



Estácio

FACULDADE ESTÁCIO

CÂMPUS VOLTA REDONDA – RJ

DESENVOLVIMENTO FULL STACK

DISCIPLINA – BACK-END SEM BANCO NÃO TEM

TURMA – 2023.2

SEMESTRE – 3

VOLTA REDONDA, AGOSTO 2024.

DESENVOLVIMENTO FULL STACK

DISCIPLINA – BACK-END SEM BANCO NÃO TEM

TURMA – 2023.2

SEMESTRE – 3

ALUNO – BRUNO SAMPAIO BASTOS

TUTOR – SIMONE INGRID MONTEIRO

GITHUB - <https://github.com/BrunoTI-Code?tab=repositories>

VOLTA REDONDA, AGOSTO 2024.

1 1 PROCEDIMENTO | MAPEAMENTO OBJETO-RELACIONAL E DAO

1.1 OBJETIVO DA PRÁTICA

Implementar persistência com base no middleware JDBC.

Utilizar o padrão DAO (Data Access Object) no manuseio de dados.

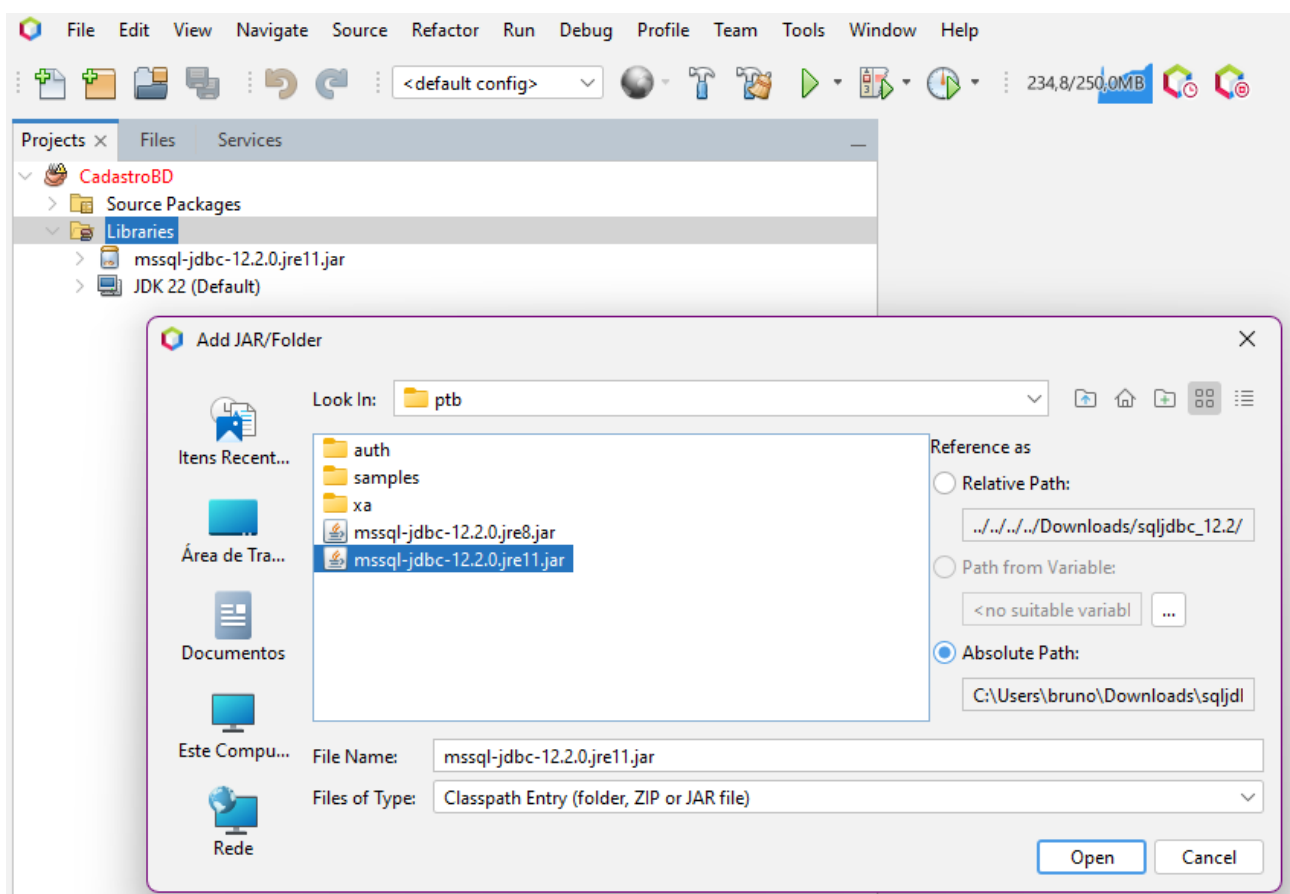
Implementar o mapeamento objeto-relacional em sistemas Java.

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.

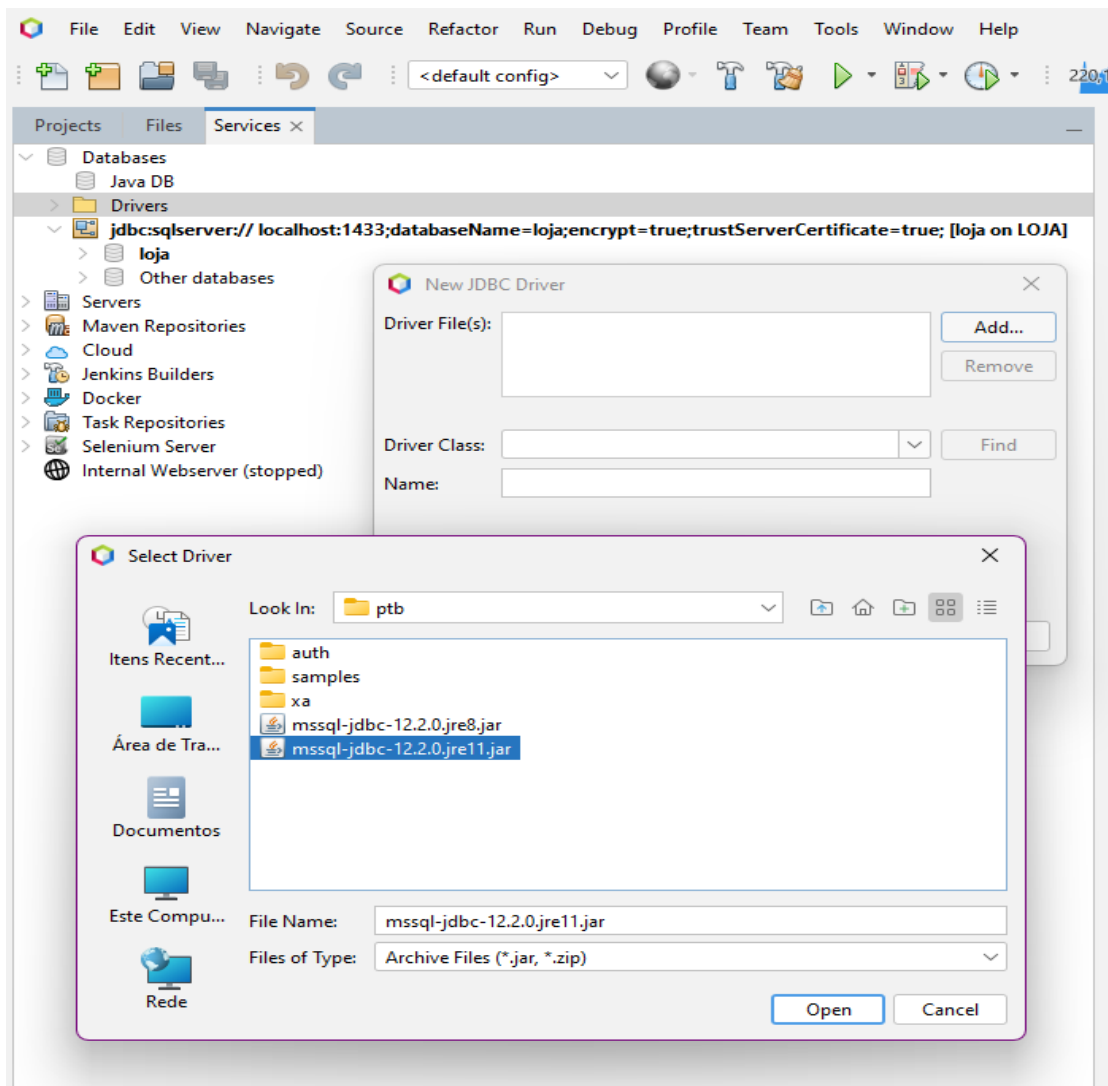
2 CRIAR O PROJETO E CONFIGURAR AS BIBLIOTECAS NECESSÁRIAS:

- 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*.



3 CONFIGURAR O ACESSO AO BANCO PELA ABA DE SERVIÇOS DO NETBEANS.

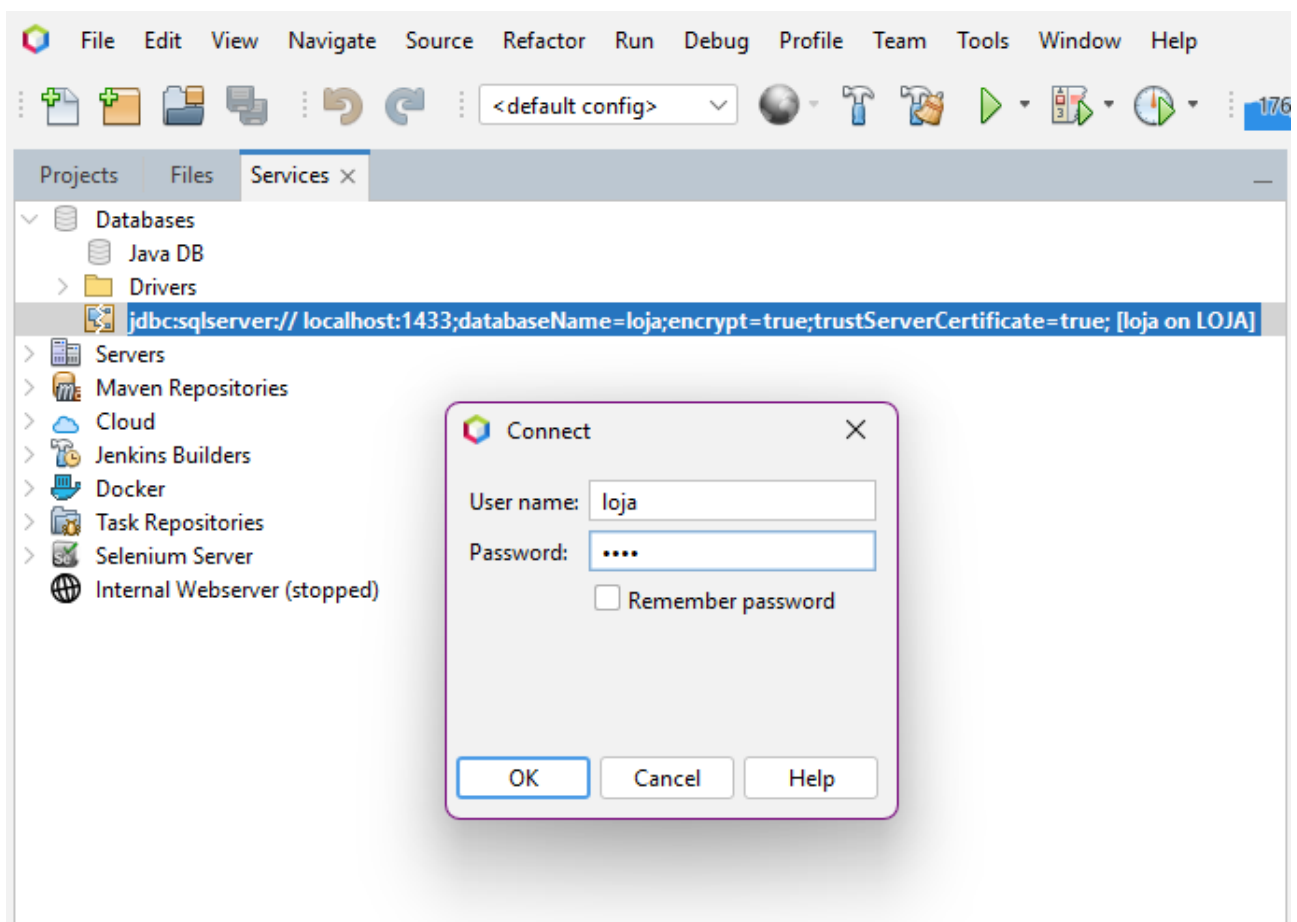
- Na aba de Serviços, divisão Banco de Dados, clique com o botão direito em Drivers e escolha Novo Driver.
- Na janela que se abrirá, clicar em Add (Adicionar), escolher o arquivo jar utilizado no passo anterior e finalizar com Ok.
- O reconhecimento será automático, e podemos definir uma conexão com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



• 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.

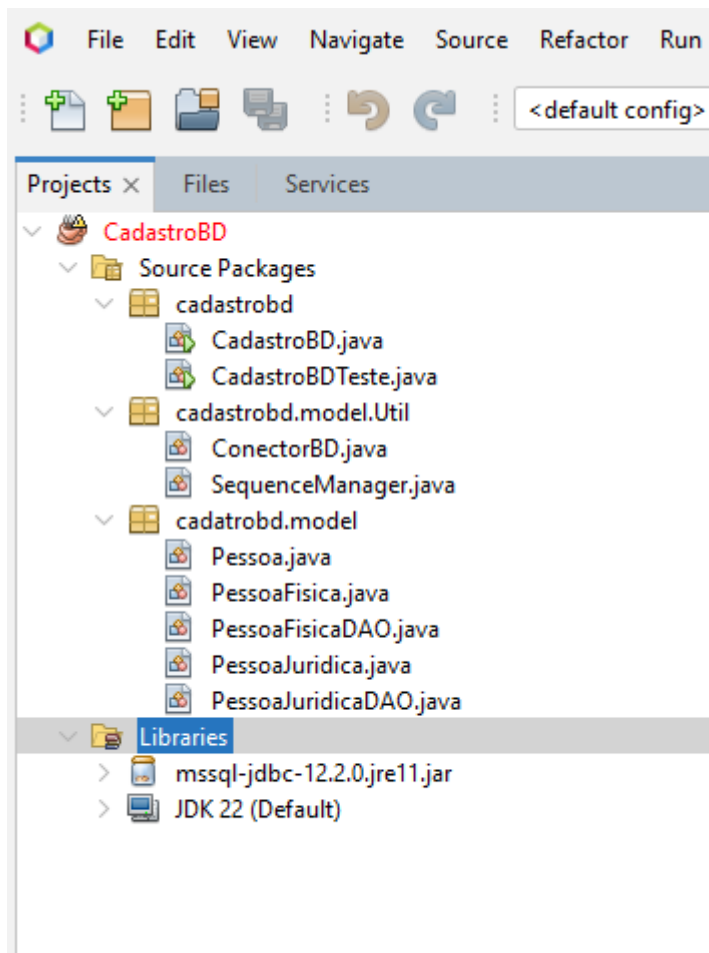
• Utilizar o clique do botão direito sobre as tabelas, e escolher Visualizar Dados (View Data), para consultar os dados atualmente no banco.

