: 2222-2222
Email: bruno@gmail.com
CPF: 22222222222
-----
ID: 3
Nome: Claudio
Logradouro: Rua W, 40
Cidade: Cariacica
Estado: ES
Telefone: 3333-3333
Email: claudio@gmail.com
CPF: 33333333333
-----
ID: 6
Nome: Angelo Pereira
Logradouro: Rua B, 11
Cidade: Salvador
Estado: BA
Telefone: 9999-8888
Email: angelo.pereira@email.com
CPF: 12345678900

-----
Pessoa fisica excluida com ID: 6
-----

Pessoa juridica criada:
-----
ID: 7
Nome: Empresa Vespor
Logradouro: Av. Darlin Santos, 90
Cidade: Vitoria
Estado: ES
Telefone: 1234-5678
Email: empresa@vespor.com
CPF: 12345678901234

-----
Dados da pessoa juridica alterados.
-----

Todas as pessoas juridicas:
-----
ID: 4
Nome: Distribuidora Vespor
Logradouro: Avenida J, 80
Cidade: Vitoria
Estado: ES
Telefone: 4444-4444
Email: vespord@gmail.com
CPF: 444444444444444
-----
ID: 5
Nome: Empresa Vespor
Logradouro: Avenida H, 70
Cidade: Vila Velha
Estado: ES
Telefone: 5555-5555
Email: vesporv@gmail.com
CPF: 555555555555555
-----
ID: 7
Nome: Companhia AutomotivaSA
Logradouro: Av. Nova, 200
Cidade: Salvador
Estado: BA
Telefone: 9876-5432
Email: companhia@automotivasa.com
CPF: 98765432109876

