



## **Trabalho Prático | PROGRAMACAO BACK-END COM JAVA**

**Érica Abrantes de Oliveira Lima Ignatios 202405045475**

**Campus Conselheiro Lafaiete - MG**

**DGT2821 Programação Back-end com Java – 4 Semestre**

### **Objetivo da Prática**

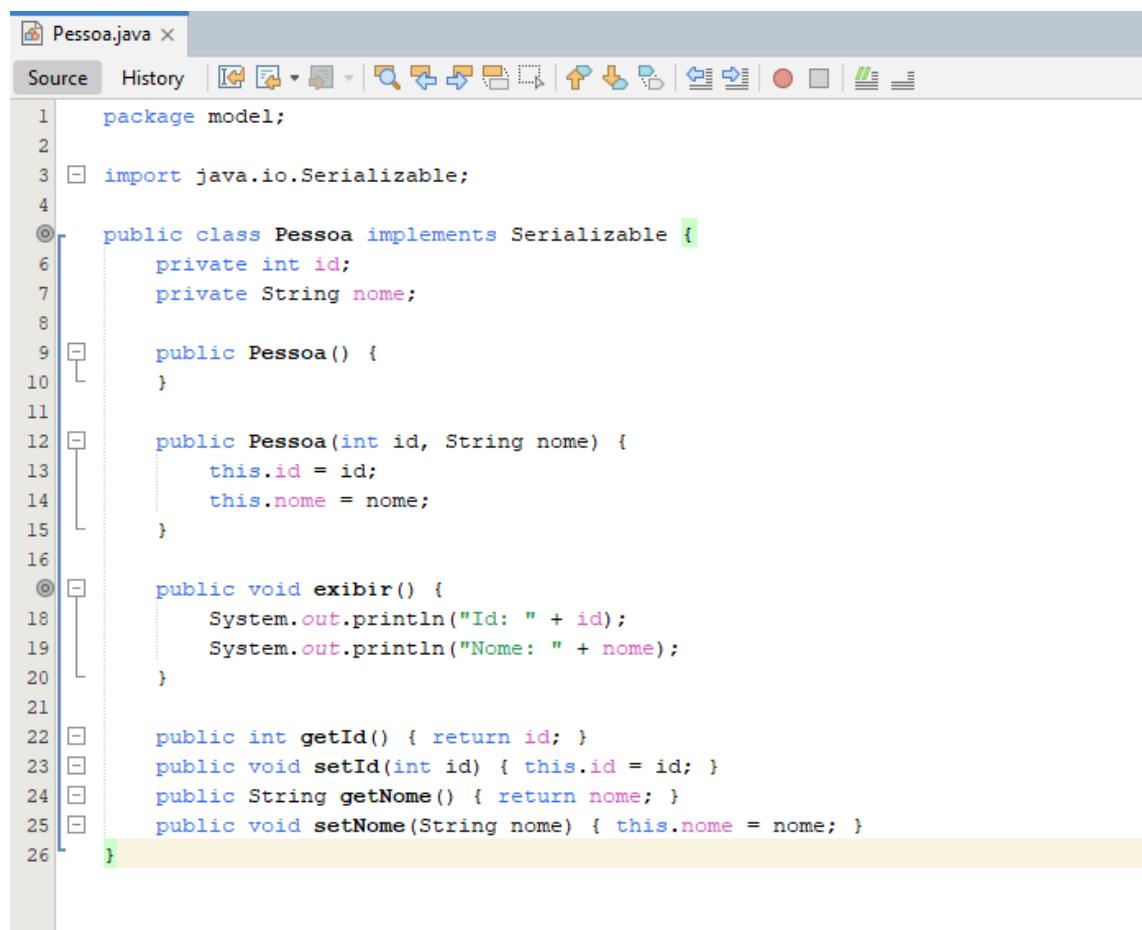
O objetivo desta prática foi implementar um sistema de cadastro de clientes em Java utilizando o paradigma de Orientação a Objetos. O projeto focou na utilização de herança e polimorfismo para definição das entidades (Pessoa, Pessoa Física e Pessoa Jurídica) e na implementação da persistência de dados em arquivos binários (serialização), permitindo salvar e recuperar as informações do disco. Além disso, buscou-se exercitar o tratamento de exceções e a manipulação de fluxos de entrada e saída na linguagem Java