The screenshot shows the Apache NetBeans IDE interface. On the left, the 'Projects' pane displays a tree structure of database objects under the 'jdbc:sqlserver://localhost:1433;databaseName=lojencrypt-truetrustServerCertificate=true;loja on L...' connection. The 'Tables' folder is expanded, showing a list of tables including 'Movimento', 'Pessoa', 'PessoaFisica', 'PessoaJuridica', 'Produto', 'Usuario', 'Views', 'Procedures', 'guest', 'INFORMATION_SCHEMA', 'sys', and 'Other databases'. The 'Pessoa' table is selected. In the center, the 'SQL 1' editor shows the query: `SELECT TOP 100 * FROM dbo.Pessoa;`. Below the editor, the 'SELECT TOP 100 * FROM dbo.Pessoa' window displays a table with 5 rows and 8 columns: #, idPessoa, nome, logradouro, cidade, estado, telefone, and email. The data is as follows:

#	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Natalia	Rua L 50	Angra dos Reis	RJ	1111-1111	Natalia@gmail.com
2	2	Bruno	Rua P 10	Volta Redonda	RJ	2222-2222	Bruno@gmail.com
3	3	Jonas	Rua S 80	Vitoria	ES	3333-3333	Jonas@gmail.com
4	4	Distribuidora Renova	Avenida J 80	São Paulo	SP	4444-4444	RenovaJat@gmail.com
5	5	Empresa Jasper	Avenida H 70	São Paulo	SP	5555-5555	Jasper@gmail.com

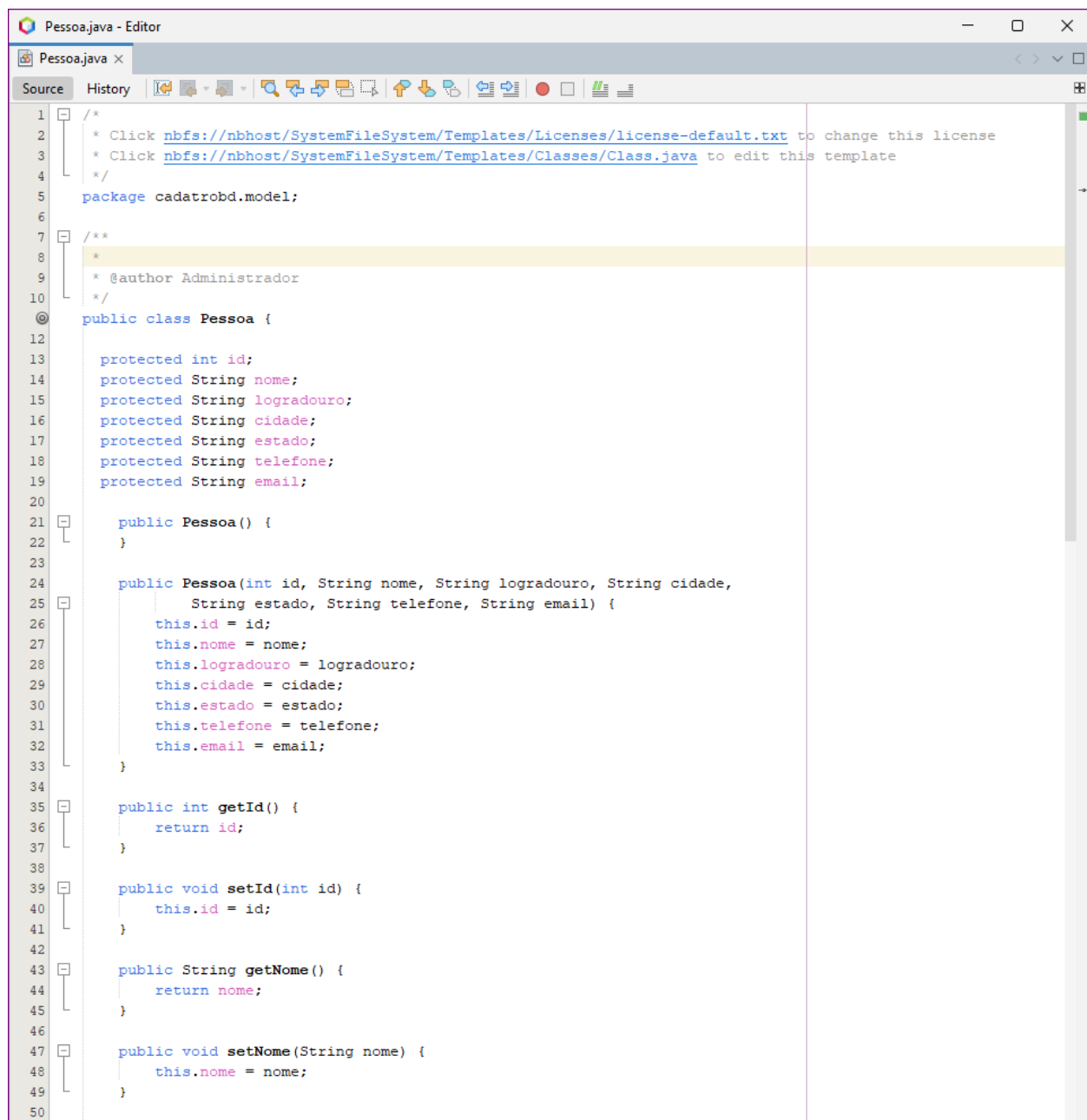
At the bottom, the 'Output - SQL 1 execution' window shows the message: 'Execution finished after 0,044 s, no errors occurred.'

4 TODOS OS CÓDIGOS SOLICITADOS NESTE ROTEIRO DE AULA:



4.1 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 to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
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       public Pessoa(int id, String nome, String logradouro, String cidade,
25           String estado, String telefone, String email) {
26           this.id = id;
27           this.nome = nome;
28           this.logradouro = logradouro;
29           this.cidade = cidade;
30           this.estado = estado;
31           this.telefone = telefone;
32           this.email = email;
33       }
34
35       public int getId() {
36           return id;
37       }
38
39       public void setId(int id) {
40           this.id = id;
41       }
42
43       public String getNome() {
44           return nome;
45       }
46
47       public void setNome(String nome) {
48           this.nome = nome;
49       }
50   }
```

Pessoa.java - Editor

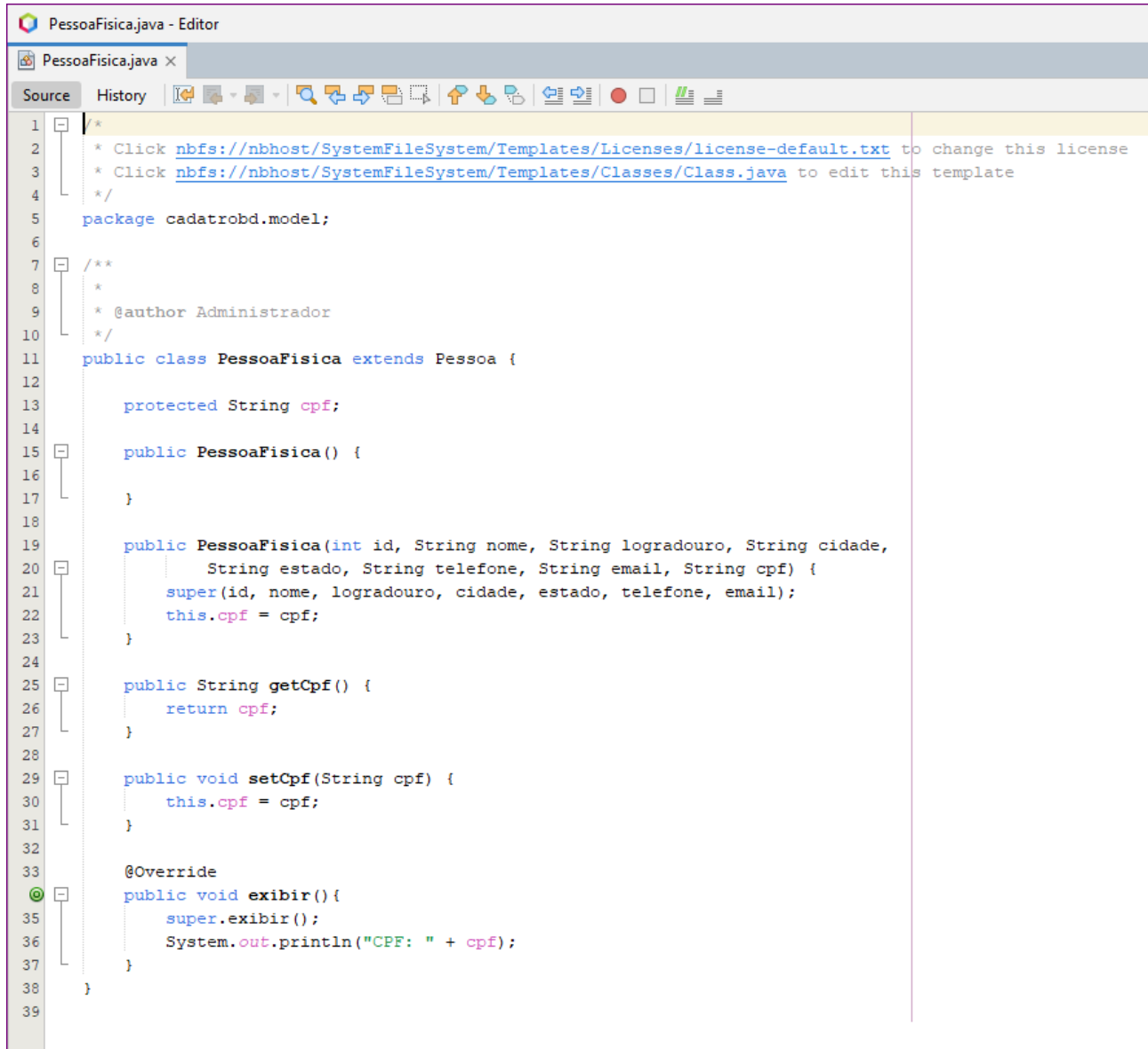
Pessoa.java x

Source History

```
50
51 public String getLogradouro() {
52     return logradouro;
53 }
54
55 public void setLogradouro(String logradouro) {
56     this.logradouro = logradouro;
57 }
58
59 public String getCidade() {
60     return cidade;
61 }
62
63 public void setCidade(String cidade) {
64     this.cidade = cidade;
65 }
66
67 public String getEstado() {
68     return estado;
69 }
70
71 public void setEstado(String estado) {
72     this.estado = estado;
73 }
74
75 public String getTelefone() {
76     return telefone;
77 }
78
79 public void setTelefone(String telefone) {
80     this.telefone = telefone;
81 }
82
83 public String getEmail() {
84     return email;
85 }
86
87 public void setEmail(String email) {
88     this.email = email;
89 }
90
91 public void exibir(){
92     System.out.println("-----");
93     System.out.println("ID: " + id);
94     System.out.println("Nome: " + nome);
95     System.out.println("Logradouro: " + logradouro);
96     System.out.println("Cidade: " + cidade);
97     System.out.println("Estado: " + estado);
98     System.out.println("Telefone: " + telefone);
99     System.out.println("Email: " + email);
100 }
```

cadatrobd.model.Pessoa >

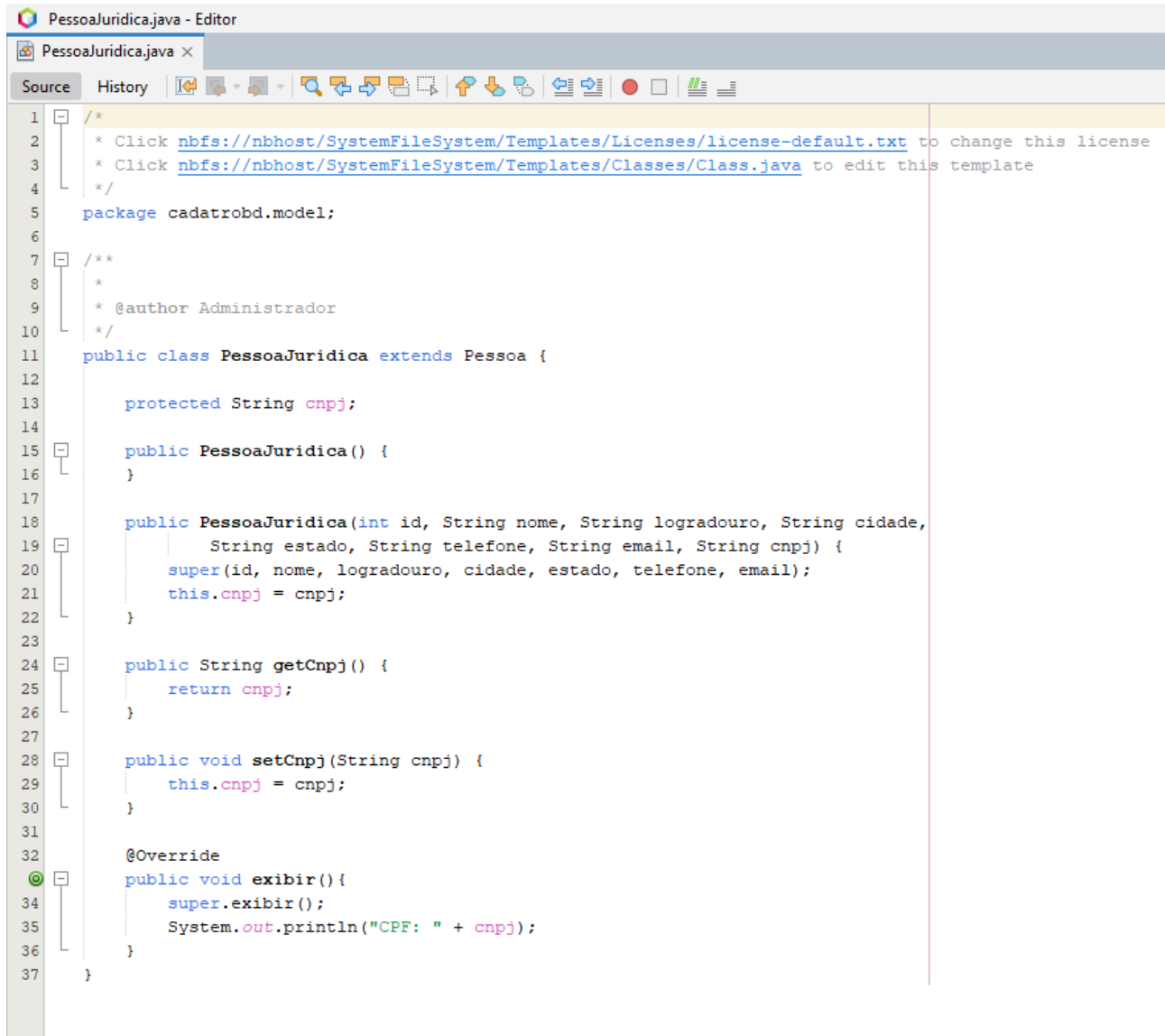
Criando a Classe Pessoa Fisica:



The screenshot shows an IDE window titled "PessoaFisica.java - Editor". The editor displays the following Java code:

```
1  /**
2   * Click nbfs:///nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs:///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   }
39
```

Criando a Classe Pessoa Juridica:

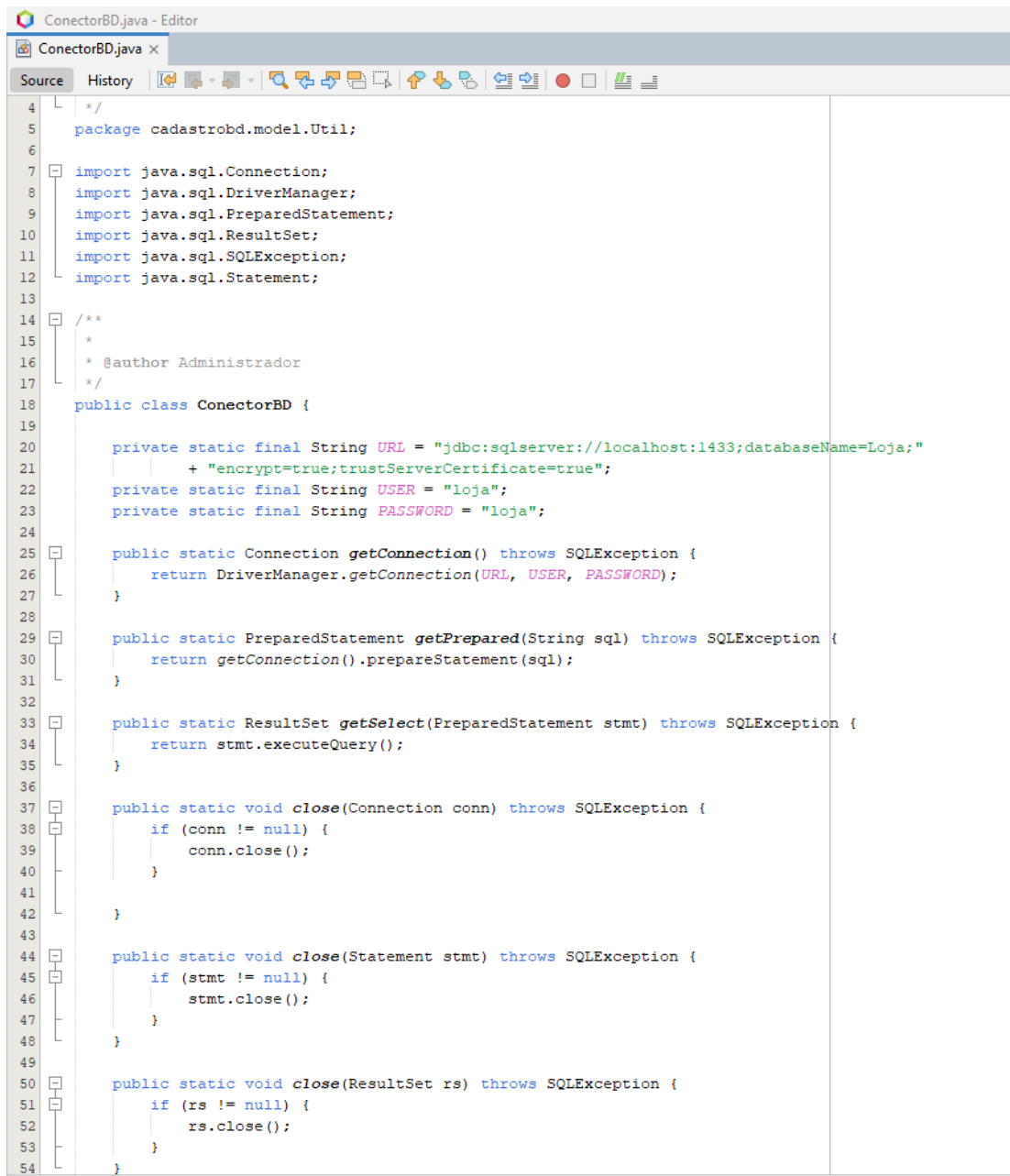


The screenshot shows a Java IDE window titled "PessoaJuridica.java - Editor". The editor contains the following code:

```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://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 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   }
```

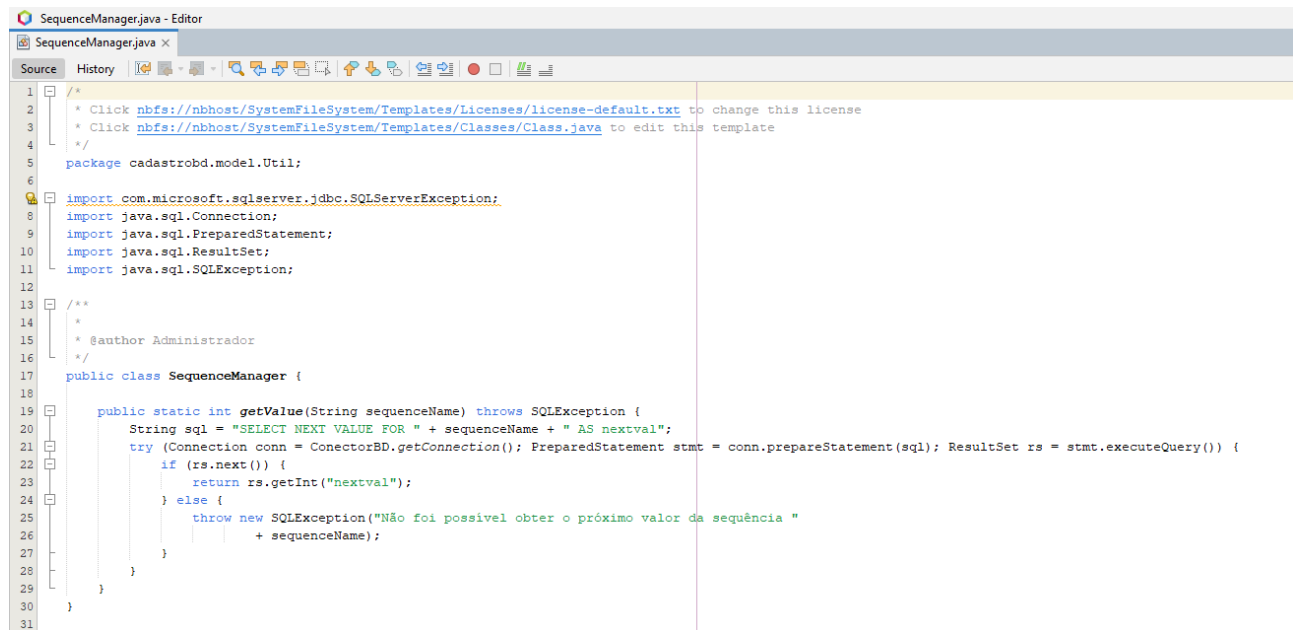
4.2 CRIAR OS PACOTES CADASTRO.MODEL.UTIL, PARA INCLUSÃO DAS CLASSES UTILITÁRIAS QUE SÃO APRESENTADAS A SEGUIR:

Criando a Classe ConectorBD:



```
ConectorBD.java - Editor
ConectorBD.java x
Source History
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 }
```

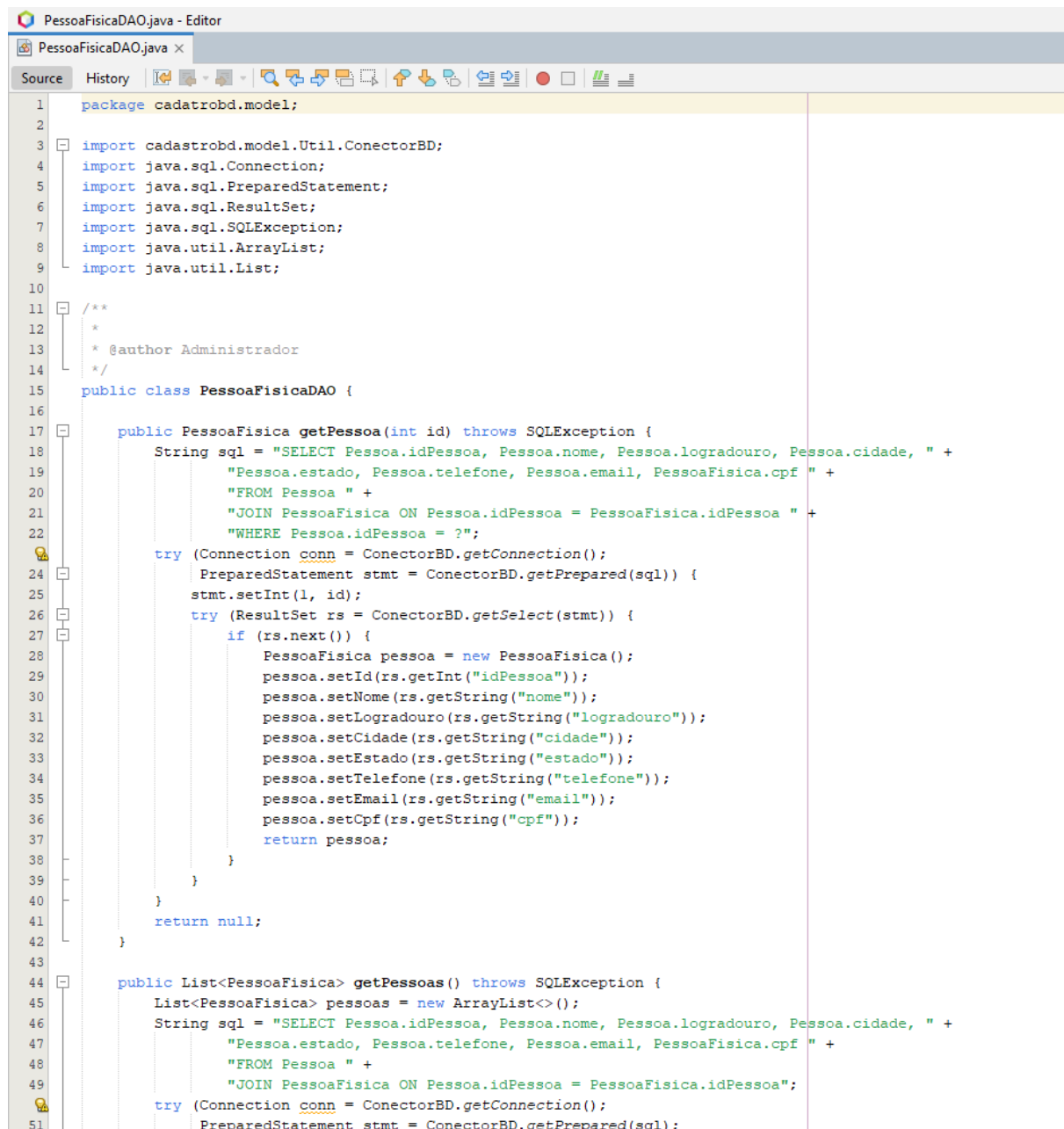
Criando a Classe SequenceManager:



```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastrdbd.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(); PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
22             if (rs.next()) {
23                 return rs.getInt("nextval");
24             } else {
25                 throw new SQLException("Não foi possível obter o próximo valor da sequência "
26                     + sequenceName);
27             }
28         }
29     }
30 }
31
```

4.3 CODIFICAR AS CLASSES NO PADRÃO DAO, NO PACOTE CADASTRO.MODEL.

Criando a Classe PessoaFisicaDAO:



```
1 package cadastrobd.model;
2
3 import cadastrobd.model.Util.ConectorBD;
4 import java.sql.Connection;
5 import java.sql.PreparedStatement;
6 import java.sql.ResultSet;
7 import java.sql.SQLException;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 /**
12  *
13  * @author Administrador
14  */
15 public class PessoaFisicaDAO {
16
17     public PessoaFisica getPessoa(int id) throws SQLException {
18         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
19             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaFisica.cpf " +
20             "FROM Pessoa " +
21             "JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa " +
22             "WHERE Pessoa.idPessoa = ?";
23         try (Connection conn = ConectorBD.getConnection();
24             PreparedStatement stmt = ConectorBD.getPrepared(sql)) {
25             stmt.setInt(1, id);
26             try (ResultSet rs = ConectorBD.getSelect(stmt)) {
27                 if (rs.next()) {
28                     PessoaFisica pessoa = new PessoaFisica();
29                     pessoa.setId(rs.getInt("idPessoa"));
30                     pessoa.setNome(rs.getString("nome"));
31                     pessoa.setLogradouro(rs.getString("logradouro"));
32                     pessoa.setCidade(rs.getString("cidade"));
33                     pessoa.setEstado(rs.getString("estado"));
34                     pessoa.setTelefone(rs.getString("telefone"));
35                     pessoa.setEmail(rs.getString("email"));
36                     pessoa.setCpf(rs.getString("cpf"));
37                     return pessoa;
38                 }
39             }
40         }
41         return null;
42     }
43
44     public List<PessoaFisica> getPessoas() throws SQLException {
45         List<PessoaFisica> pessoas = new ArrayList<>();
46         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
47             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaFisica.cpf " +
48             "FROM Pessoa " +
49             "JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa";
50         try (Connection conn = ConectorBD.getConnection();
51             PreparedStatement stmt = ConectorBD.getPrepared(sql);
```