-----
Pessoa juridica excluida com ID: 7
-----

BUILD SUCCESSFUL (total time: 2 seconds)
```


5 – Análise e Conclusão:

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

Os componentes de middleware, como o JDBC (Java Database Connectivity), são fundamentais na arquitetura de sistemas de software por várias razões:

- ✓ **Abstração e Simplificação:** Middleware como o JDBC fornece uma camada de abstração entre a aplicação e o banco de dados. Isso simplifica o desenvolvimento, permitindo que os desenvolvedores interajam com diferentes tipos de bancos de dados de maneira uniforme, sem se preocupar com os detalhes específicos de cada um.
- ✓ **Portabilidade:** Com o uso de JDBC, aplicações Java podem se conectar a qualquer banco de dados que tenha um driver JDBC compatível. Isso facilita a portabilidade da aplicação entre diferentes ambientes de banco de dados.
- ✓ **Gestão de Conexões:** O middleware gere as conexões com o banco de dados, otimizando recursos e melhorando a performance da aplicação. Ele pode implementar pools de conexões para reutilizar conexões abertas, reduzindo a sobrecarga de abrir e fechar conexões repetidamente.
- ✓ **Segurança:** JDBC pode ajudar a implementar práticas de segurança, como a utilização de prepared statements para prevenir ataques de SQL injection, ao separar a lógica da aplicação dos comandos SQL.
- ✓ **Transações:** Ele permite a gestão de transações, garantindo que múltiplas operações de banco de dados sejam executadas de forma consistente e confiável, mantendo a integridade dos dados.
- ✓ **Escalabilidade:** Middleware como JDBC permite que aplicações escalem de forma eficiente ao gerir conexões de forma otimizada e permitir que diferentes partes da aplicação acessem os recursos do banco de dados de forma coordenada.
- ✓ **Facilidade de Manutenção:** Ao centralizar a lógica de acesso ao banco de dados no middleware, facilita-se a manutenção do código, pois alterações na lógica de acesso ao banco de dados não necessitam de mudanças nas outras partes da aplicação.

Em suma, o uso de middleware como o JDBC é crucial para o desenvolvimento de aplicações robustas, escaláveis e seguras, proporcionando uma interface uniforme para acesso a dados e abstraindo a complexidade inerente à comunicação direta com os bancos de dados.

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

A principal diferença entre o uso de Statement e PreparedStatement na manipulação de dados em Java, utilizando JDBC, reside em aspectos de segurança, desempenho e funcionalidade. Aqui estão os principais pontos que diferenciam os dois:

“Statement”

- ✓ **Execução Simples e Direta:** É usado para executar instruções SQL estáticas, onde o comando SQL é passado como uma string.
- ✓ **Desempenho:** Cada vez que um Statement é executado, o SQL é analisado e compilado pelo banco de dados. Isso pode levar a uma sobrecarga, especialmente se a mesma instrução SQL for executada várias vezes.
- ✓ **Segurança:** O Statement é mais suscetível a ataques de SQL injection, pois os valores dos parâmetros são diretamente concatenados na string SQL.

“PreparedStatement”

- ✓ **Execução Pré-Compilada:** O comando SQL é pré-compilado e armazenado no banco de dados, e pode ser executado várias vezes com diferentes valores de parâmetros.
- ✓ **Desempenho:** O uso de PreparedStatement pode melhorar o desempenho, especialmente em operações repetitivas, porque a instrução SQL é analisada e compilada apenas uma vez.
- ✓ **Segurança:** PreparedStatement previne ataques de SQL injection, pois os valores dos parâmetros são definidos separadamente da instrução SQL. O JDBC lida automaticamente com a inserção de valores de maneira segura.
- ✓ **Uso de Parâmetros:** Permite o uso de parâmetros, que são representados por ? na string SQL. Esses parâmetros são definidos usando métodos como setString, setInt, etc.

Comparação Resumida

Característica	Statement	PreparedStatement
Execução	Direta e simples	Pré-compilada
Desempenho	Menor, especialmente em operações repetitivas	Melhor, devido à compilação única
Segurança	Suscetível a SQL injection	Previne SQL injection
Uso de Parâmetros	Não suporta	Suporta, usando placeholders (`?`)
Reutilização	Menos eficiente	Mais eficiente para comandos repetitivos

❖ Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (Data Access Object) é uma técnica de design amplamente utilizada para separar a lógica de acesso a dados da lógica de negócios em uma aplicação. Isso traz uma série de benefícios que contribuem significativamente para a manutenibilidade do software. Segue abaixo algumas das principais maneiras pelas quais o padrão DAO melhora a manutenibilidade:

✓ **Separação de Responsabilidades**

O DAO isola a lógica de acesso a dados da lógica de negócios. Isso significa que alterações na forma como os dados são armazenados ou recuperados (por exemplo, mudar de um banco de dados relacional para um banco de dados NoSQL) não afetam a lógica de negócios. Essa separação torna o código mais modular e mais fácil de entender e manter.

✓ **Facilidade de Mudanças e Extensões**

Com o padrão DAO, mudanças no esquema do banco de dados ou na tecnologia subjacente podem ser feitas apenas na camada DAO. A lógica de negócios e outras partes da aplicação permanecem inalteradas. Isso reduz significativamente o esforço necessário para realizar mudanças no sistema.

✓ **Reutilização de Código**

DAOs encapsulam toda a lógica de acesso a dados em um único lugar. Isso facilita a reutilização do código de acesso a dados em diferentes partes da aplicação ou mesmo em diferentes aplicações, sem duplicação de código.

✓ **Testabilidade**

A separação da lógica de acesso a dados facilita a escrita de testes unitários e testes de integração. Com o padrão DAO, é possível criar mocks ou stubs da camada de acesso a dados para testar a lógica de negócios sem depender de um banco de dados real. Isso aumenta a cobertura dos testes e ajuda a identificar bugs mais cedo.

✓ **Clareza e Organização do Código**

Usar o padrão DAO organiza o código de uma maneira mais clara e estruturada. Cada classe DAO é responsável por uma entidade ou um conjunto de operações relacionadas a dados, tornando o código mais legível e compreensível para os desenvolvedores.

✓ **Manutenção Simplificada**

Quando o acesso a dados é centralizado em DAOs, a manutenção do código torna-se mais fácil. Alterações, correções de bugs e otimizações podem ser feitas diretamente nas classes DAO, sem a necessidade de vasculhar o código da aplicação inteira para encontrar todas as instâncias onde o acesso a dados é realizado.

✓ **Melhor Abstração**

DAOs fornecem uma abstração clara entre a aplicação e o banco de dados. Os desenvolvedores que trabalham na lógica de negócios não precisam se preocupar com detalhes específicos do banco de dados, como consultas SQL ou transações, já que tudo isso é gerenciado pela camada DAO.

O padrão DAO é, portanto, uma prática recomendada para desenvolver aplicações robustas e fáceis de manter, proporcionando uma arquitetura mais limpa e organizada.

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

Quando se lida com um modelo estritamente relacional em banco de dados, a herança geralmente é refletida com uso de uma técnica conhecida como "tabelas de junção" (ou "tabelas de associação"). Esta abordagem é chamada de modelagem de herança de tabela única, ou modelagem de herança de tabela por classe. Eis como funciona:

- ✓ **Tabela Base (ou Superclasse):** Uma tabela é criada para representar a classe base ou superclasse. Esta tabela contém os atributos comuns a todas as subclasses.
- ✓ **Tabelas de Subclasse:** Para cada subclasse, é criada uma tabela separada contendo apenas os atributos específicos daquela subclasse. Essas tabelas também terão uma chave estrangeira que referencia a tabela base.
- ✓ **Chave Estrangeira:** A chave primária da tabela base é usada como chave estrangeira nas tabelas de subclasse para estabelecer a relação entre elas.
- ✓ **Junção de Tabelas:** Quando uma consulta é feita para recuperar dados de uma hierarquia de herança, é necessário fazer uma junção (JOIN) entre a tabela base e as tabelas de subclasse usando as chaves primárias e estrangeiras correspondentes.

Essas abordagens permitem que a hierarquia de herança seja representada de forma eficiente em um modelo relacional, de modo a manter a integridade referencial entre as tabelas e permitir consultas que recuperam dados de todas as classes relacionadas na hierarquia de herança.

Como programador, a escolha da estratégia de mapeamento de herança para um banco de dados relacional depende das necessidades da aplicação. Se precisamos de consultas rápidas e o banco de dados pode lidar com muitos valores NULL, a tabela única é atraente. Para manter um design limpo e evitar valores NULL, mas aceitando junções complexas, a tabela por subclasse é adequada. Se queremos uma implementação direta e não nos importamos com a duplicação, a tabela por classe concreta é a solução. Cada abordagem tem seus trade-offs e a decisão final deve considerar desempenho, manutenção e integridade dos dados.

2º Procedimento | Alimentando a Base

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 SQL Server na persistência de dados.

Todos os códigos solicitados neste roteiro de aula:

Criando a Classe CadastroBD