### **1º Procedimento | Criação das Entidades e Sistema de Persistência**

#### **1º Procedimento | Criação das Entidades e Sistema de Persistência**

Abaixo estão os códigos fonte desenvolvidos para a criação do modelo de dados e dos repositórios responsáveis pela persistência, lembrando que os códigos se encontram no repositório do github no link : <https://github.com/ericaabrantes/cadastroPOO-Java>

## 1. Classe Pessoa.java (Superclasse)



The screenshot shows a Java code editor window titled "Pessoa.java". The menu bar includes "Source" and "History". The toolbar contains various icons for file operations like new, open, save, and search. The code itself is as follows:

```
1 package model;
2
3 import java.io.Serializable;
4
5 public class Pessoa implements Serializable {
6     private int id;
7     private String nome;
8
9     public Pessoa() {
10    }
11
12    public Pessoa(int id, String nome) {
13        this.id = id;
14        this.nome = nome;
15    }
16
17    public void exibir() {
18        System.out.println("Id: " + id);
19        System.out.println("Nome: " + nome);
20    }
21
22    public int getId() { return id; }
23    public void setId(int id) { this.id = id; }
24    public String getNome() { return nome; }
25    public void setNome(String nome) { this.nome = nome; }
26}
```

## 2. Classe PessoaFisica.java

The screenshot shows a Java code editor window titled "PessoaFisica.java". The menu bar includes "Source" and "History". The toolbar contains various icons for file operations like new, open, save, and search. The code itself is as follows:

```
1 package model;
2
3 import java.io.Serializable;
4
5 public class PessoaFisica extends Pessoa implements Serializable {
6     private String cpf;
7     private int idade;
8
9     public PessoaFisica() {
10    }
11
12
13     public PessoaFisica(int id, String nome, String cpf, int idade) {
14         super(id, nome);
15         this.cpf = cpf;
16         this.idade = idade;
17     }
18
19     @Override
20     public void exibir() {
21         super.exibir();
22         System.out.println("CPF: " + cpf);
23         System.out.println("Idade: " + idade);
24     }
25
26     public String getCpf() { return cpf; }
27     public void setCpf(String cpf) { this.cpf = cpf; }
28     public int getIdade() { return idade; }
29     public void setIdade(int idade) { this.idade = idade; }
30 }
```

### 3. Classe PessoaJuridica.java

The screenshot shows a Java code editor window with the title bar "PessoaJuridica.java x". The menu bar includes "Source" and "History". Below the menu is a toolbar with various icons for file operations. The code editor displays the following Java code:

```
1 package model;
2
3 import java.io.Serializable;
4
5 public class PessoaJuridica extends Pessoa implements Serializable {
6     private String cnpj;
7
8     public PessoaJuridica() {
9     }
10
11    public PessoaJuridica(int id, String nome, String cnpj) {
12        super(id, nome);
13        this.cnpj = cnpj;
14    }
15
16    @Override
17    public void exibir() {
18        super.exibir();
19        System.out.println("CNPJ: " + cnpj);
20    }
21
22    public String getCnpj() { return cnpj; }
23    public void setCnpj(String cnpj) { this.cnpj = cnpj; }
24}
```

The code defines a class PessoaJuridica that extends Pessoa and implements Serializable. It has a private attribute cnpj and two constructors. The first constructor is the default. The second constructor takes an int id, a String nome, and a String cnpj, calling the super constructor and setting the cnpj attribute. It also overrides the exibir method to print the CNPJ value. Finally, it provides getters and setters for the cnpj attribute.