```

Source History
49      "JOIN PessoaFisica ON Pessoa.idPessoa = PessoaFisica.idPessoa";
50  try (Connection conn = ConectorBD.getConnection();
51      PreparedStatement stmt = ConectorBD.getPrepared(sql);
52      ResultSet rs = ConectorBD.getSelect(stmt)) {
53      while (rs.next()) {
54          PessoaFisica pessoa = new PessoaFisica();
55          pessoa.setId(rs.getInt("idPessoa"));
56          pessoa.setNome(rs.getString("nome"));
57          pessoa.setLogradouro(rs.getString("logradouro"));
58          pessoa.setCidade(rs.getString("cidade"));
59          pessoa.setEstado(rs.getString("estado"));
60          pessoa.setTelefone(rs.getString("telefone"));
61          pessoa.setEmail(rs.getString("email"));
62          pessoa.setCpf(rs.getString("cpf"));
63          pessoas.add(pessoa);
64      }
65  }
66  return pessoas;
67  }
68
69  public void incluir(PessoaFisica pessoa) throws SQLException {
70      String sqlInsertPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade,"
71      + " estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
72      String sqlInsertPessoaFisica = "INSERT INTO PessoaFisica (idPessoa, cpf)"
73      + " VALUES (?, ?)";
74      try (Connection conn = ConectorBD.getConnection();
75          PreparedStatement stmtInsertPessoa = conn.prepareStatement(sqlInsertPessoa);
76          PreparedStatement stmtInsertPessoaFisica = conn.prepareStatement(sqlInsertPessoaFisica)) {
77
78          // Inserir na tabela Pessoa
79          stmtInsertPessoa.setInt(1, pessoa.getId());
80          stmtInsertPessoa.setString(2, pessoa.getNome());
81          stmtInsertPessoa.setString(3, pessoa.getLogradouro());
82          stmtInsertPessoa.setString(4, pessoa.getCidade());
83          stmtInsertPessoa.setString(5, pessoa.getEstado());
84          stmtInsertPessoa.setString(6, pessoa.getTelefone());
85          stmtInsertPessoa.setString(7, pessoa.getEmail());
86          stmtInsertPessoa.executeUpdate();
87
88          // Inserir na tabela PessoaFisica
89          stmtInsertPessoaFisica.setInt(1, pessoa.getId());
90          stmtInsertPessoaFisica.setString(2, pessoa.getCpf());
91          stmtInsertPessoaFisica.executeUpdate();
92      }
93  }
94
95  public void alterar(PessoaFisica pessoa, String novoNome, String novoLogradouro, String novaCidade,
96  String novoEstado, String novoTelefone, String novoEmail, String novoCpf) throws SQLException {
97      String sql = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?,"
98      + " estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
99      String sqlFisica = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa = ?";

```



```
PessoaFisicaDAO.java - Editor
PessoaFisicaDAO.java x
Source History

94
95 public void alterar(PessoaFisica pessoa, String novoNome, String novoLogradouro, String novaCidade,
96 String novoEstado, String novoTelefone, String novoEmail, String novoCpf) throws SQLException {
97 String sql = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, "
98 + " estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
99 String sqlFisica = "UPDATE PessoaFisica SET cpf = ? WHERE idPessoa = ?";
100
101 try (Connection conn = ConectorBD.getConnection();
102 PreparedStatement stmt = conn.prepareStatement(sql);
103 PreparedStatement stmtFisica = conn.prepareStatement(sqlFisica)) {
104
105 // Atualizar dados na tabela Pessoa
106 stmt.setString(1, novoNome);
107 stmt.setString(2, novoLogradouro);
108 stmt.setString(3, novaCidade);
109 stmt.setString(4, novoEstado);
110 stmt.setString(5, novoTelefone);
111 stmt.setString(6, novoEmail);
112 stmt.setInt(7, pessoa.getId());
113 stmt.executeUpdate();
114
115 // Atualizar CPF na tabela PessoaFisica
116 stmtFisica.setString(1, novoCpf);
117 stmtFisica.setInt(2, pessoa.getId());
118 stmtFisica.executeUpdate();
119 }
120 }
121
122 public void excluir(int id) throws SQLException {
123 String sql = "DELETE FROM PessoaFisica WHERE idPessoa = ?";
124 String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
125
126 try (Connection conn = ConectorBD.getConnection();
127 PreparedStatement stmt = conn.prepareStatement(sql);
128 PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {
129 // Excluir da tabela PessoaFisica
130 stmt.setInt(1, id);
131 stmt.executeUpdate();
132
133 // Excluir da tabela Pessoa
134 stmtPessoa.setInt(1, id);
135 stmtPessoa.executeUpdate();
136
137 System.out.println("Pessoa fisica excluida com ID: " + id);
138 }
139 }
140 }
141
```

Criando a Classe PessoaJuridicaDAO:

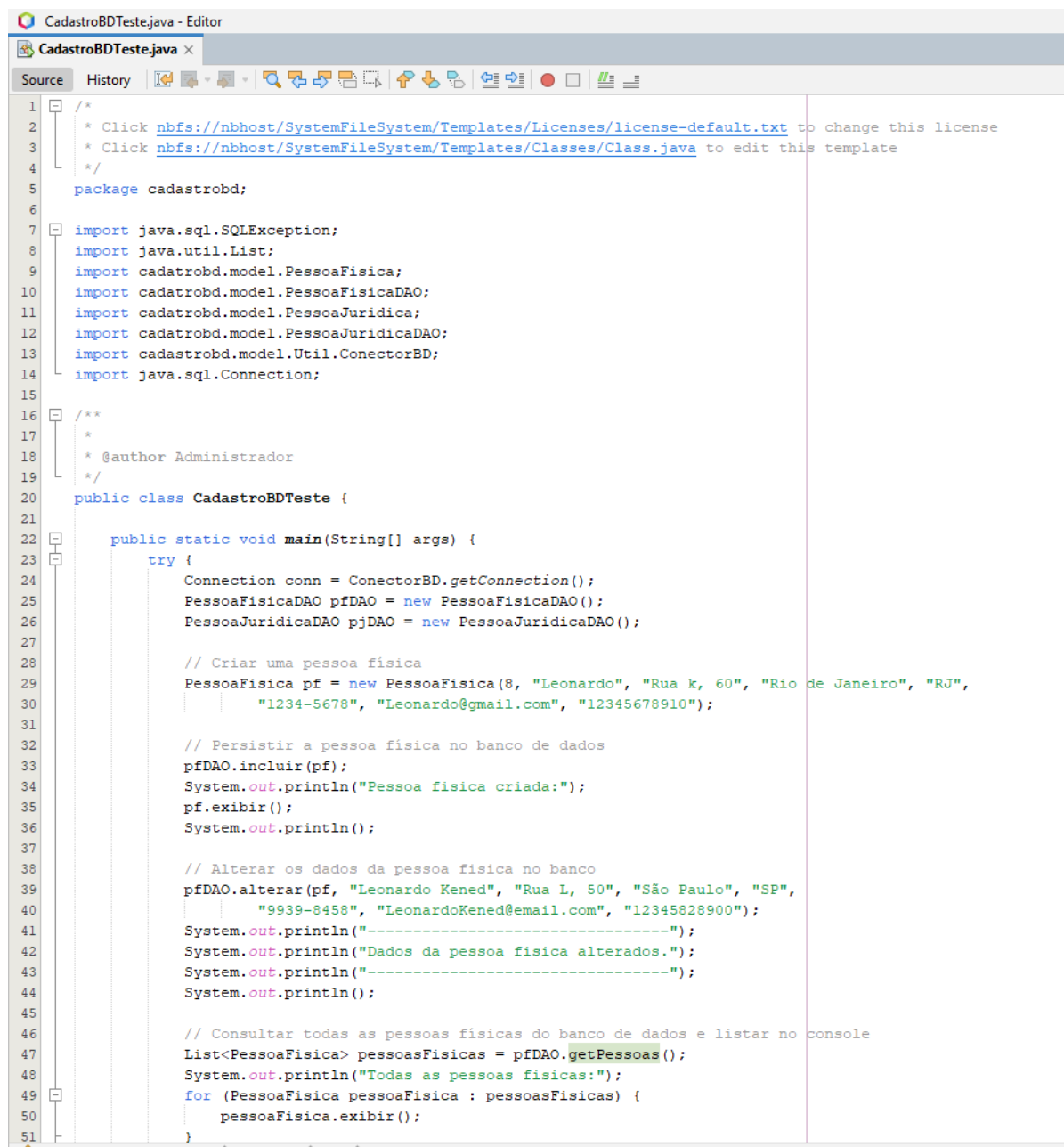
```
PessoaJuridicaDAO.java - Editor
PessoaJuridicaDAO.java x
Source History
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package cadastrobd.model;
6
7  import cadastrobd.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 *
17 * @author Administrador
18 */
19 public class PessoaJuridicaDAO {
20
21     public PessoaJuridica getPessoa(int id) throws SQLException {
22         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
23             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaJuridica.cnpj " +
24             "FROM Pessoa " +
25             "JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa " +
26             "WHERE Pessoa.idPessoa = ?";
27
28         try (Connection conn = ConectorBD.getConnection();
29             PreparedStatement stmt = ConectorBD.getPrepared(sql)) {
30             stmt.setInt(1, id);
31             try (ResultSet rs = ConectorBD.getSelect(stmt)) {
32                 if (rs.next()) {
33                     PessoaJuridica pessoa = new PessoaJuridica();
34                     pessoa.setId(rs.getInt("idPessoa"));
35                     pessoa.setNome(rs.getString("nome"));
36                     pessoa.setLogradouro(rs.getString("logradouro"));
37                     pessoa.setCidade(rs.getString("cidade"));
38                     pessoa.setEstado(rs.getString("estado"));
39                     pessoa.setTelefone(rs.getString("telefone"));
40                     pessoa.setEmail(rs.getString("email"));
41                     pessoa.setCnpj(rs.getString("cnpj"));
42                     return pessoa;
43                 }
44             }
45         }
46         return null;
47     }
48
49     public List<PessoaJuridica> getPessoas() throws SQLException {
50         List<PessoaJuridica> pessoas = new ArrayList<>();
51         String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
52             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaJuridica.cnpj " +
```

```
PessoaJuridicaDAO.java - Editor
PessoaJuridicaDAO.java x
Source History
49 List<PessoaJuridica> pessoas = new ArrayList<>();
50 String sql = "SELECT Pessoa.idPessoa, Pessoa.nome, Pessoa.logradouro, Pessoa.cidade, " +
51             "Pessoa.estado, Pessoa.telefone, Pessoa.email, PessoaJuridica.cnpj " +
52             "FROM Pessoa " +
53             "JOIN PessoaJuridica ON Pessoa.idPessoa = PessoaJuridica.idPessoa";
54 try (Connection conn = ConectorBD.getConnection();
55      PreparedStatement stmt = ConectorBD.getPrepared(sql);
56      ResultSet rs = ConectorBD.getSelect(stmt)) {
57     while (rs.next()) {
58         PessoaJuridica pessoa = new PessoaJuridica();
59         pessoa.setId(rs.getInt("idPessoa"));
60         pessoa.setNome(rs.getString("nome"));
61         pessoa.setLogradouro(rs.getString("logradouro"));
62         pessoa.setCidade(rs.getString("cidade"));
63         pessoa.setEstado(rs.getString("estado"));
64         pessoa.setTelefone(rs.getString("telefone"));
65         pessoa.setEmail(rs.getString("email"));
66         pessoa.setCnpj(rs.getString("cnpj"));
67         pessoas.add(pessoa);
68     }
69 }
70 return pessoas;
71 }
72
73 public void incluir(PessoaJuridica pessoa) throws SQLException {
74     String sqlInsertPessoa = "INSERT INTO Pessoa (idPessoa, nome, logradouro, cidade, " +
75                             "estado, telefone, email) VALUES (?, ?, ?, ?, ?, ?, ?)";
76     String sqlInsertPessoaJuridica = "INSERT INTO PessoaJuridica (idPessoa, cnpj) " +
77                                     "VALUES (?, ?)";
78     try (Connection conn = ConectorBD.getConnection();
79          PreparedStatement stmtInsertPessoa = conn.prepareStatement(sqlInsertPessoa);
80          PreparedStatement stmtInsertPessoaJuridica = conn.prepareStatement(sqlInsertPessoaJuridica)) {
81
82         // Inserir na tabela Pessoa
83         stmtInsertPessoa.setInt(1, pessoa.getId());
84         stmtInsertPessoa.setString(2, pessoa.getNome());
85         stmtInsertPessoa.setString(3, pessoa.getLogradouro());
86         stmtInsertPessoa.setString(4, pessoa.getCidade());
87         stmtInsertPessoa.setString(5, pessoa.getEstado());
88         stmtInsertPessoa.setString(6, pessoa.getTelefone());
89         stmtInsertPessoa.setString(7, pessoa.getEmail());
90         stmtInsertPessoa.executeUpdate();
91
92         // Inserir na tabela PessoaJuridica
93         stmtInsertPessoaJuridica.setInt(1, pessoa.getId());
94         stmtInsertPessoaJuridica.setString(2, pessoa.getCnpj());
95         stmtInsertPessoaJuridica.executeUpdate();
96     }
97 }
98
99 public void alterar(PessoaJuridica pessoa, String novoNome, String novoLogradouro, String novaCidade,
```