```
7 package cadastrobd;
8
9 import cadastrobd.modelo.PessoaFisica;
10 import cadastrobd.modelo.PessoaJuridica;
11 import cadastrobd.modelo.PessoaJuridicaDAO;
12 import java.util.Scanner;
13 import java.sql.SQLException;
14 import java.util.ArrayList;
15
16 /**
17  * @author Administrator
18  */
19 public class CadastroBD {
20
21     private static final Scanner sc = new Scanner(System.in);
22     private static final PessoaFisicaDAO pfDao = new PessoaFisicaDAO();
23     private static final PessoaJuridicaDAO pjDao = new PessoaJuridicaDAO();
24
25     public static void main(String[] args) {
26         int opcao = -1;
27         while (opcao != 0) {
28             printMenu();
29             opcao = inputInt("ESCOLHA: ");
30             switch (opcao) {
31                 case 1 -> {
32                     incluir();
33                 }
34                 case 2 -> {
35                     alterar();
36                 }
37                 case 3 -> {
38                     excluir();
39                 }
40                 case 4 -> {
38                     buscarPeloId();
39                 }
40                 case 5 -> {
41                     exibirTodos();
42                 }
43                 case 0 -> {
44                     System.out.println("Finalizando...");
45                     default -> {
46                         System.out.println("Escolha invalida!");
47                     }
48             }
49         }
50     }
51
52     private static void printMenu() {
53         System.out.println("=====");
54         System.out.println("1 - Incluir");
55         System.out.println("2 - Alterar");
56         System.out.println("3 - Excluir");
57         System.out.println("4 - Buscar pelo ID");
58         System.out.println("5 - Exibir todos");
59         System.out.println("0 - Sair");
60         System.out.println("=====");
61     }
62
63     private static String input(String prompt) {
64         System.out.print(prompt);
65         return sc.nextLine();
66     }
67
68     private static int inputInt(String prompt) {
69         System.out.print(prompt);
70         try {
71             return Integer.parseInt(sc.nextLine());
72         } catch (NumberFormatException e) {
73             System.out.println("Erro: Entrada invalida. Tente novamente.");
74             return inputInt(prompt);
75         }
76     }
77
78     private static void incluir() {
79         System.out.println("Incluindo pessoa...");
80         System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
81         String tipoPessoa = input("TIPO DE PESSOA: ").toLowerCase();
82         Integer id = input("Informe o ID: ");
83         switch (tipoPessoa) {
84             case "f" -> {
85                 try {
86                     pfDao.incluir(cadastrarPessoaFisica(id));
87                     System.out.println("Pessoa fisica incluida com sucesso!");
88                 } catch (SQLException e) {
89                     System.out.println("Erro ao incluir pessoa fisica: " + e.getMessage());
90                 }
91             }
92             case "j" -> {
93                 try {
94                     pjDao.incluir(cadastrarPessoaJuridica(id));
95                     System.out.println("Pessoa juridica incluida com sucesso!");
96                 } catch (SQLException e) {
97                     System.out.println("Erro ao incluir pessoa juridica: " + e.getMessage());
98                 }
99             }
100             default -> {
101                 System.out.println("Tipo de pessoa invalido!");
102             }
103         }
104     }
105
106     private static PessoaFisica cadastrarPessoaFisica(Integer id) {
107         System.out.println("Criando Pessoa Fisica...");
108         String nome = input("Informe o nome: ");
109         String logradouro = input("Informe o logradouro: ");
110         String cidade = input("Informe a cidade: ");
111         String estado = input("Informe o estado: ");
112         String telefone = input("Informe o telefone: ");
113         String email = input("Informe o email: ");
114         String cpf = input("Informe o CPF: ");
115         return new PessoaFisica(id, nome, logradouro, cidade, estado, telefone, email, cpf);
116     }
117
118     private static PessoaJuridica cadastrarPessoaJuridica(Integer id) {
119         System.out.println("Criando Pessoa Juridica...");
120         String nome = input("Informe o nome: ");
121         String logradouro = input("Informe o logradouro: ");
122         String cidade = input("Informe a cidade: ");
123         String estado = input("Informe o estado: ");
124         String telefone = input("Informe o telefone: ");
125         String email = input("Informe o email: ");
126         String cnpj = input("Informe o CNPJ: ");
127         return new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone, email, cnpj);
128     }
129
130     private static void alterar() {
131         System.out.println("Alterando pessoa...");
132         System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
133         String tipoPessoa = input("TIPO DE PESSOA: ").toLowerCase();
134         Integer id = input("Informe o ID da Pessoa Fisica: ");
135         PessoaFisica pf = pfDao.getPessoa(id);
136         if (pf != null) {
137             System.out.println("Dados atuais da Pessoa Fisica:");
138             pf.exibir();
139             String novoNome = input("Informe o novo nome: ");
140             String novoLogradouro = input("Informe o novo logradouro: ");
141             String novaCidade = input("Informe a nova cidade: ");
142             String novoEstado = input("Informe o novo estado: ");
143             String novoTelefone = input("Informe o novo telefone: ");
144             String novoEmail = input("Informe o novo email: ");
145             String novoCpf = input("Informe o novo CPF: ");
