



## Relatório

UNIVERSADE ESTÁCIO DE SÁ POLO Paraíba  
CURSO: DESENVOLVIMENTO FULL STACK  
DISCIPLINA: INICIADO O CAMINHO PELO JAVA

Aluno: Yvo Murilo

Campus: Polo João Pessoa - Paraíba – PB

TURMA: 22.3

3º - SEMESTRE

### Objetivo da Prática

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

## Todos os códigos



```
1 package cadastrobd;
2
3 import java.sql.SQLException;
4 import java.util.ArrayList;
5 import java.util.Scanner;
6
7 import cadastrobd.model.PessoaFisica;
8 import cadastrobd.model.PessoaFisicaDAO;
9 import cadastrobd.model.PessoaJuridica;
10 import cadastrobd.model.PessoaJuridicaDAO;
11
12 public class CadastroBD {
13
14     public static void main(String[] args) {
15         try (Scanner scanner = new Scanner(System.in)) {
16             PessoaFisicaDAO pfDAO = new PessoaFisicaDAO();
17             PessoaJuridicaDAO pjDAO = new PessoaJuridicaDAO();
18
19             while (true) {
20                 System.out.println("=====");
21                 System.out.println("Escolha uma opção:");
22                 System.out.println("1 - Incluir");
23                 System.out.println("2 - Alterar");
24                 System.out.println("3 - Excluir");
25                 System.out.println("4 - Exibir pelo ID");
26                 System.out.println("5 - Exibir todos");
27                 System.out.println("0 - Sair");
28                 System.out.println("=====");
29                 int opcao = scanner.nextInt();
30                 scanner.nextLine();
31
32                 if (opcao == 0) {
33                     System.out.println("Finalizando...");
34                     break;
35                 }
36
37                 System.out.println("Escolha o tipo:");
38                 System.out.println("1 - Física");
39                 System.out.println("2 - Jurídica");
40                 int tipo = scanner.nextInt();
41                 scanner.nextLine();
42
43                 try {
44                     switch (opcao) {
45                         case 1:
```

```
CadastroBDTeste.java x
Source History
1 package cadastrbd;
2
3 import java.sql.SQLException;
4 import java.util.ArrayList;
5
6 import cadastrbd.model.PessoaFisica;
7 import cadastrbd.model.PessoaFisicaDAO;
8 import cadastrbd.model.PessoaJuridica;
9 import cadastrbd.model.PessoaJuridicaDAO;
10
11 public class CadastroBDTeste {
12
13     public static void main(String[] args) {
14         PessoaFisicaDAO pfDao = new PessoaFisicaDAO();
15         PessoaJuridicaDAO pjDao = new PessoaJuridicaDAO();
16
17         try {
18
19             PessoaFisica pf = new PessoaFisica("114455669988", null, "Mateus Araujo Azevedo",
20                 "Rua Doutor Itamar Guar", "Vila Mariana", "MA", "65905215", "Mateus@gmail.com");
21
22
23             if (pf.getNome() == null || pf.getNome().trim().isEmpty()) {
24                 System.out.println("O nome est nulo ou vazio antes da chamada de incluir.");
25                 return;
26             }
27
28
29             int idPf = pfDao.incluir(pf);
30             System.out.println("Pessoa Fisica incluída com o ID: " + idPf);
31
32
33             pf.setCidade("Campo Grande");
34             pf.setEstado("MS");
35             pfDao.alterar(pf);
36
37
38             ArrayList<PessoaFisica> listaPf = pfDao.getPessoas();
39             listaPf.forEach(pessoa -> System.out.println(pessoa.getNome()));
40
41
42             pfDao.excluir(pf);
43
44
45             PessoaJuridica pj = new PessoaJuridica("74562368123", null, "Bianca Araujo Fernandes",
```

```
Pessoa.java x
Source History
package cadastrobd.model;

public class Pessoa {
    private Integer id;
    private String nome;
    private String endereco;
    private String cidade;
    private String estado;
    private String telefone;
    private String email;

    public Pessoa() {
        // Construtor padrão
    }

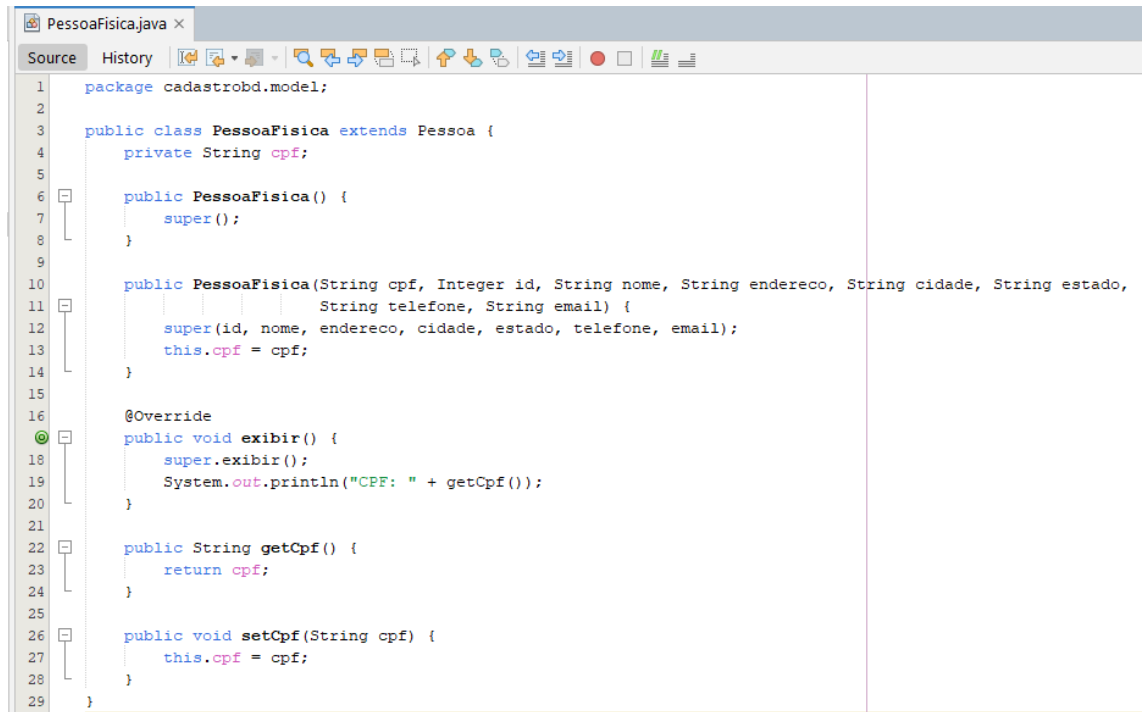