```
PessoaJuridicaDAO.java - Editor
PessoaJuridicaDAO.java x
Source History
97 }
98
99 public void alterar(PessoaJuridica pessoa, String novoNome, String novoLogradouro, String novaCidade,
100 String novoEstado, String novoTelefone, String novoEmail, String novoCnpj) throws SQLException {
101 String sql = "UPDATE Pessoa SET nome = ?, logradouro = ?, cidade = ?, "
102 + "estado = ?, telefone = ?, email = ? WHERE idPessoa = ?";
103 String sqlJuridica = "UPDATE PessoaJuridica SET cnpj = ? WHERE idPessoa = ?";
104
105 try (Connection conn = ConectorBD.getConnection();
106 PreparedStatement stmt = conn.prepareStatement(sql);
107 PreparedStatement stmtJuridica = conn.prepareStatement(sqlJuridica)) {
108
109 // Atualizar dados na tabela Pessoa
110 stmt.setString(1, novoNome);
111 stmt.setString(2, novoLogradouro);
112 stmt.setString(3, novaCidade);
113 stmt.setString(4, novoEstado);
114 stmt.setString(5, novoTelefone);
115 stmt.setString(6, novoEmail);
116 stmt.setInt(7, pessoa.getId());
117 stmt.executeUpdate();
118
119 // Atualizar CNPJ na tabela PessoaJuridica
120 stmtJuridica.setString(1, novoCnpj);
121 stmtJuridica.setInt(2, pessoa.getId());
122 stmtJuridica.executeUpdate();
123 }
124 }
125
126 public void excluir(int id) throws SQLException {
127 String sql = "DELETE FROM PessoaJuridica WHERE idPessoa = ?";
128 String sqlPessoa = "DELETE FROM Pessoa WHERE idPessoa = ?";
129
130 try (Connection conn = ConectorBD.getConnection();
131 PreparedStatement stmt = conn.prepareStatement(sql);
132 PreparedStatement stmtPessoa = conn.prepareStatement(sqlPessoa)) {
133 // Excluir da tabela PessoaJuridica
134 stmt.setInt(1, id);
135 stmt.executeUpdate();
136
137 // Excluir da tabela Pessoa
138 stmtPessoa.setInt(1, id);
139 stmtPessoa.executeUpdate();
140
141 System.out.println("Pessoa juridica excluida com ID: " + id);
142 }
143 }
144 }
```

4.4 CRIAR UMA CLASSE PRINCIPAL DE TESTES COM O NOME CADASTROBDTESTE, EFETUANDO AS OPERAÇÕES SEGUINTE NO MÉTODO MAIN:

Criando a Classe CadastroBDT:



```
1  /*
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4   */
5   package cadastrobd;
6
7   import java.sql.SQLException;
8   import java.util.List;
9   import cadastrobd.model.PessoaFisica;
10  import cadastrobd.model.PessoaFisicaDAO;
11  import cadastrobd.model.PessoaJuridica;
12  import cadastrobd.model.PessoaJuridicaDAO;
13  import cadastrobd.model.Util.ConectorBD;
14  import java.sql.Connection;
15
16  /**
17   *
18   * @author Administrador
19   */
20  public class CadastroBDTeste {
21
22      public static void main(String[] args) {
23          try {
24              Connection conn = ConectorBD.getConnection();
25              PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
26              PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();
27
28              // Criar uma pessoa fisica
29              PessoaFisica pf = new PessoaFisica(8, "Leonardo", "Rua k, 60", "Rio de Janeiro", "RJ",
30                  "1234-5678", "Leonardo@gmail.com", "12345678910");
31
32              // Persistir a pessoa fisica no banco de dados
33              pfDAO.incluir(pf);
34              System.out.println("Pessoa fisica criada:");
35              pf.exibir();
36              System.out.println();
37
38              // Alterar os dados da pessoa fisica no banco
39              pfDAO.alterar(pf, "Leonardo Kened", "Rua L, 50", "São Paulo", "SP",
40                  "9939-8458", "LeonardoKened@email.com", "12345828900");
41              System.out.println("-----");
42              System.out.println("Dados da pessoa fisica alterados.");
43              System.out.println("-----");
44              System.out.println();
45
46              // Consultar todas as pessoas fisicas do banco de dados e listar no console
47              List<PessoaFisica> pessoasFisicas = pfDAO.getPessoas();
48              System.out.println("Todas as pessoas fisicas:");
49              for (PessoaFisica pessoaFisica : pessoasFisicas) {
50                  pessoaFisica.exibir();
51              }
52          }
53      }
54  }
```

```
CadastroBDTeste.java - Editor
CadastroBDTeste.java x
Source History
49 for (PessoaFisica pessoaFisica : pessoasFisicas) {
50     pessoaFisica.exibir();
51 }
52 System.out.println();
53
54 // Excluir a pessoa fisica criada anteriormente no banco
55 System.out.println("-----");
56 pfDAO.excluir(pf.getId());
57 System.out.println("-----");
58 System.out.println();
59
60 // Criar uma pessoa juridica
61 PessoaJuridica pj = new PessoaJuridica(9, "Empresa Variante", "Av. São Luiz, 1900",
62     "Volta Redonda", "RJ", "1234-5678", "Variante@gmail.com", "12347578901234");
63
64 // Persistir a pessoa juridica no banco de dados
65 pjDAO.incluir(pj);
66 System.out.println("Pessoa juridica criada:");
67 pj.exibir();
68 System.out.println();
69
70 // Alterar os dados da pessoa juridica no banco
71 pjDAO.alterar(pj, "Empresa Junina", "Av. Retiro, 50", "Barra do Pirai", "RJ",
72     "9876-5432", "Junina@gmail.com", "98765432176276");
73 System.out.println("-----");
74 System.out.println("Dados da pessoa juridica alterados.");
75 System.out.println("-----");
76 System.out.println();
77
78 // Consultar todas as pessoas juridicas do banco de dados e listar no console
79 List<PessoaJuridica> pessoasJuridicas = pjDAO.getPessoas();
80 System.out.println("Todas as pessoas juridicas:");
81 for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
82     pessoaJuridica.exibir();
83 }
84 System.out.println();
85
86 // Excluir a pessoa juridica criada anteriormente no banco
87 System.out.println("-----");
88 pjDAO.excluir(pj.getId());
89 System.out.println("-----");
90 System.out.println();
91
92 ConectorBD.close(conn);
93
94 } catch (SQLException e) {
95     System.out.println("Ocorreu um erro: " + e.getMessage());
96 }
97 }
98 }
99 }
```

- *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.*

A saída do sistema deverá ser semelhante à que é apresentada a seguir:

```
Output - CadastroBD (run) x
run:
Pessoa fisica criada:
-----
ID: 8
Nome: Leonardo
Logradouro: Rua k, 60
Cidade: Rio de Janeiro
Estado: RJ
Telefone: 1234-5678
Email: Leonardo@gmail.com
CPF: 12345678910

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

Todas as pessoas fisicas:
-----
ID: 1
Nome: Natalia
Logradouro: Rua I, 50
Cidade: Angra dos Reis
Estado: RJ
Telefone: 1111-1111
Email: Natalia@gmail.com
CPF: 11111111111

-----
ID: 2
Nome: Bruno
Logradouro: Rua P, 10
Cidade: Volta Redonda
Estado: RJ
Telefone: 2222-2222
Email: Bruno@gmail.com
CPF: 22222222222

-----
ID: 3
Nome: Jonas
Logradouro: Rua S, 80
Cidade: Vitoria
Estado: ES
Telefone: 3333-3333
Email: Jonas@gmail.com
CPF: 33333333333

-----
ID: 8
Nome: Leonardo Kened
Logradouro: Rua L, 50
Cidade: São Paulo
Estado: SP
Telefone: 9939-8458
Email: LeonardoKened@email.com
CPF: 12345828900
```

Pessoa fisica excluida com ID: 8

```
Pessoa juridica criada:
-----
ID: 9
Nome: Empresa Variante
Logradouro: Av. São Luiz, 1900
Cidade: Volta Redonda
Estado: RJ
Telefone: 1234-5678
Email: Variante@gmail.com
CPF: 12347578901234

-----
Dados da pessoa juridica alterados.
-----
|
Todas as pessoas juridicas:
-----
ID: 4
Nome: Distribuidora Renova
Logradouro: Avenida J, 80
Cidade: São Paulo
Estado: SP
Telefone: 4444-4444
Email: RenovaDist@gmail.com
CPF: 4444444444444444

-----
ID: 5
Nome: Empresa Jasper
Logradouro: Avenida H, 70
Cidade: São Paulo
Estado: SP
Telefone: 5555-5555
Email: Jasper@gmail.com
CPF: 5555555555555555

-----
ID: 9
Nome: Empresa Junina
Logradouro: Av. Retiro, 50
Cidade: Barra do Pirai
Estado: RJ
Telefone: 9876-5432
Email: Junina@gmail.com
CPF: 98765432176276

-----
Pessoa juridica excluida com ID: 9
-----

BUILD SUCCESSFUL (total time: 0 seconds)
```


5 ANÁLISE E CONCLUSÃO:

5.1 QUAL A IMPORTÂNCIA DOS COMPONENTES DE MIDDLEWARE, COMO O JDBC?

Componentes de middleware, como o JDBC, são essenciais para a criação de sistemas de software, pois oferecem:

Abstração: Simplificam a comunicação com bancos de dados, ocultando detalhes específicos.

Portabilidade: Permitem que a mesma aplicação funcione com diferentes SGBDs sem grandes mudanças no código.

Segurança: Facilitam a implementação de conexões seguras.

Gerenciamento de Conexões: Otimizam o desempenho através de pools de conexões.

Interoperabilidade: Facilitam a comunicação entre sistemas diferentes.

Facilidade de Desenvolvimento: Simplificam operações de banco de dados, permitindo foco na lógica de negócios.

Padronização: Oferecem uma API consistente e confiável para acesso a dados em Java.

Em suma, middleware como o JDBC facilita a construção de sistemas escaláveis e manuteníveis, ao abstrair a complexidade e melhorar a interoperabilidade.

5.2 QUAL A DIFERENÇA NO USO DE STATEMENT OU PREPAREDSTATEMENT PARA A MANIPULAÇÃO DE DADOS?

A diferença principal entre o uso de Statement e PreparedStatement no JDBC está relacionada à segurança, desempenho e flexibilidade na manipulação de dados em um banco de dados.

• **Statement**

Uso: O Statement é utilizado para executar comandos SQL estáticos, ou seja, quando a consulta SQL não muda durante a execução.

Segurança: Menos seguro, pois é vulnerável a ataques de SQL Injection. Como a consulta é construída como uma string, se os dados inseridos pelo usuário não forem devidamente validados, um atacante pode injetar código SQL malicioso.

Desempenho: Menos eficiente para consultas repetitivas, já que o SQL é compilado e executado a cada vez que é enviado ao banco de dados.

- ***PreparedStatement***

Uso: O PreparedStatement é utilizado para executar comandos SQL parametrizados, o que significa que a consulta SQL pode ter parâmetros que são definidos em tempo de execução.

Segurança: Mais seguro contra SQL Injection, pois os parâmetros são passados separadamente e tratados pelo driver JDBC, evitando que qualquer código malicioso seja executado.

Desempenho: Mais eficiente para consultas repetitivas, pois a consulta SQL é pré-compilada uma vez e pode ser executada várias vezes com diferentes parâmetros, resultando em menos sobrecarga de processamento.

Flexibilidade: Suporta facilmente a reutilização de consultas com diferentes valores, tornando o código mais modular e limpo.

5.3 COMO O PADRÃO DAO MELHORA A MANUTENIBILIDADE DO SOFTWARE?

O padrão DAO (Data Access Object) melhora a manutenibilidade do software de várias maneiras:

Separação de Responsabilidades: O DAO isola a lógica de acesso a dados do restante da aplicação, permitindo que a lógica de negócios e a lógica de persistência sejam desenvolvidas e mantidas separadamente.

Facilidade de Alteração: Mudanças no banco de dados ou na tecnologia de persistência (como trocar de MySQL para PostgreSQL) podem ser feitas no DAO sem impactar o restante da aplicação.

Reutilização de Código: O DAO centraliza as operações de acesso a dados, evitando duplicação de código e facilitando a reutilização.

Testabilidade: O uso do DAO facilita a criação de testes unitários, pois permite a simulação (mocking) do acesso a dados sem interferir na lógica de negócios.

Escalabilidade: A abstração oferecida pelo DAO facilita a introdução de novas funcionalidades ou a adaptação a novas necessidades, mantendo o código organizado e modular.

5.4 COMO A HERANÇA É REFLETIDA NO BANCO DE DADOS, QUANDO LIDAMOS COM UM MODELO ESTRITAMENTE RELACIONAL?

Quando lidamos com herança em um modelo estritamente relacional, existem três abordagens principais para refletir a herança no banco de dados:

- ***Tabela Única por Hierarquia (Single Table Inheritance)***

Descrição: Toda a hierarquia de classes é mapeada para uma única tabela no banco de dados. A tabela contém todas as colunas necessárias para armazenar os atributos de todas as subclasses, com uma coluna adicional para identificar o tipo da subclasse (discriminador).

Vantagens: Simplicidade e melhor desempenho para consultas, já que não há necessidade de joins.

Desvantagens: Pode resultar em muitas colunas nulas (sparsidade) se as subclasses tiverem atributos muito diferentes, e a tabela pode se tornar muito grande e difícil de gerenciar.

- ***Tabela por Subclasse (Class Table Inheritance)***

Descrição: Cada classe na hierarquia de herança tem sua própria tabela no banco de dados. A tabela da superclasse armazena os atributos comuns, enquanto as tabelas das subclasses armazenam atributos específicos e uma chave estrangeira que referencia a superclasse.

Vantagens: Sem redundância de dados, e melhor normalização. A tabela de cada subclasse contém apenas os atributos relevantes.

Desvantagens: Consultas requerem joins, o que pode afetar o desempenho.

- ***Tabela por Concreta (Concrete Table Inheritance)***

Descrição: Cada subclasse tem sua própria tabela, que contém todos os atributos da subclasse, incluindo aqueles herdados da superclasse.

Vantagens: Simplicidade nas consultas, pois não há necessidade de joins. Cada tabela representa uma entidade completa.

Desvantagens: Redundância de dados, já que os atributos comuns às subclasses são repetidos em várias tabelas.

- ***Resumo***

Tabela Única por Hierarquia: Simples, mas pode gerar sparsidade.

Tabela por Subclasse: Normalizada, mas requer joins.

Tabela por Concreta: Sem joins, mas com redundância de dados.

6 2º PROCEDIMENTO | ALIMENTANDO A BASE

6.1 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.

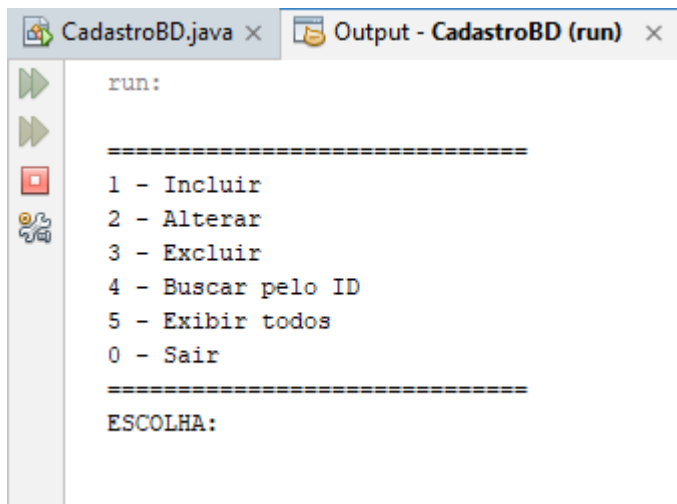
No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

7 ALTERAR O MÉTODO MAIN DA CLASSE PRINCIPAL DO PROJETO, PARA IMPLEMENTAÇÃO DO CADASTRO EM MODO TEXTO:

7.1 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.