146             pfDao.alterar(pf, novoNome, novoLogradouro, novaCidade,
147                 novoEstado, novoTelefone, novoEmail, novoCpf);
148             System.out.println("Pessoa fisica alterada com sucesso!");
149         } else {
150             System.out.println("ID errado!");
151         }
152     } catch (NullPointerException | SQLException e) {
153         System.out.println("Erro ao alterar pessoa fisica: " + e.getMessage());
154     }
155 }
156
157 else if (tipoPessoa.equals("j")) {
158     Integer id = input("Informe o ID da Pessoa Juridica: ");
159     PessoaJuridica pj = pjDao.getPessoa(id);
160     if (pj != null) {
161         System.out.println("Dados atuais da Pessoa Juridica:");
162         pj.exibir();
163         String novoNome = input("Informe o novo nome: ");
164         String novoLogradouro = input("Informe o novo logradouro: ");
165         String novaCidade = input("Informe a nova cidade: ");
166         String novoEstado = input("Informe o novo estado: ");
167         String novoTelefone = input("Informe o novo telefone: ");
168         String novoEmail = input("Informe o novo email: ");
169         String novoCnpj = input("Informe o novo CNPJ: ");
170         pjDao.alterar(pj, novoNome, novoLogradouro, novaCidade,
171             novoEstado, novoTelefone, novoEmail, novoCnpj);
172         System.out.println("Pessoa juridica alterada com sucesso!");
173     } else {
174         System.out.println("ID errado!");
175     }
176 } catch (NullPointerException | SQLException e) {
177     System.out.println("Erro ao alterar pessoa juridica: " + e.getMessage());
178 }
179 }
180
181 } else {
182     System.out.println("Tipo de pessoa invalido!");
183 }
184
185 private static void excluir() {
186     System.out.println("Excluindo pessoa...");
187     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
188     String tipoPessoa = input("TIPO DE PESSOA: ").toLowerCase();
189     switch (tipoPessoa) {
190         case "f" -> {
191             Integer id = input("Informe o ID da Pessoa Fisica: ");
192             PessoaFisica pf = pfDao.getPessoa(id);
193             if (pf != null) {
194                 pfDao.excluir(pf.getId());
195                 System.out.println("Sucesso ao excluir!");
196             } else {
197                 System.out.println("ID errado!");
198             }
199         } catch (NullPointerException | SQLException e) {
200             System.out.println("Erro ao excluir pessoa fisica: " + e.getMessage());
201         }
202         case "j" -> {
203             Integer id = input("Informe o ID da Pessoa Juridica: ");
204             PessoaJuridica pj = pjDao.getPessoa(id);
205             if (pj != null) {
206                 pjDao.excluir(pj.getId());
207                 System.out.println("Sucesso ao excluir!");
208             } else {
209                 System.out.println("ID errado!");
210             }
211         } catch (NullPointerException | SQLException e) {
212             System.out.println("Erro ao excluir pessoa juridica: " + e.getMessage());
213         }
214         default -> {
215             System.out.println("Tipo de pessoa invalido!");
216         }
217     }
218 }
219
220 private static void buscarPeloId() {
221     System.out.println("Buscando pessoa pelo ID...");
222     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
223     String tipoPessoa = input("TIPO DE PESSOA: ").toLowerCase();
224     switch (tipoPessoa) {
225         case "f" -> {
226             Integer id = input("Informe o ID da Pessoa Fisica: ");
227             PessoaFisica pf = pfDao.getPessoa(id);
228             if (pf != null) {
229                 pf.exibir();
230             } else {
231                 System.err.println("Pessoa fisica com o ID " + id + " nao encontrada!");
232             }
233         } catch (SQLException e) {
234             System.err.println("Erro ao buscar pessoa fisica: " + e.getMessage());
235         }
236         case "j" -> {
237             Integer id = input("Informe o ID da Pessoa Juridica: ");
238             PessoaJuridica pj = pjDao.getPessoa(id);
239             if (pj != null) {
240                 pj.exibir();
241             } else {
242                 System.err.println("Pessoa juridica com o ID " + id + " nao encontrada!");
243             }
244         } catch (SQLException e) {
245             System.err.println("Erro ao buscar pessoa juridica: " + e.getMessage());
246         }
247         default -> {
248             System.out.println("Tipo de pessoa invalido!");
249         }
250     }
251 }
252
253 private static void exibirTodos() {
254     System.out.println("Exibindo todas as pessoas...");
255     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
256     String tipoPessoa = input("TIPO DE PESSOA: ").toLowerCase();
257     switch (tipoPessoa) {
258         case "f" -> {
259             ArrayList<PessoaFisica> listaPf = (ArrayList<PessoaFisica>) pfDao.getPessoas();
260             for (PessoaFisica pessoa : listaPf) {
261                 pessoa.exibir();
262             }
263         }
264         case "j" -> {
265             ArrayList<PessoaJuridica> listaPj = (ArrayList<PessoaJuridica>) pjDao.getPessoas();
266             for (PessoaJuridica pessoa : listaPj) {
267                 pessoa.exibir();
268             }
269         }
270         default -> {
271             System.out.println("Tipo de pessoa invalido!");
272         }
273     }
274 } catch (SQLException e) {
275     System.out.println("Erro ao exibir pessoas: " + e.getMessage());
276 }
277 }
```

“Descrição”

método **main** da classe principal do projeto, para implementação do cadastro em modo texto:

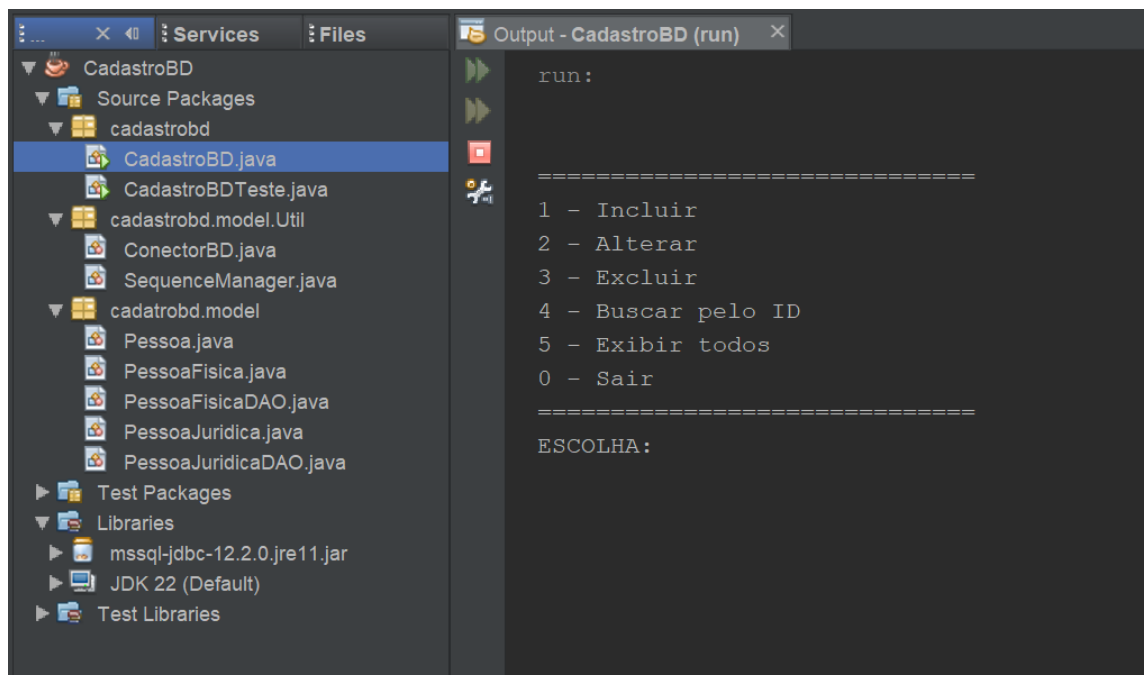
- Apresentar as opções do programa para o usuário, sendo 1 para incluir, 2 para alterar, 3 para excluir, 4 para exibir pelo id, 5 para exibir todos e 0 para finalizar a execução.