    public Pessoa(Integer id, String nome, String endereco, String cidade, String estado, String telefone, String email) {
        this.id = id;
        this.nome = nome;
        this.endereco = endereco;
        this.cidade = cidade;
        this.estado = estado;
        this.telefone = telefone;
        this.email = email;
    }

    @Override
    public void exibir() {
        System.out.println("id: " + getId());
        System.out.println("nome: " + getNome());
        System.out.println("endereco: " + getEndereco());
        System.out.println("cidade: " + getCidade());
        System.out.println("estado: " + getEstado());
        System.out.println("telefone: " + getTelefone());
        System.out.println("email: " + getEmail());
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }
}
```



```
1 package cadastrobd.model;
2
3 public class PessoaFisica extends Pessoa {
4     private String cpf;
5
6     public PessoaFisica() {
7         super();
8     }
9
10    public PessoaFisica(String cpf, Integer id, String nome, String endereco, String cidade, String estado,
11        String telefone, String email) {
12        super(id, nome, endereco, cidade, estado, telefone, email);
13        this.cpf = cpf;
14    }
15
16    @Override
17    public void exibir() {
18        super.exibir();
19        System.out.println("CPF: " + getCpf());
20    }
21
22    public String getCpf() {
23        return cpf;
24    }
25
26    public void setCpf(String cpf) {
27        this.cpf = cpf;
28    }
29 }
```

```
PessoaJuridica.java x
Source History
1 package cadastrdb.model;
2
3
4
5 public class PessoaJuridica extends Pessoa {
6
7     private String cnpj;
8
9     public PessoaJuridica() {
10         super();
11     }
12
13     public PessoaJuridica(String cnpj, Integer _id, String _nome, String _endereco, String _cidade, String _estado, String _telefone, String _email) {
14         super(_id, _nome, _endereco, _cidade, _estado, _telefone, _email);
15         this.cnpj = cnpj;
16     }
17
18     public PessoaJuridica(String number, String companyXLtda, String s, String fortaleza, String ce, String number1, String mail) {
19     }
20
21     @Override
22     public void exibir() {
23         super.exibir();
24         System.out.println("CNPJ: " + this.getCnpj());
25     }
26
27     public String getCnpj() {
28         return cnpj;
29     }
30
31
32     public void setCnpj(String cnpj) {
33         this.cnpj = cnpj;
34     }
35
36
37 }
```


















```
PessoaFisicaDAO.java x
Source History
1 package cadastrdb.model;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.ArrayList;
9
10 import cadastrdb.model.util.ConectorBD;
11
12 public class PessoaFisicaDAO {
13
14     public ConectorBD cnx = new ConectorBD();
15
16     public PessoaFisica getPessoa(Integer id) throws SQLException {
17         String sql = "SELECT pf.id_pessoa, pf.cpf, p.nome, p.endereco, p.cidade, p.estado, p.telefone, p.email "
18             + "FROM Pessoa_Fisica pf "
19             + "INNER JOIN pessoa p ON pf.id_pessoa = p.id_Pessoa "
20             + "WHERE pf.id_pessoa = !";
21         try (Connection con = cnx.getConnection();
22             PreparedStatement stmt = con.prepareStatement(sql)) {
23
24             stmt.setInt(1, id);
25             try (ResultSet rs = stmt.executeQuery()) {
26                 if (rs.next()) {
27                     PessoaFisica p = new PessoaFisica(
28                         rs.getString("cpf"),
29                         rs.getInt("id_pessoa"),
30                         rs.getString("nome"),
31                         rs.getString("endereco"),
32                         rs.getString("cidade"),
33                         rs.getString("estado"),
34                         rs.getString("telefone"),
35                         rs.getString("email")
36                     );
37                     return p;
38                 }
39             }
40         }