#### 4. Classe PessoaFisicaRepo.java (Repositório)

The screenshot shows a Java code editor window with the title bar "PessoaFisicaRepo.java". The menu bar includes "Source" and "History". Below the menu is a toolbar with various icons. The code itself is a Java class named "PessoaFisicaRepo" that manages a list of "PessoaFisica" objects using an ArrayList. It includes methods for inserting, updating, deleting, and retrieving individuals by ID, as well as persisting the list to a file and reading it back.

```
1 package model;
2 import java.io.*;
3 import java.util.ArrayList;
4
5 public class PessoaFisicaRepo {
6     private ArrayList<PessoaFisica> lista = new ArrayList<>();
7
8     public void inserir(PessoaFisica pessoa) {
9         lista.add(pessoa);
10    }
11
12    public void alterar(PessoaFisica pessoaNova) {
13        for (int i = 0; i < lista.size(); i++) {
14            if (lista.get(i).getId() == pessoaNova.getId()) {
15                lista.set(i, pessoaNova);
16                return;
17            }
18        }
19    }
20
21    public void excluir(int id) {
22        lista.removeIf(p -> p.getId() == id);
23    }
24
25    public PessoaFisica obter(int id) {
26        for (PessoaFisica p : lista) {
27            if (p.getId() == id) {
28                return p;
29            }
30        }
31        return null;
32    }
33
34    public ArrayList<PessoaFisica> obterTodos() {
35        return lista;
36    }
37
38    public void persistir(String nomeArquivo) throws IOException {
39        try (ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
40            output.writeObject(lista);
41        }
42    }
43
44    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
45        try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
46            lista = (ArrayList<PessoaFisica>) input.readObject();
47        }
48    }
49}
```

## 5. Classe PessoaJuridicaRepo.java (Repositório)

The screenshot shows a Java code editor window with the title bar "PessoaJuridicaRepo.java". The menu bar includes "Source", "History", and various tool icons. The code itself is a Java class named "PessoaJuridicaRepo" that manages a list of "PessoaJuridica" objects using an ArrayList. It includes methods for inserting, updating, deleting, and retrieving individuals by ID, as well as persisting the list to a file and reading it back.

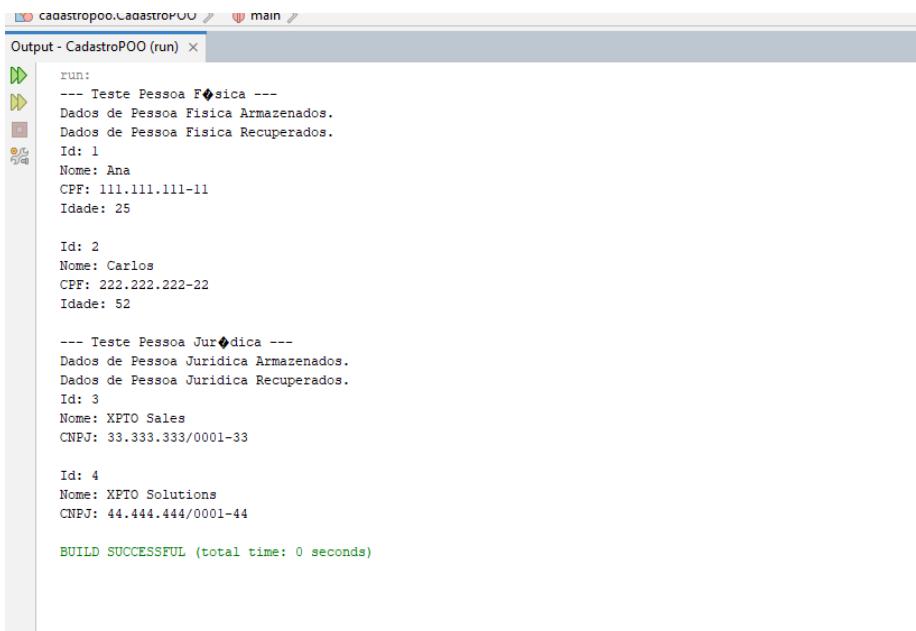
```
1 package model;
2
3 import java.io.*;
4 import java.util.ArrayList;
5
6 public class PessoaJuridicaRepo {
7     private ArrayList<PessoaJuridica> lista = new ArrayList<>();
8
9     public void inserir(PessoaJuridica pessoa) {
10         lista.add(pessoa);
11     }
12
13     public void alterar(PessoaJuridica pessoaNova) {
14         for (int i = 0; i < lista.size(); i++) {
15             if (lista.get(i).getId() == pessoaNova.getId()) {
16                 lista.set(i, pessoaNova);
17                 return;
18             }
19         }
20     }
21
22     public void excluir(int id) {
23         lista.removeIf(p -> p.getId() == id);
24     }
25
26     public PessoaJuridica obter(int id) {
27         for (PessoaJuridica p : lista) {
28             if (p.getId() == id) {
29                 return p;
30             }
31         }
32         return null;
33     }
34
35     public ArrayList<PessoaJuridica> obterTodos() {
36         return lista;
37     }
38
39     public void persistir(String nomeArquivo) throws IOException {
40         try (ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
41             output.writeObject(lista);
42         }
43     }
44
45     public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
46         try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
47             lista = (ArrayList<PessoaJuridica>) input.readObject();
48         }
49     }
50 }
```