```

15  /**
16  *
17  * @author Administrador
18  */
19  public class CadastroBD {
20
21      private static final Scanner sc = new Scanner(System.in);
22      private static final PessoaFisicaDAO pfDao = new PessoaFisicaDAO();
23      private static final PessoaJuridicaDAO pjDao = new PessoaJuridicaDAO();
24
25      public static void main(String[] args) {
26          int opcao = -1;
27          while (opcao != 0) {
28              printMenu();
29              opcao = inputInt("ESCOLHA: ");
30              switch (opcao) {
31                  case 1 ->
32                      incluir();
33                  case 2 ->
34                      alterar();
35                  case 3 ->
36                      excluir();
37                  case 4 ->
38                      buscarPeloId();
39                  case 5 ->
40                      exibirTodos();
41                  case 0 ->
42                      System.out.println("Finalizando...");
43                  default ->
44                      System.out.println("Escolha invalida!");
45              }
46          }
47      }

```

Resultado:



- Selecionada a opção incluir, escolher o tipo (Física ou Jurídica), receber os dados a partir do teclado e adicionar no banco de dados através da classe DAO correta.

```

75 private static void incluir() {
76     System.out.println("\nIncluindo pessoa...");
77     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
78     String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
79     Integer id = inputInt("Informe o ID: ");
80     switch (tipoPessoa) {
81         case "F" -> {
82             try {
83                 pfDao.incluir(criarPessoaFisica(id));
84                 System.out.println("Pessoa fisica incluida com sucesso!");
85             } catch (SQLException e) {
86                 System.out.println("Erro ao incluir pessoa fisica: " + e.getMessage());
87             }
88         }
89         case "J" -> {
90             try {
91                 pjDao.incluir(criarPessoaJuridica(id));
92                 System.out.println("Pessoa juridica incluida com sucesso!");
93             } catch (SQLException e) {
94                 System.out.println("Erro ao incluir pessoa juridica: " + e.getMessage());
95             }
96         }
97         default ->
98             System.out.println("Tipo de pessoa invalido!");
99     }
100 }
101
102 private static PessoaFisica criarPessoaFisica(Integer id) {
103     System.out.println("Criando Pessoa Fisica...");
104     String nome = input("Informe o nome: ");
105     String logradouro = input("Informe o logradouro: ");
106     String cidade = input("Informe a cidade: ");
107     String estado = input("Informe o estado: ");
108     String telefone = input("Informe o telefone: ");
109     String email = input("Informe o email: ");
110     String cpf = input("Informe o CPF: ");
111     return new PessoaFisica(id, nome, logradouro, cidade, estado, telefone, email, cpf);
112 }
113
114 private static PessoaJuridica criarPessoaJuridica(Integer id) {
115     System.out.println("Criando Pessoa Juridica...");
116     String nome = input("Informe o nome: ");
117     String logradouro = input("Informe o logradouro: ");
118     String cidade = input("Informe a cidade: ");
119     String estado = input("Informe o estado: ");
120     String telefone = input("Informe o telefone: ");
121     String email = input("Informe o email: ");
122     String cnpj = input("Informe o CNPJ: ");
123     return new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone, email, cnpj);
124 }
125

```

Resultado:

=====	=====
1 - Incluir	1 - Incluir
2 - Alterar	2 - Alterar
3 - Excluir	3 - Excluir
4 - Buscar pelo ID	4 - Buscar pelo ID
5 - Exibir todos	5 - Exibir todos
0 - Sair	0 - Sair
=====	=====
ESCOLHA: 1	ESCOLHA: 1
 Incluindo pessoa...	 Incluindo pessoa...
F - Pessoa Fisica J - Pessoa Juridica	F - Pessoa Fisica J - Pessoa Juridica
TIPO DE PESSOA: F	TIPO DE PESSOA: J
Informe o ID: 6	Informe o ID: 6
Criando Pessoa Fisica...	Criando Pessoa Juridica...
Informe o nome: Helena	Informe o nome: bluecasasa
Informe o logradouro: Rua U 55	Informe o logradouro: Darlin Santos 55
Informe a cidade: Vila Velha	Informe a cidade: Vila Velha
Informe o estado: ES	Informe o estado: ES
Informe o telefone: 3344-55667	Informe o telefone: 8099-77665
Informe o email: helena@gmail.com	Informe o email: bluecasasa@gmail.com
Informe o CPF: 12345678911	Informe o CNPJ: 33233344455566
Pessoa fisica incluida com sucesso!	Pessoa juridica incluida com sucesso!

- Selecionada a opção alterar, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado, apresentar os dados atuais, solicitar os novos dados e alterar no banco de dados através do DAO.