41         return null;
42     }
43
44     public ArrayList<PessoaFisica> getPessoas() throws SQLException {
45         ArrayList<PessoaFisica> list = new ArrayList<>();
46     }
47 }
```

```
PessoaJuridicaDAO.java x
Source History
1 package cadastrobd.model;
2
3 import cadastrobd.model.util.ConectorBD;
4
5 import java.sql.*;
6 import java.util.ArrayList;
7
8 public class PessoaJuridicaDAO {
9
10     ConectorBD cnx = new ConectorBD();
11
12     public PessoaJuridica getPessoa(Integer id) throws SQLException {
13         ResultSet rs = cnx.getSelect("""
14             select
15                 \tPessoa_Juridica.id_pessoa as id,
16                 \tPessoa_Juridica.cnpj,
17                 \tp.nome,
18                 \tp.endereco,
19                 \tp.cidade,
20                 \tp.estado,
21                 \tp.telefone,
22                 \tp.email
23             \tfrom Pessoa_Juridica
24             INNER JOIN Pessoa as p on Pessoa_Juridica.id_pessoa = p.id_Pessoa
25             WHERE
26                 \tPessoa_Juridica.id_pessoa = "" + id.toString());
27
28         rs.next();
29         PessoaJuridica p = new PessoaJuridica(
30             rs.getString("cnpj"),
31             rs.getInt("id"),
32             rs.getString("nome"),
33             rs.getString("endereco"),
34             rs.getString("cidade"),
35             rs.getString("estado"),
36             rs.getString("telefone"),
37             rs.getString("email")
38         );
39         p.exibir();
40         cnx.close();
41         return p;
42     }
43
44     public ArrayList<PessoaJuridica> getPessoas() throws SQLException {
45         ArrayList<PessoaJuridica> list = new ArrayList<>();
```

```
SequenceManager.java x
Source History
1 package cadastrabd.model.util;
2
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5
6 public class SequenceManager {
7
8     ConectorBD cnx = new ConectorBD();
9
10    public int getValue(String sequenceName) throws SQLException {
11        String sql = "SELECT NEXT VALUE FOR " + sequenceName;
12        ResultSet rs = cnx.getSelect(sql);
13        if (rs.next()) {
14            return rs.getInt(1);
15        } else {
16            throw new SQLException("Não foi possível obter o próximo valor da sequência: " + sequenceName);
17        }
18    }
19 }
```

```
ConectorBD.java x
Source History
1 package cadastrabd.model.util;
2
3 import java.sql.*;
4
5 public class ConectorBD {
6
7     public Connection con;
8     public PreparedStatement stmt;
9     public ResultSet rs;
10
11    public Connection getConnection() throws SQLException {
12        String url = "jdbc:sqlserver://localhost:1433;databaseName=loja;encrypt=true;trustServerCertificate=true";
13        String user = "sa";
14        String password = "loja";
15        con = DriverManager.getConnection(url, user, password);
16        return con;
17    }
18
19    public ResultSet getSelect(String sql) throws SQLException {
20        stmt = getConnection().prepareStatement(sql);
21        rs = stmt.executeQuery();
22        return rs;
23    }
24
25    public int insert(String sql) throws SQLException {
26        stmt = getConnection().prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
27        stmt.executeUpdate();
28        rs = stmt.getGeneratedKeys();
29        if (rs.next()) {
30            return rs.getInt(1);
31        } else {
32            throw new SQLException("Falha ao inserir dados.");
33        }
34    }
35
36    public boolean update(String sql) throws SQLException {
37        stmt = getConnection().prepareStatement(sql);
38        int affectedRows = stmt.executeUpdate();
39        return affectedRows > 0;
40    }
41
42    public void close() throws SQLException {
43        if (rs != null && !rs.isClosed()) rs.close();
44        if (stmt != null && !stmt.isClosed()) stmt.close();
45        if (con != null && !con.isClosed()) con.close();
46    }
47 }
```