```
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      }
48
49      private static void printMenu() {
50          System.out.println("\n=====");
51          System.out.println("1 - Incluir");
52          System.out.println("2 - Alterar");
53          System.out.println("3 - Excluir");
54          System.out.println("4 - Buscar pelo ID");
55          System.out.println("5 - Exibir todos");
56          System.out.println("0 - Sair");
57          System.out.println("=====");
58      }
59  }
```

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

7.2 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.

```
}

private static String input(String prompt) {
    System.out.print(prompt);
    return sc.nextLine();
}

private static int inputInt(String prompt) {
    System.out.print(prompt);
    try {
        return Integer.parseInt(sc.nextLine());
    } catch (NumberFormatException e) {
        System.out.println("Erro: Entrada invalida. Tente novamente.");
        return inputInt(prompt);
    }
}

private static void incluir() {
    System.out.println("\nIncluindo pessoa...");
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
    Integer id = inputInt("Informe o ID: ");
    switch (tipoPessoa) {
        case "F" -> {
            try {
                pfDao.incluir(criarPessoaFisica(id));
                System.out.println("Pessoa fisica incluida com sucesso!");
            } catch (SQLException e) {
                System.out.println("Erro ao incluir pessoa fisica: " + e.getMessage());
            }
        }
        case "J" -> {
            try {
                pjDao.incluir(criarPessoaJuridica(id));
                System.out.println("Pessoa juridica incluida com sucesso!");
            } catch (SQLException e) {
                System.out.println("Erro ao incluir pessoa juridica: " + e.getMessage());
            }
        }
        default ->
            System.out.println("Tipo de pessoa invalido!");
    }
}
```

```

private static PessoaFisica criarPessoaFisica(Integer id) {
    System.out.println("Criando Pessoa Fisica...");
    String nome = input("Informe o nome: ");
    String logradouro = input("Informe o logradouro: ");
    String cidade = input("Informe a cidade: ");
    String estado = input("Informe o estado: ");
    String telefone = input("Informe o telefone: ");
    String email = input("Informe o email: ");
    String cpf = input("Informe o CPF: ");
    return new PessoaFisica(id, nome, logradouro, cidade, estado, telefone, email, cpf);
}

private static PessoaJuridica criarPessoaJuridica(Integer id) {
    System.out.println("Criando Pessoa Juridica...");
    String nome = input("Informe o nome: ");
    String logradouro = input("Informe o logradouro: ");
    String cidade = input("Informe a cidade: ");
    String estado = input("Informe o estado: ");
    String telefone = input("Informe o telefone: ");
    String email = input("Informe o email: ");
    String cnpj = input("Informe o CNPJ: ");
    return new PessoaJuridica(id, nome, logradouro, cidade, estado, telefone, email, cnpj);
}

```

=====	=====
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: 15	Informe o ID: 70
Criando Pessoa Fisica...	Criando Pessoa Juridica...
Informe o nome: joao	Informe o nome: ministerio da educaçao
Informe o logradouro: rua M	Informe o logradouro: av dos trabalhadores
Informe a cidade: Rio de Janeiro	Informe a cidade: volta redonda
Informe o estado: RJ	Informe o estado: rj
Informe o telefone: 548912556	Informe o telefone: 485684668
Informe o email: joao@gmail.com	Informe o email: mineducaçao@gmail.com
Informe o CPF: 45815925865	Informe o CNPJ: 1254439800019
Pessoa fisica incluída com sucesso!	Pessoa juridica incluída com sucesso!

7.3 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.

```

private static void alterar() {
    System.out.println("\nAlterando pessoa...");
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
    if (tipoPessoa.equals("F")) {
        try {
            Integer id = inputInt("Informe o ID da Pessoa Fisica: ");
            PessoaFisica pf = pfDao.getPessoa(id);
            if (pf != null) {
                System.out.println("Dados atuais da Pessoa Fisica:");
                pf.exibir();

                String novoNome = input("Informe o novo nome: ");
                String novoLogradouro = input("Informe o novo logradouro: ");
                String novaCidade = input("Informe a nova cidade: ");
                String novoEstado = input("Informe o novo estado: ");
                String novoTelefone = input("Informe o novo telefone: ");
                String novoEmail = input("Informe o novo email: ");
                String novoCpf = input("Informe o novo CPF: ");

                pfDao.alterar(pf, novoNome, novoLogradouro, novaCidade,
                    novoEstado, novoTelefone, novoEmail, novoCpf);
                System.out.println("Pessoa fisica alterada com sucesso!");
            } else {
                System.out.println("ID errado!");
            }
        } catch (NullPointerException | SQLException e) {
            System.out.println("Erro ao alterar pessoa fisica: " + e.getMessage());
        }
    } else if (tipoPessoa.equals("J")) {
        try {
            Integer id = inputInt("Informe o ID da Pessoa Juridica: ");
            PessoaJuridica pj = pjDao.getPessoa(id);
            if (pj != null) {
                System.out.println("Dados atuais da Pessoa Juridica:");
                pj.exibir();
            }
        }
    }
}

```

```

        pj.exibir();

        String novoNome = input("Informe o novo nome: ");
        String novoLogradouro = input("Informe o novo logradouro: ");
        String novaCidade = input("Informe a nova cidade: ");
        String novoEstado = input("Informe o novo estado: ");
        String novoTelefone = input("Informe o novo telefone: ");
        String novoEmail = input("Informe o novo email: ");
        String novoCnpj = input("Informe o novo CNPJ: ");

        pjDao.alterar(pj, novoNome, novoLogradouro, novaCidade,
            novoEstado, novoTelefone, novoEmail, novoCnpj);
        System.out.println("Pessoa juridica alterada com sucesso!");
    } else {
        System.out.println("ID errado!");
    }
} catch (NullPointerException | SQLException e) {
    System.out.println("Erro ao alterar pessoa juridica: " + e.getMessage());
}
} else {
    System.out.println("Tipo de pessoa invalido!");
}
}
}

```

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

ESCOLHA: 2

Alterando pessoa...

F - Pessoa Fisica | J - Pessoa Juridica

TIPO DE PESSOA: j

Informe o ID da Pessoa Juridica: 70

Dados atuais da Pessoa Juridica:

ID: 70

Nome: ministerio da educaçao

Logradouro: av dos trabalhadores

Cidade: volta redonda

Estado: rj

Telefone: 485684668

Email: mineducaçao@gmail.com

CPF: 1254439800019

Informe o novo nome: ministerio do juiz

Informe o novo logradouro: av dos ministros

Informe a nova cidade: volta redonda

Informe o novo estado: ej

Informe o novo telefone: 485894565

Informe o novo email: minjuiz@gmail.com

Informe o novo CNPJ: 1584569810002

Pessoa juridica alterada com sucesso!

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

ESCOLHA: 2

Alterando pessoa...

F - Pessoa Fisica | J - Pessoa Juridica

TIPO DE PESSOA: f

Informe o ID da Pessoa Fisica: 15+

Erro: Entrada invalida. Tente novamente.

Informe o ID da Pessoa Fisica: 15

Dados atuais da Pessoa Fisica:

ID: 15

Nome: joao

Logradouro: rua M

Cidade: Rio de Janeiro

Estado: RJ

Telefone: 548912556

Email: joao@gmail.com

CPF: 45815925865

Informe o novo nome: maria

Informe o novo logradouro: rua n

Informe a nova cidade: rio de janeiro

Informe o novo estado: rj

Informe o novo telefone: 8544255

Informe o novo email: maria@gmail.com

Informe o novo CPF: 18456958123

Pessoa fisica alterada com sucesso!

7.4 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.


```

private static void excluir() {
    System.out.println("\nExcluindo pessoa...");
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
    switch (tipoPessoa) {
        case "F" -> {
            try {
                Integer id = inputInt("Informe o ID da Pessoa Fisica: ");
                PessoaFisica pf = pfDao.getPessoa(id);
                if (pf != null) {
                    pfDao.excluir(pf.getId());
                    System.out.println("Sucesso ao excluir!");
                } else {
                    System.out.println("ID errado!");
                }
            } catch (NullPointerException | SQLException e) {
                System.out.println("Erro ao excluir pessoa fisica: " + e.getMessage());
            }
        }
        case "J" -> {
            try {
                Integer id = inputInt("Informe o ID da Pessoa Juridica: ");
                PessoaJuridica pj = pjDao.getPessoa(id);
                if (pj != null) {
                    pjDao.excluir(pj.getId());
                    System.out.println("Sucesso ao excluir!");
                } else {
                    System.out.println("ID errado!");
                }
            } catch (NullPointerException | SQLException e) {
                System.out.println("Erro ao excluir pessoa juridica: " + e.getMessage());
            }
        }
        default ->
            System.out.println("Tipo de pessoa invalido!");
    }
}

```

```
=====
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: j
Informe o ID da Pessoa Juridica: 70
Pessoa juridica excluida com ID: 70
Sucesso ao excluir!

```
=====
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: 15
Pessoa fisica excluida com ID: 15
Sucesso ao excluir!

7.5 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.

```

private static void buscarPeloId() {
    System.out.println("\nBuscando pessoa pelo ID...");
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
    switch (tipoPessoa) {
        case "F" -> {
            try {
                Integer id = inputInt("Informe o ID da Pessoa Fisica: ");
                PessoaFisica pf = pfDao.getPessoa(id);
                if (pf != null) {
                    pf.exibir();
                } else {
                    System.err.println("Pessoa fisica com o ID " + id + " nao encontrada!");
                }
            } catch (SQLException e) {
                System.err.println("Erro ao buscar pessoa fisica: " + e.getMessage());
            }
        }
        case "J" -> {
            try {
                Integer id = inputInt("Informe o ID da Pessoa Juridica: ");
                PessoaJuridica pj = pjDao.getPessoa(id);
                if (pj != null) {
                    pj.exibir();
                } else {
                    System.err.println("Pessoa juridica com o ID " + id + " nao encontrada!");
                }
            } catch (SQLException e) {
                System.err.println("Erro ao buscar pessoa juridica: " + e.getMessage());
            }
        }
        default ->
            System.out.println("Tipo de pessoa invalido!");
    }
}

```

```
=====
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: 5

ID: 5

Nome: Empresa Jasper

Logradouro: Avenida H, 70

Cidade: São Paulo

Estado: SP

Telefone: 5555-5555

Email: Jasper@gmail.com

CPF: 55555555555555

```
=====
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: 70

Pessoa juridica com o ID 70 nao encontrada!

```
=====
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: f

Informe o ID da Pessoa Fisica: 1

ID: 1

Nome: Natalia

Logradouro: Rua I, 50

Cidade: Angra dos Reis

Estado: RJ

Telefone: 1111-1111

Email: Natalia@gmail.com

CPF: 111111111111

7.6 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.

```

private static void exibirTodos() {
    System.out.println("\nExibindo todas as pessoas...");
    System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
    String tipoPessoa = input("TIPO DE PESSOA: ").toUpperCase();
    try {
        switch (tipoPessoa) {
            case "F" -> {
                ArrayList<PessoaFisica> listaPf = (ArrayList<PessoaFisica>) pfDao.getPessoas();
                for (PessoaFisica pessoa : listaPf) {
                    pessoa.exibir();
                }
            }
            case "J" -> {
                ArrayList<PessoaJuridica> listaPj = (ArrayList<PessoaJuridica>) pjDao.getPessoas();
                for (PessoaJuridica pessoa : listaPj) {
                    pessoa.exibir();
                }
            }
            default ->
                System.out.println("Tipo de pessoa invalido!");
        }
    } catch (SQLException e) {
        System.out.println("Erro ao exibir pessoas: " + e.getMessage());
    }
}

```

=====

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: j

ID: 4

Nome: Distribuidora Renova

Logradouro: Avenida J, 80

Cidade: São Paulo

Estado: SP

Telefone: 4444-4444

Email: RenovaDist@gmail.com

CPF: 4444444444444444

ID: 5

Nome: Empresa Jasper

Logradouro: Avenida H, 70

Cidade: São Paulo

Estado: SP

Telefone: 5555-5555

Email: Jasper@gmail.com

CPF: 5555555555555555

=====

=====

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: Natalia

Logradouro: Rua I, 50

Cidade: Angra dos Reis

Estado: RJ

Telefone: 1111-1111

Email: Natalia@gmail.com

CPF: 111111111111

ID: 2

Nome: Bruno

Logradouro: Rua P, 10

Cidade: Volta Redonda

Estado: RJ

Telefone: 2222-2222

Email: Bruno@gmail.com

CPF: 222222222222

ID: 3

Nome: Jonas

Logradouro: Rua S, 80

Cidade: Vitoria

Estado: ES

Telefone: 3333-3333

Email: Jonas@gmail.com

CPF: 333333333333

=====

7.7 SELECIONADA A OPÇÃO SAIR, FINALIZAR A EXECUÇÃO DO SISTEMA.

```
=====
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
=====
ESCOLHA: 0
Finalizando...
BUILD SUCCESSFUL (total time: 9 minutes 30 seconds)
```

7.8 FEITAS AS OPERAÇÕES, VERIFICAR OS DADOS NO SQL SERVER, COM A UTILIZAÇÃO DA ABA SERVICES, DIVISÃO DATABASES, DO NETBEANS, OU ATRAVÉS DO SQL SERVER MANAGEMENT STUDIO.