```

126 private static void alterar() {
127     System.out.println("\nAlterando pessoa...");
128     System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
129     String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
130     if (tipoPessoa.equals("F")) {
131         try {
132             Integer id = inputInt("Informe o ID da Pessoa Física: ");
133             PessoaFisica pf = pfDao.getPessoa(id);
134             if (pf != null) {
135                 System.out.println("Dados atuais da Pessoa Física:");
136                 pf.exibir();
137
138                 String novoNome = input("Informe o novo nome: ");
139                 String novoLogradouro = input("Informe o novo logradouro: ");
140                 String novaCidade = input("Informe a nova cidade: ");
141                 String novoEstado = input("Informe o novo estado: ");
142                 String novoTelefone = input("Informe o novo telefone: ");
143                 String novoEmail = input("Informe o novo email: ");
144                 String novoCpf = input("Informe o novo CPF: ");
145
146                 pfDao.alterar(pf, novoNome, novoLogradouro, novaCidade,
147                     novoEstado, novoTelefone, novoEmail, novoCpf);
148                 System.out.println("Pessoa fisica alterada com sucesso!");
149             } else {
150                 System.out.println("ID errado!");
151             }
152         } catch (NullPointerException | SQLException e) {
153             System.out.println("Erro ao alterar pessoa fisica: " + e.getMessage());
154         }
155     } else if (tipoPessoa.equals("J")) {
156         try {
157             Integer id = inputInt("Informe o ID da Pessoa Jurídica: ");
158             PessoaJuridica pj = pjDao.getPessoa(id);
159             if (pj != null) {
160                 System.out.println("Dados atuais da Pessoa Jurídica:");
161                 pj.exibir();
162
163                 String novoNome = input("Informe o novo nome: ");
164                 String novoLogradouro = input("Informe o novo logradouro: ");
165                 String novaCidade = input("Informe a nova cidade: ");
166                 String novoEstado = input("Informe o novo estado: ");
167                 String novoTelefone = input("Informe o novo telefone: ");
168                 String novoEmail = input("Informe o novo email: ");
169                 String novoCnpj = input("Informe o novo CNPJ: ");
170
171                 pjDao.alterar(pj, novoNome, novoLogradouro, novaCidade,
172                     novoEstado, novoTelefone, novoEmail, novoCnpj);
173                 System.out.println("Pessoa juridica alterada com sucesso!");
174             } else {
175                 System.out.println("ID errado!");
176             }
177         } catch (NullPointerException | SQLException e) {
178             System.out.println("Erro ao alterar pessoa juridica: " + e.getMessage());
179         }
180     } else {
181         System.out.println("Tipo de pessoa invalido!");
182     }
183 }
184

```

Resultado:

```

1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair

=====
ESCOLHA: 2

Alterando pessoa...
F - Pessoa Física | J - Pessoa Jurídica
TIPO DE PESSOA: F
Informe o ID da Pessoa Física: 6
Dados atuais da Pessoa Física:
-----
ID: 6
Nome: Helena
Logradouro: Rua U 55
Cidade: Vila Velha
Estado: ES
Telefone: 3344-55667
Email: helena@gmail.com
CPF: 12345678911
Informe o novo nome: Heloisa
Informe o novo logradouro: Rua J 59
Informe a nova cidade: Vitoria
Informe o novo estado: ES
Informe o novo telefone: 9988-77889
Informe o novo email: heloisa@gmail.com
Informe o novo CPF: 1122233344
Pessoa fisica alterada com sucesso!

```

SQLQuery1.sql - USU...PF.Loja (loja (57)) * X código -3.sql - USU...OPF.Loja (loja (54))

SELECT *
FROM PessoaFisica
INNER JOIN Pessoa ON PessoaFisica.idPessoa = Pessoa.idPessoa

107 %
Messages

	idPessoa	cpf	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	11111111111	1	Ana	Rua T 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	22222222222	2	Bruno	Rua N 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	33333333333	3	Claudio	Rua W 40	Carapica	ES	3333-3333	claudio@gmail.com
4	6	1122233344	6	Heloisa	Rua J 59	Vitoria	ES	9988-77889	heloisa@gmail.com

- Selecionada a opção excluir, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e remover do banco de dados através do DAO.

```

184
185 private static void excluir() {
186     System.out.println("\nExcluindo pessoa...");
187     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
188     String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
189     switch (tipoPessoa) {
190         case "F" -> {
191             try {
192                 Integer id = inputInt("Informe o ID da Pessoa Fisica: ");
193                 PessoaFisica pf = pfDao.getPessoa(id);
194                 if (pf != null) {
195                     pfDao.excluir(pf.getId());
196                     System.out.println("Sucesso ao excluir!");
197                 } else {
198                     System.out.println("ID errado!");
199                 }
200             } catch (NullPointerException | SQLException e) {
201                 System.out.println("Erro ao excluir pessoa fisica: " + e.getMessage());
202             }
203         }
204         case "J" -> {
205             try {
206                 Integer id = inputInt("Informe o ID da Pessoa Juridica: ");
207                 PessoaJuridica pj = pjDao.getPessoa(id);
208                 if (pj != null) {
209                     pjDao.excluir(pj.getId());
210                     System.out.println("Sucesso ao excluir!");
211                 } else {
212                     System.out.println("ID errado!");
213                 }
214             } catch (NullPointerException | SQLException e) {
215                 System.out.println("Erro ao excluir pessoa juridica: " + e.getMessage());
216             }
217         }
218         default ->
219             System.out.println("Tipo de pessoa invalido!");
220     }
221 }
222

```

Resultado:

```

=====
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
=====
ESCOLHA: 3

Excluindo pessoa...
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: F
Informe o ID da Pessoa Fisica: 6
Pessoa fisica excluida com ID: 6
Sucesso ao excluir!

```

SQLQuery1.sql - USU...PF.Loja (loja (57))

SELECT *
FROM PessoaFisica
INNER JOIN Pessoa ON PessoaFisica.idPessoa = Pessoa.idPessoa

107 %

	idPessoa	cpf	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	111111111111	1	Ana	Rua T, 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	222222222222	2	Bruno	Rua N, 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	333333333333	3	Claudio	Rua W, 40	Cariacica	ES	3333-3333	claudio@gmail.com

- Selecionada a opção obter, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e apresentar os dados atuais, recuperados do banco através do DAO.