- ✓  CadastroBD
  - ✓  Source Packages
    - ✓  cadastrobd
      -  CadastroBD.java
      -  CadastroBDTeste.java
    - ✓  cadastrobd.model
      -  Pessoa.java
      -  PessoaFisica.java
      -  PessoaFisicaDAO.java
      -  PessoaJuridica.java
      -  PessoaJuridicaDAO.java
    - ✓  cadastrobd.model.util
      -  ConectorBD.java
      -  SequenceManager.java
  - ✓  Libraries
    - >  mssql-jdbc-12.2.0.jre11.jar
    - >  JDK 17 (Default)

## Resultados

```
Output - CadastroBD (run) #4 x
run:
=====
Escolha uma opção:
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir pelo ID
5 - Exibir todos
0 - Sair
=====

1
Escolha o tipo:
1 - Física
2 - Jurídica
1
Digite o nome:
Mateus Araujo Azevedo
Digite o CPF:
114455669988
Digite o endereço:
Rua Doutor Itamar Guará
Digite a cidade:
Vila Mariana
Digite o estado:
MA
Digite o telefone:
65905215
Digite o email:
Mateus@gmail.com
```

```
run:
=====
Escolha uma opção:
1 - Incluir
2 - Alterar
3 - Excluir
4 - Exibir pelo ID
5 - Exibir todos
0 - Sair
=====
1
Escolha o tipo:
1 - Física
2 - Jurídica
2
Digite o nome:
Bianca Araujo Fernandes
Digite o CNPJ:
74562368123
Digite o endereço:
Avenida Dom Pedro II
Digite a cidade:
PortoAlegre
Digite o estado:
PO
Digite o telefone:
8845612398
Digite o email:
Bianca@outlook.com
```

## Análise e Conclusão

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

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel crucial no desenvolvimento de software, atuando como intermediários entre diferentes camadas de uma aplicação.

O JDBC, especificamente, é projetado para fornecer uma interface padrão para conectar aplicativos Java a bancos de dados relacionais. Isso é fundamental porque permite que os desenvolvedores interajam com bancos de dados de maneira uniforme, independentemente do sistema de gerenciamento de banco de dados subjacente.

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

Statement e PreparedStatement são duas interfaces na linguagem de programação Java usadas para executar consultas SQL em um banco de dados. A principal diferença entre elas está na forma como lidam com os parâmetros nas consultas.

#### Statement:

O Statement é usado para executar consultas SQL simples sem parâmetros.

As consultas são criadas como strings e enviadas diretamente ao banco de dados para execução.

Vulnerável a ataques de injeção de SQL se não for usado com cuidado, pois não realiza tratamento automático de escape de caracteres especiais.

#### PreparedStatement:

O PreparedStatement é usado quando você precisa executar consultas SQL parametrizadas.

As consultas parametrizadas são pré-compiladas, o que melhora o desempenho se a mesma consulta for executada várias vezes com diferentes parâmetros.

Previne ataques de injeção de SQL, pois os parâmetros são tratados automaticamente, eliminando a necessidade de escapar manualmente caracteres especiais.

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

O padrão DAO (Data Access Object) é um padrão de design de software que tem como objetivo separar a lógica de acesso a dados do restante da aplicação. Ele fornece uma abstração entre a lógica de negócios da aplicação e os detalhes de como os dados são armazenados e recuperados. A adoção do padrão DAO pode trazer benefícios significativos para a manutenibilidade do software.

O DAO encapsula toda a lógica relacionada ao acesso a dados em uma camada separada. Isso significa que as mudanças na forma como os dados são armazenados (como a mudança de um banco de dados relacional para um banco de dados NoSQL) podem ser feitas no DAO sem afetar o restante da aplicação. Isso reduz a dependência do código de negócios em detalhes específicos de armazenamento de dados.

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

Quando lidamos com um modelo estritamente relacional em bancos de dados, a herança é frequentemente implementada usando uma abordagem conhecida como modelagem de herança. Existem basicamente três estratégias comuns para representar herança em bancos de dados relacionais: herança única tabela, herança tabela por classe e herança tabela por subclasse.