SELECT TOP 100 * FROM dbo... ×							
		Max. rows:	100	Fetches Rows: 7	Matching Rows:		
#	idPessoa	nome	logradouro	cidade	estado	telefone	email
1	1	Natalia	Rua I, 50	Angra dos Reis	RJ	1111-1111	Natalia@gmail.com
2	2	Bruno	Rua P, 10	Volta Redonda	RJ	2222-2222	Bruno@gmail.com
3	3	Jonas	Rua 5, 80	Vitoria	ES	3333-3333	Jonas@gmail.com
4	4	Distribuidora Renova	Avenida J, 80	São Paulo	SP	4444-4444	RenovaDist@gmail.com
5	5	Empresa Jasper	Avenida H, 70	São Paulo	SP	5555-5555	Jasper@gmail.com
6	15	lion	rua L	Vitoria	ES	14582236	Lion@gmail.com
7	60	ministerio da educação	rua dos trabalhadores	volta redonda	rj	48569598	MinEducação@gmail.co

8 ANÁLISE E CONCLUSÃO:

8.1 QUAIS AS DIFERENÇAS ENTRE A PERSISTÊNCIA EM ARQUIVO E A PERSISTÊNCIA EM BANCO DE DADOS ?

A persistência de dados refere-se à prática de armazenar informações de forma que elas permaneçam disponíveis para uso futuro, mesmo após o término de um programa ou a reinicialização de um sistema. Existem diferentes formas de implementar a persistência, sendo duas delas o armazenamento em arquivos e em bancos de dados. Abaixo estão as principais diferenças entre essas abordagens:

- **Estrutura dos Dados**

Arquivo: Menos estruturado, adequado para dados simples.

Banco de Dados: Altamente estruturado, ideal para dados complexos.

- **Consultas:**

Arquivo: Pesquisa menos eficiente, manual.

Banco de Dados: Rápido, com suporte a SQL.

- **Concorrência e Segurança:**

Arquivo: Concorrência difícil, segurança básica.

Banco de Dados: Suporte avançado a múltiplos usuários e segurança.

- **Escalabilidade:**

Arquivo: Limitada, adequado para pequenos volumes de dados.

Banco de Dados: Alta escalabilidade para grandes volumes de dados.

- **Complexidade:**

Arquivo: Simples de implementar, mas limitado.

Banco de Dados: Mais complexo, mas poderoso e eficiente.

8.2 COMO O USO DE OPERADOR LAMBDA SIMPLIFICOU A IMPRESSÃO DOS VALORES CONTIDOS NAS ENTIDADES, NAS VERSÕES MAIS RECENTES DO JAVA?

Nas versões mais recentes do Java, o uso de expressões lambda simplificou significativamente a forma como lidamos com coleções de dados, especialmente quando se trata de imprimir os valores contidos em entidades. Aqui está como isso acontece:

• *Redução de Código Boilerplate:*

Antes das expressões lambda, para iterar sobre uma coleção e imprimir seus valores, era necessário escrever um bloco de código relativamente longo, geralmente utilizando loops for ou classes anônimas.

```
java Copiar código  
  
List<String> lista = Arrays.asList("Java", "Python", "C++");  
for (String item : lista) {  
    System.out.println(item);  
}
```

Com expressões lambda, o mesmo resultado pode ser obtido de forma muito mais concisa:

```
java Copiar código  
  
List<String> lista = Arrays.asList("Java", "Python", "C++");  
lista.forEach(item -> System.out.println(item));
```

• **Uso de Method References:**

A lambda pode ser simplificada ainda mais utilizando referências a métodos (method references), o que elimina a necessidade de especificar explicitamente os parâmetros e a lógica de impressão:

```
java Copiar código  
  
lista.forEach(System.out::println);
```

Isso torna o código mais legível e direto ao ponto.

• **Integração com APIs de Stream:**

As expressões lambda são frequentemente usadas em conjunto com a API de Streams, introduzida no Java 8, permitindo que operações como filtro, mapeamento e iteração sejam feitas de maneira fluente e expressiva:

```
java Copiar código  
  
lista.stream()  
    .filter(s -> s.startsWith("J"))  
    .forEach(System.out::println);
```

Aqui, você pode filtrar, mapear e realizar outras operações de forma muito mais simples e clara.

• **Conclusão:**

O uso de expressões lambda em Java moderniza o código, reduz a quantidade de código boilerplate e melhora a legibilidade, especialmente em tarefas comuns como a impressão de valores em coleções. Isso simplifica consideravelmente o desenvolvimento, tornando o código mais fácil de escrever e manter.

8.3 POR QUE MÉTODOS ACIONADOS DIRETAMENTE PELO MÉTODO MAIN, SEM O USO DE UM OBJETO, PRECISAM SER MARCADOS COMO STATIC?

Métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static por causa da forma como a linguagem Java gerencia a execução dos programas. Aqui estão os motivos principais:

- ***Método main é Estático:***

O método main é o ponto de entrada de qualquer aplicação Java e é definido como static. Isso significa que ele pertence à classe, não a uma instância de objeto. Quando a JVM (Java Virtual Machine) inicia a execução do programa, ela chama o método main sem criar uma instância da classe que o contém.

Como o main é estático, ele só pode acessar outros métodos e variáveis estáticas diretamente. Métodos não estáticos (ou seja, métodos de instância) requerem que um objeto seja criado para serem invocados.

- ***Pertencem à Classe, Não a Objetos:***

Métodos marcados como static pertencem à classe em si e não a uma instância específica dessa classe. Isso significa que você pode chamar um método static sem precisar criar um objeto da classe.

Quando você chama um método static diretamente no main, como `MinhaClasse.meuMetodo()`, não há necessidade de criar uma instância da `MinhaClasse`. Isso simplifica a chamada e o gerenciamento de métodos que não dependem de estados específicos de um objeto.

- **Restrições de Acesso:**

Métodos static não podem acessar diretamente membros de instância (não estáticos) da classe porque esses membros de instância estão associados a um objeto específico. Para acessar membros de instância dentro de um método estático, você precisaria primeiro criar uma instância do objeto.

Portanto, quando você quer que um método seja acessível diretamente pelo main, sem a necessidade de criar uma instância, ele deve ser static para que a JVM possa chamá-lo diretamente.

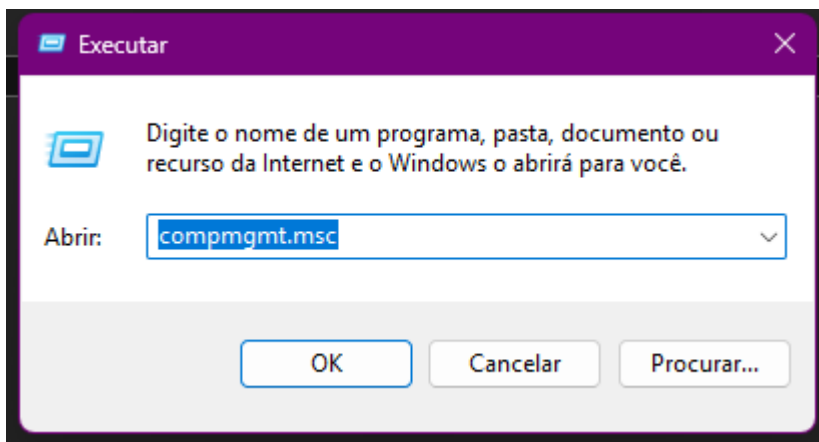
- **Resumo:**

Métodos acionados diretamente pelo método main precisam ser marcados como static porque o main é estático e, por isso, só pode acessar outros membros estáticos. Isso permite que esses métodos sejam chamados sem criar uma instância de sua classe, simplificando a execução do programa.

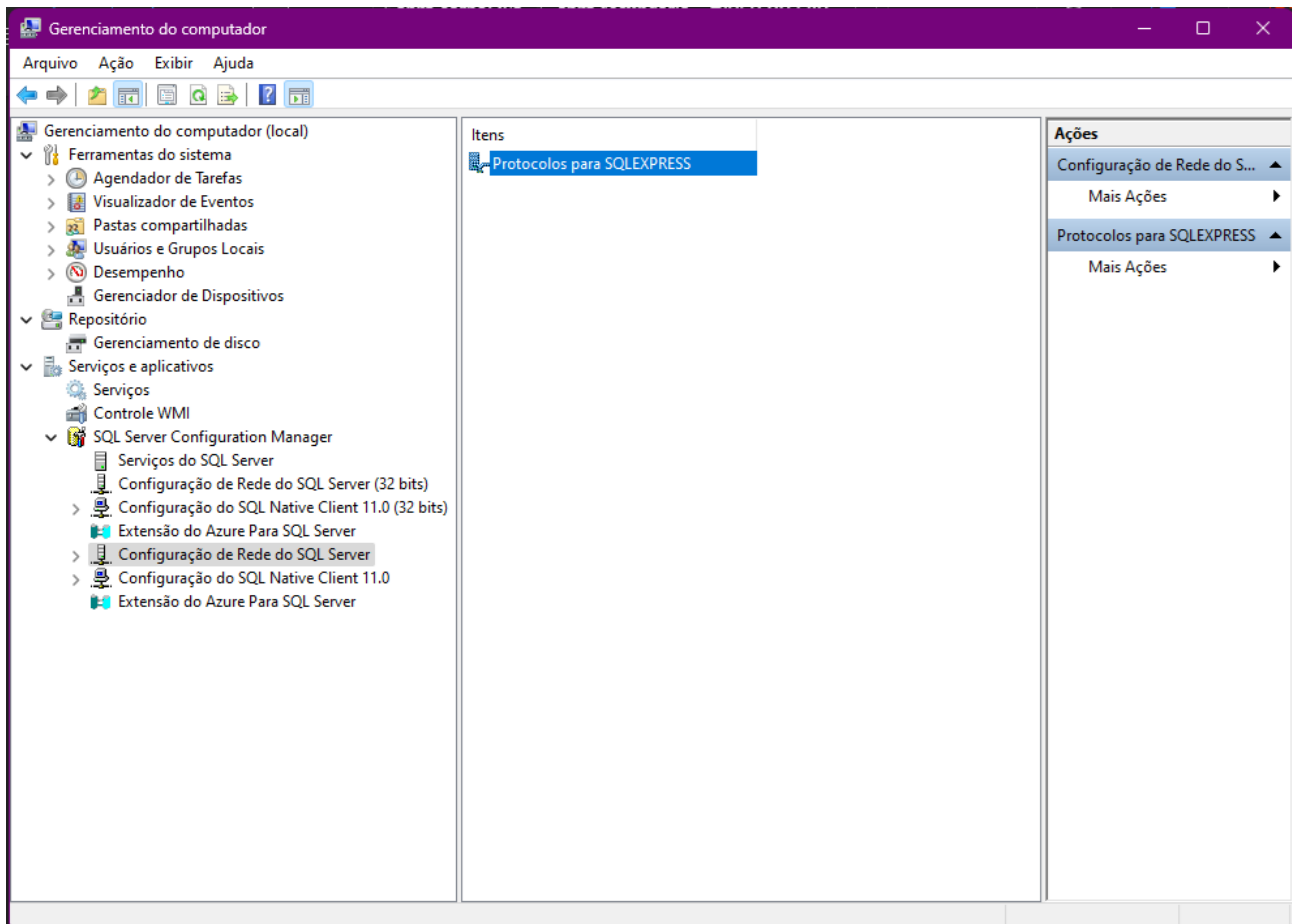
9 SEGUIE ALGUNS PASSOS QUE TIVE QUE FAZER PARA LIBERAR A PORTA 1433 PARA RODAR O SERVIDOR

Primeiro liberar a porta 1433 caso ela não esteja ativa siga os passos abaixo:

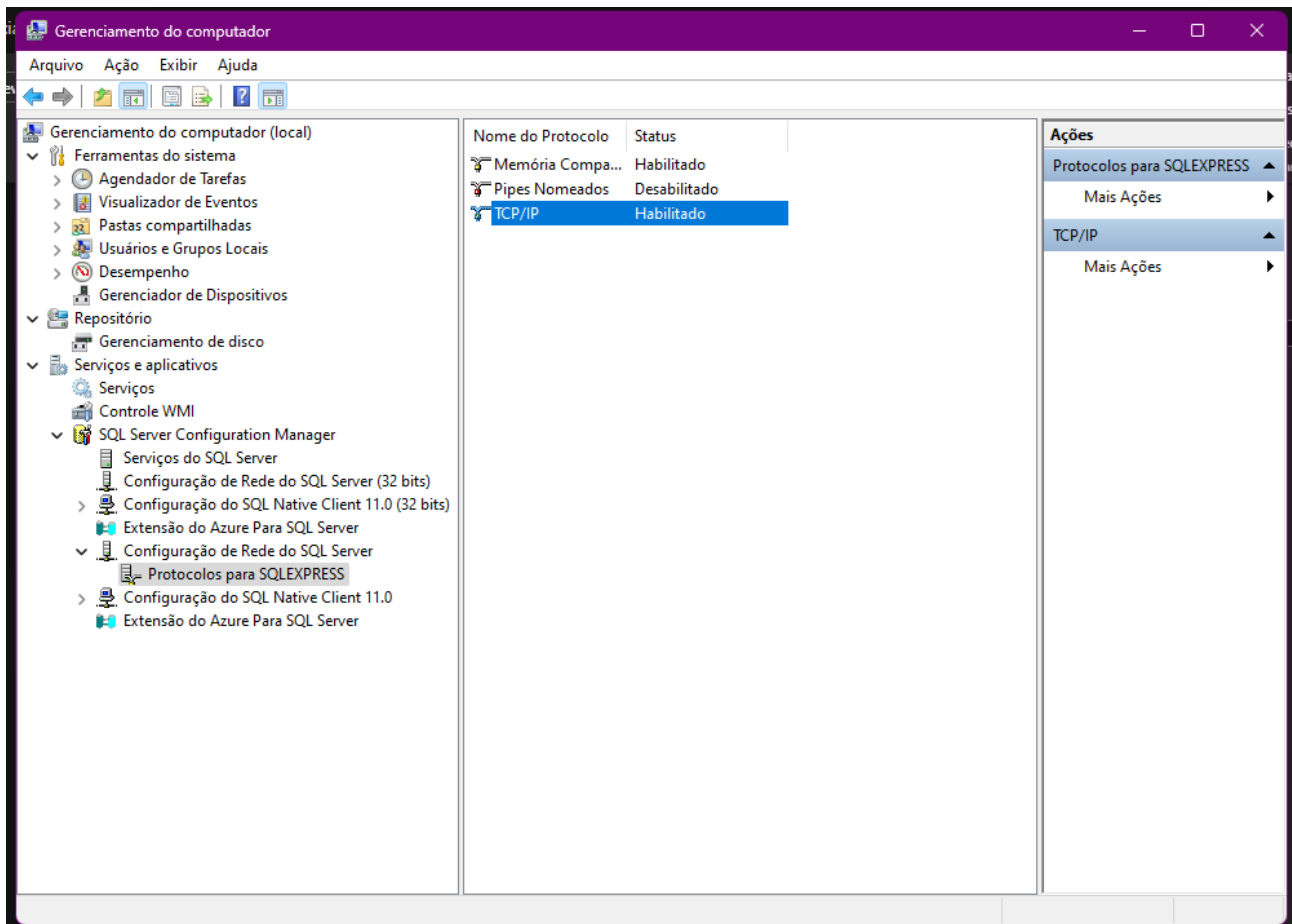
Abra o executar (win + R) e digite o seguinte código “compmgmt.msc”