```

223 private static void buscarPeloId() {
224     System.out.println("\nBuscando pessoa pelo ID...");
225     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
226     String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
227     switch (tipoPessoa) {
228         case "F" -> {
229             try {
230                 Integer id = inputInt("Informe o ID da Pessoa Fisica: ");
231                 PessoaFisica pf = pfDao.getPessoa(id);
232                 if (pf != null) {
233                     pf.exibir();
234                 } else {
235                     System.err.println("Pessoa fisica com o ID " + id + " nao encontrada!");
236                 }
237             } catch (SQLException e) {
238                 System.err.println("Erro ao buscar pessoa fisica: " + e.getMessage());
239             }
240         }
241         case "J" -> {
242             try {
243                 Integer id = inputInt("Informe o ID da Pessoa Juridica: ");
244                 PessoaJuridica pj = pjDao.getPessoa(id);
245                 if (pj != null) {
246                     pj.exibir();
247                 } else {
248                     System.err.println("Pessoa juridica com o ID " + id + " nao encontrada!");
249                 }
250             } catch (SQLException e) {
251                 System.err.println("Erro ao buscar pessoa juridica: " + e.getMessage());
252             }
253         }
254         default ->
255             System.out.println("Tipo de pessoa invalido!");
256     }
257 }
258

```

Resultado:

```

1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
=====
ESCOLHA: 4

Buscando pessoa pelo ID...
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: J
Informe o ID da Pessoa Juridica: 4
-----
ID: 4
Nome: Distribuidora Vespor
Logradouro: Avenida J, 80
Cidade: Vitoria
Estado: ES
Telefone: 4444-4444
Email: vespord@gmail.com
CPF: 44444444444444

```

SQLQuery1.sql - USU...PF.Loja (loja (57)) * X código -3.sql - USU...OPF.Loja (loja (54))

```

SELECT *
FROM PessoaJuridica
INNER JOIN Pessoa ON PessoaJuridica.idPessoa = Pessoa.idPessoa

```

107 % Messages

	idPessoa	cnpj	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	4	4444444444444444	4	Distribuidora Vespor	Avenida J, 80	Vitoria	ES	4444-4444	vespord@gmail.com
2	5	5555555555555555	5	Empresa Vespor	Avenida H, 70	Vila Velha	ES	5555-5555	vesporv@gmail.com

- Selecionada a opção obterTodos, escolher o tipo (Física ou Jurídica) e apresentar os dados de todas as entidades presentes no banco de dados por intermédio do DAO.

```

259 private static void exibirTodos() {
260     System.out.println("\nExibindo todas as pessoas...");
261     System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
262     String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
263     try {
264         switch (tipoPessoa) {
265             case "F" -> {
266                 ArrayList<PessoaFisica> listaPf = (ArrayList<PessoaFisica>) pfDao.getPessoas();
267                 for (PessoaFisica pessoa : listaPf) {
268                     pessoa.exibir();
269                 }
270             }
271             case "J" -> {
272                 ArrayList<PessoaJuridica> listaPj = (ArrayList<PessoaJuridica>) pjDao.getPessoas();
273                 for (PessoaJuridica pessoa : listaPj) {
274                     pessoa.exibir();
275                 }
276             }
277             default ->
278                 System.out.println("Tipo de pessoa invalido!");
279         }
280     } catch (SQLException e) {
281         System.out.println("Erro ao exibir pessoas: " + e.getMessage());
282     }
283 }
284
285

```

Resultado:

```

=====
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
=====
ESCOLHA: 5

Exibindo todas as pessoas...
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: F
-----
ID: 1
Nome: Ana
Logradouro: Rua T, 20
Cidade: Vitoria
Estado: ES
Telefone: 1111-1111
Email: ana@gmail.com
CPF: 11111111111
-----
ID: 2
Nome: Bruno
Logradouro: Rua N, 30
Cidade: Santos
Estado: SP
Telefone: 2222-2222
Email: bruno@gmail.com
CPF: 22222222222
-----
ID: 3
Nome: Claudio
Logradouro: Rua W, 40
Cidade: Cariacica
Estado: ES
Telefone: 3333-3333
Email: claudio@gmail.com
CPF: 33333333333

```

The screenshot shows a SQL query window with the following SQL statement:

```
SELECT *
FROM PessoaFisica
INNER JOIN Pessoa ON PessoaFisica.idPessoa = Pessoa.idPessoa
```

The results pane shows the following data:

	idPessoa	cpf	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	11111111111	1	Ana	Rua T, 20	Vitoria	ES	1111-1111	ana@gmail.com
2	2	22222222222	2	Bruno	Rua N, 30	Santos	SP	2222-2222	bruno@gmail.com
3	3	33333333333	3	Claudio	Rua W, 40	Cariacica	ES	3333-3333	claudio@gmail.com

5- Análise e Conclusão:

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

A persistência de dados é fundamental para qualquer aplicação que precisa armazenar informações de forma duradoura. Existem várias abordagens para persistência, mas duas das mais comuns são a persistência em arquivo e a persistência em banco de dados. A escolha entre uma e outra depende dos requisitos específicos da aplicação. Aqui estão as principais diferenças entre elas:

Persistência em Arquivo

Descrição:

A persistência em arquivo envolve armazenar dados em arquivos no sistema de arquivos. Os formatos podem variar, incluindo texto simples (CSV, JSON, XML) ou binário.

Persistência em Banco de Dados

Descrição:

A persistência em banco de dados envolve armazenar dados em sistemas de gerenciamento de banco de dados (SGBD), como MySQL, PostgreSQL, Oracle, ou bancos de dados NoSQL como MongoDB.

Comparação Resumida

Aspecto	Persistência em Arquivo	Persistência em Banco de Dados
Simplicidade	Fácil de implementar e entender	Requer configuração e manutenção de um SGBD
Desempenho	Menos eficiente para grandes volumes de dados	Otimizado para grandes volumes e operações complexas
Concorrência	Difícil de gerenciar	Gerenciado pelo SGBD
Escalabilidade	Menos adequado para grandes volumes	Adequado para aplicações escaláveis
Consultas	Limitadas a operações simples	Linguagens de consulta poderosas como SQL
Segurança	Proteção manual e menos robusta	Mecanismos robustos de segurança integrados
Portabilidade	Fácil de mover entre sistemas	Depende da configuração e compatibilidade do SGBD

A escolha entre persistência em arquivo e persistência em banco de dados deve ser baseada nas necessidades específicas da aplicação, considerando aspectos como desempenho, complexidade, segurança e escalabilidade.

❖ 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 nas versões mais recentes do Java, introduzido no Java 8, simplificou significativamente várias tarefas de programação, incluindo a impressão dos valores contidos nas entidades. Antes da introdução dos lambdas, os programadores precisavam utilizar classes anônimas ou loops explícitos para realizar operações sobre coleções de dados. Com lambdas, essas operações tornaram-se mais concisas e legíveis.

Antes dos Lambdas: Versão Tradicional

Antes do Java 8, imprimir valores contidos em uma lista de entidades exigia a criação de loops explícitos.

Com Lambdas: Versão Simplificada

Com a introdução dos lambdas e da API de Streams em Java 8, a impressão dos valores contidos nas entidades pode ser feita de forma muito mais concisa.

Vantagens do Uso de Lambdas

Concisão: Os lambdas reduzem a quantidade de código necessário para operações comuns, como iterar e imprimir elementos de uma coleção.

Legibilidade: Código mais curto e expressivo, que se concentra no que precisa ser feito, em vez de como fazer.

Funcionalidade: A combinação de lambdas com a API de Streams permite operações poderosas e expressivas sobre coleções de dados.

Exemplo Avançado com Filtros e Map

Os lambdas e Streams também permitem realizar operações mais complexas de forma concisa.

Conclusão

Os operadores lambda, junto com a API de Streams, proporcionam uma maneira mais elegante, concisa e poderosa de manipular coleções de dados em Java. Eles simplificam operações comuns, como imprimir valores contidos em entidades, tornando o código mais legível e fácil de manter. A adoção desses recursos é altamente recomendada para melhorar a eficiência e clareza do código Java moderno.

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


Em Java, métodos acionados diretamente pelo método main sem o uso de um objeto precisam ser marcados como static devido à maneira como o método main é definido e como a linguagem Java lida com métodos de classe versus métodos de instância.

Explicação Técnica

Método main

O ponto de entrada para uma aplicação Java é o método main, definido geralmente assim:

java

 Copiar código

```
public static void main(String[] args) {  
    // Código aqui  
}
```

- ✓ **public:** O método main precisa ser público para que a JVM possa acessá-lo.
- ✓ **static:** O método main é estático porque ele precisa ser chamado pela JVM sem a necessidade de criar uma instância da classe.
- ✓ **void:** O método main não retorna nenhum valor.
- ✓ **String[] args:** Parâmetros de linha de comando passados para a aplicação.

Métodos Estáticos

Métodos marcados como static pertencem à classe em si, e não a qualquer instância específica da classe. Isso significa que podem ser chamados diretamente usando o nome da classe, sem a necessidade de criar um objeto da classe.

java

 Copiar código

```
public class MinhaClasse {  
    public static void meuMetodoEstatico() {  
        System.out.println("Método estático chamado");  
    }  
}
```

Para chamar meuMetodoEstatico a partir do método main, fazemos:

```
java Copiar código

public class Main {
    public static void main(String[] args) {
        MinhaClasse.meuMetodoEstatico();
    }
}
```

Métodos de Instância

Métodos que não são marcados como static são métodos de instância e requerem que você crie uma instância da classe antes de chamá-los:

```
java Copiar código

public class MinhaClasse {
    public void meuMetodoDeInstancia() {
        System.out.println("Método de instância chamado");
    }
}
```

Para chamar meuMetodoDeInstancia a partir do método main, fazemos:

```
java Copiar código

public class Main {
    public static void main(String[] args) {
        MinhaClasse minhaInstancia = new MinhaClasse();
        minhaInstancia.meuMetodoDeInstancia();
    }
}
```

Por que Métodos Chamados Diretamente pelo main Precisam Ser static

- ✓ **Acesso Direto Sem Instância:** O método main é estático e pertence à classe, não a uma instância. Como tal, ele só pode chamar outros métodos estáticos diretamente, sem criar uma instância da classe.
- ✓ **Inicialização:** Quando a JVM inicia uma aplicação, ela chama o método main sem criar uma instância da classe. Portanto, qualquer método que main chama diretamente também precisa ser acessível sem uma instância.
- ✓ **Contexto de Classe:** Métodos estáticos operam no contexto da classe e não têm acesso direto a variáveis de instância. Isso reflete o design do main, que é o ponto de entrada e deve ser capaz de executar sem conhecimento ou dependência do estado de qualquer instância de objeto.

Exemplo Prático

```
java Copiar código  
  
public class Exemplo {  
    public static void metodoEstatico() {  
        System.out.println("Chamado método estático");  
    }  
  
    public void metodoDeInstancia() {  
        System.out.println("Chamado método de instância");  
    }  
  
    public static void main(String[] args) {  
        // Chamando método estático diretamente  
        metodoEstatico();  
  
        // Para chamar o método de instância, precisamos criar uma instância da classe  
        Exemplo exemplo = new Exemplo();  
        exemplo.metodoDeInstancia();  
    }  
}
```

Conclusão

Métodos acionados diretamente pelo método main precisam ser marcados como static porque main é estático e não opera em uma instância da classe. Métodos estáticos pertencem à classe como um todo, enquanto métodos de instância requerem um objeto específico. Este design permite que o método main inicie a execução de uma aplicação de forma independente, sem a necessidade de estado ou contexto de instância.