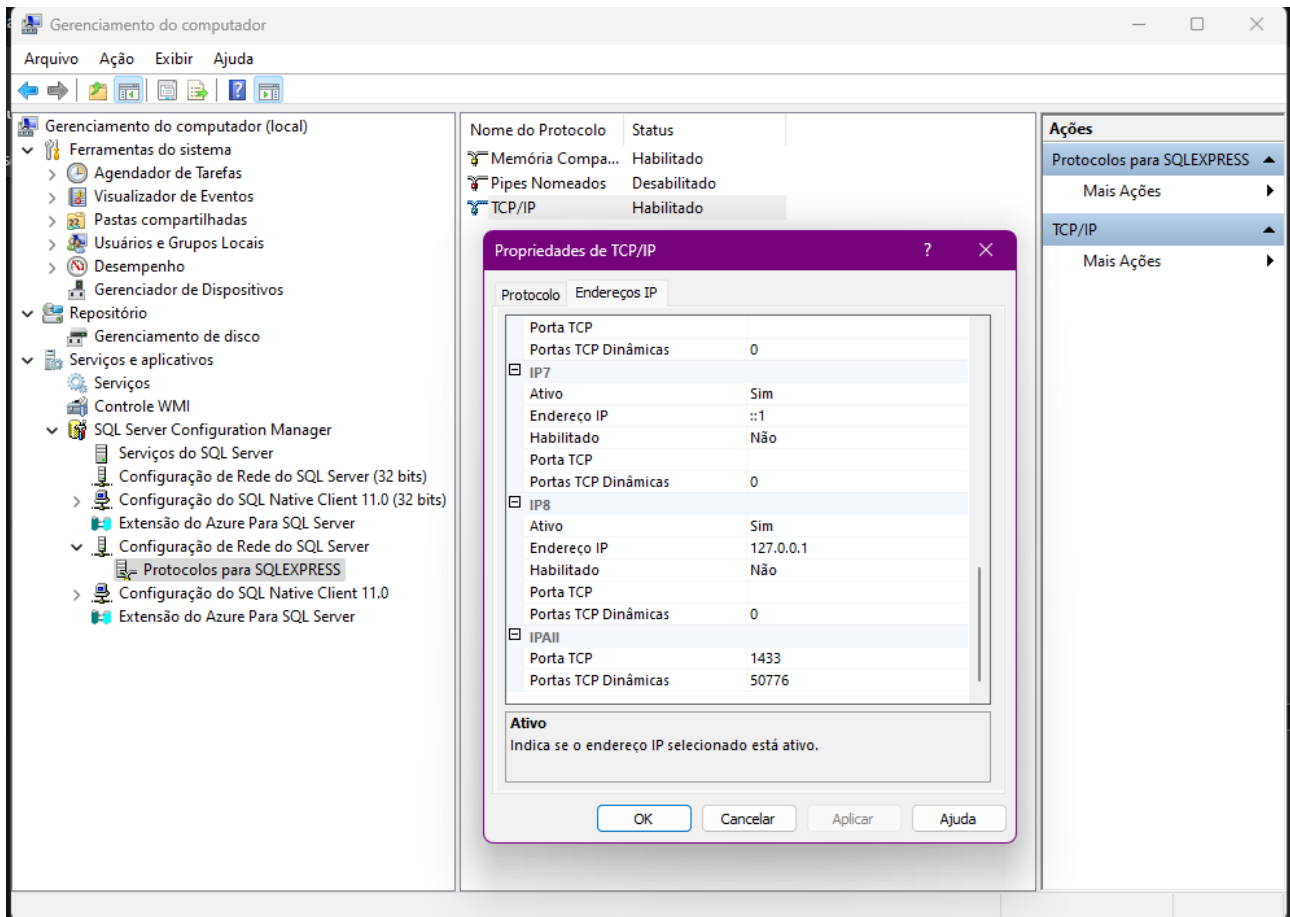
Depois selecione as configurações de rede do SQL, Clicando nela vai aparecer os serves que estão na sua maquina



Selecione o seu servidor e verifique o ip que consta nele caso o TCP/IP estiver desabilitado favor habilitar



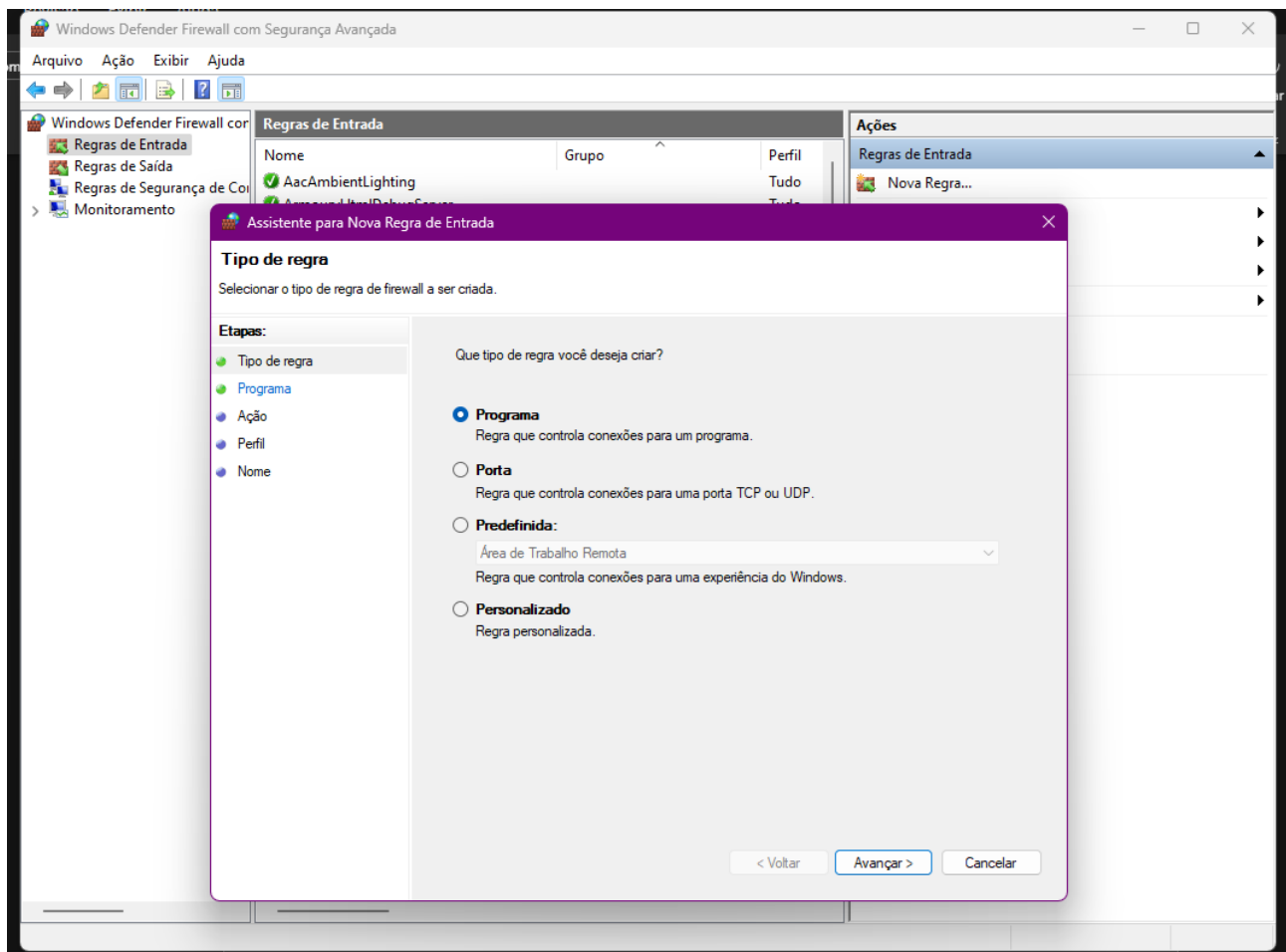
Indo em propriedades do tcp/ip na parte avançada la em abaixo vai estar sua porta altere para a 1433 depois disso vai precisar reiniciar o servidor em serviços



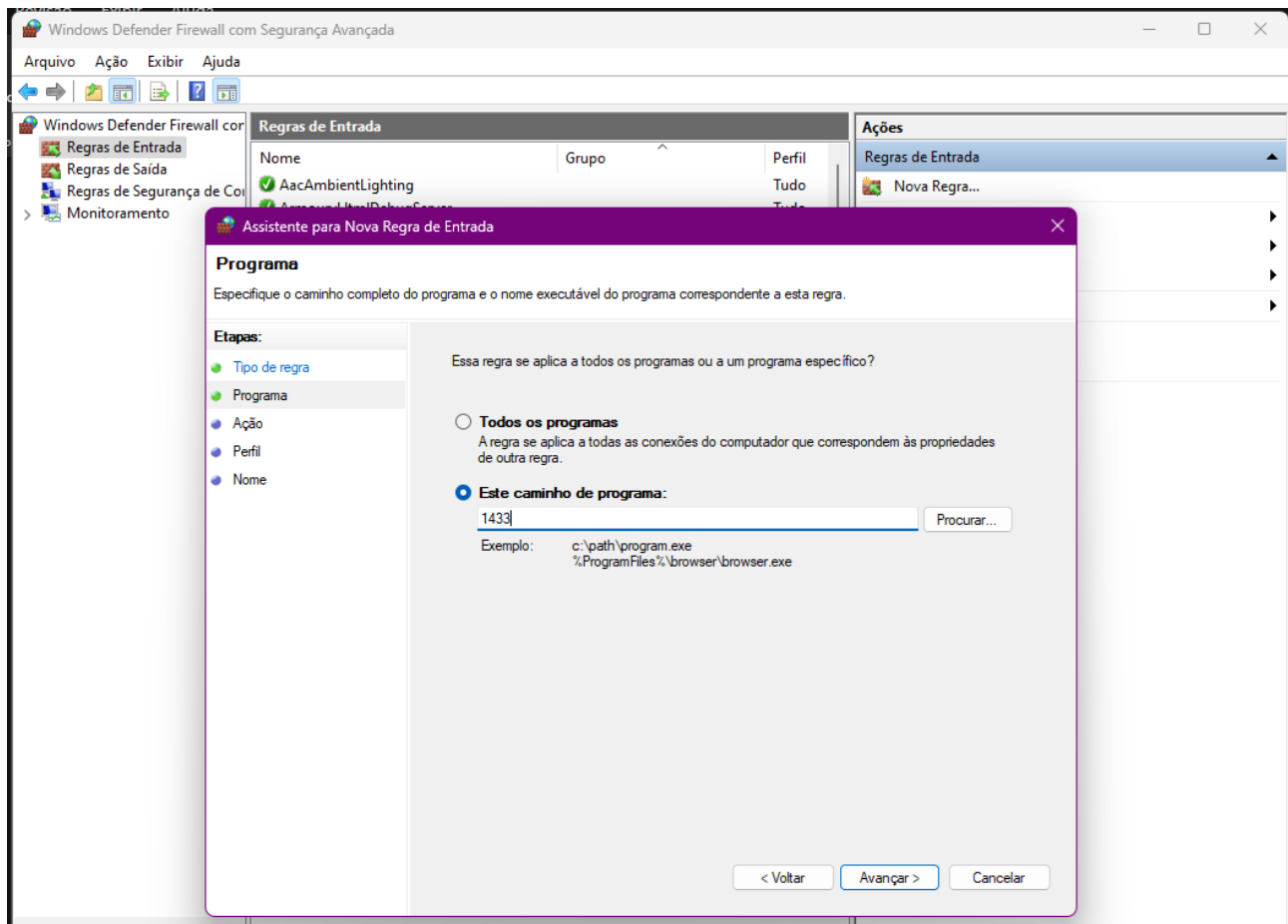
Depois disso a porta do sql serve vai estar configurada para a porta 1433.

Agora precisamos criar a porta no firewall da seguinte forma abaixo:

Click em Nova Regra e depois de um Click em Avançar



Aqui você vai digitar a sua porta no caso a 1433 que queremos liberar, após click em avançar



Clicando em avançar até chegar nessa tela, aqui você pode colocar o nome que quiser e aperta em concluir para finalizar o processo.

Depois disso sua porta vai estar liberada para ser usada esse método pode ser usado com outras portas que você queira usar.

Atenção para não utilizar portas que já estejam sendo usadas caso isso aconteça você terá um problema.