## 6. Classe CadastroPOO.java

The screenshot shows a Java code editor window titled "CadastroPOO.java x". The menu bar includes "Source" and "History". The toolbar contains various icons for file operations like new, open, save, and search. The code itself is a main method that interacts with two repository classes: PessoaFisicaRepo and PessoaJuridicaRepo. It demonstrates the insertion of physical and legal persons into binary files and their subsequent retrieval.

```
1 package cadastropoo;
2 import model.*;
3 import java.io.IOException;
4
5 public class CadastroPOO {
6
7     public static void main(String[] args) {
8         try {
9             System.out.println("---- Teste Pessoa Física ---");
10            PessoaFisicaRepo repol = new PessoaFisicaRepo();
11            repol.inserir(new PessoaFisica(1, "Ana", "111.111.111-11", 25));
12            repol.inserir(new PessoaFisica(2, "Carlos", "222.222.222-22", 52));
13
14            repol.persistir("pessoas-fisicas.bin");
15            System.out.println("Dados de Pessoa Física Armazenados.");
16
17            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
18            repo2.recuperar("pessoas-fisicas.bin");
19            System.out.println("Dados de Pessoa Física Recuperados.");
20
21            for (PessoaFisica p : repo2.obterTodos()) {
22                p.exibir();
23                System.out.println();
24            }
25
26            System.out.println("---- Teste Pessoa Jurídica ---");
27
28            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
29            repo3.inserir(new PessoaJuridica(3, "XPTO Sales", "33.333.333/0001-33"));
30            repo3.inserir(new PessoaJuridica(4, "XPTO Solutions", "44.444.444/0001-44"));
31
32
33            repo3.persistir("pessoas-juridicas.bin");
34            System.out.println("Dados de Pessoa Jurídica Armazenados.");
35
36
37            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
38            repo4.recuperar("pessoas-juridicas.bin");
39            System.out.println("Dados de Pessoa Jurídica Recuperados.");
40
41
42            for (PessoaJuridica p : repo4.obterTodos()) {
43                p.exibir();
44                System.out.println();
45            }
46
47        } catch (IOException | ClassNotFoundException e) {
48            System.out.println("Erro: " + e.getMessage());
49            e.printStackTrace();
50        }
51    }
52}
```

## Output da Missao 1



```
cadastropoo.CadastroPOO main
Output - CadastroPOO (run) ×

run:
--- Teste Pessoa Física ---
Dados de Pessoa Física Armazenados.
Dados de Pessoa Física Recuperados.
Id: 1
Nome: Ana
CPF: 111.111.111-11
Idade: 25

Id: 2
Nome: Carlos
CPF: 222.222.222-22
Idade: 52

--- Teste Pessoa Jurídica ---
Dados de Pessoa Jurídica Armazenados.
Dados de Pessoa Jurídica Recuperados.
Id: 3
Nome: XPTO Sales
CNPJ: 33.333.333/0001-33

Id: 4
Nome: XPTO Solutions
CNPJ: 44.444.444/0001-44

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

- Quais as vantagens e desvantagens do uso de herança?

A principal vantagem é o reaproveitamento de código, permitindo que atributos e métodos comuns (como id e nome na classe Pessoa) sejam definidos uma única vez e herdados pelas subclasses (PessoaFísica, PessoaJurídica). Isso também facilita o uso do polimorfismo. A desvantagem é o acoplamento forte criado entre as classes; alterações na superclasse afetam todas as subclasses, o que pode tornar a manutenção complexa e propagar erros se não for bem planejada.

- Por que a interface Serializable é necessária ao efetuar persistência em arquivos binários?

Ela atua como uma interface de marcação para a Java Virtual Machine (JVM). Ela informa que a classe está autorizada a ter seu estado (objetos) convertido em uma sequência de bytes. Sem implementar essa interface, o uso das classes de fluxo de objetos (como ObjectOutputStream) lançaria uma exceção do tipo NotSerializableException ao tentar gravar os dados.

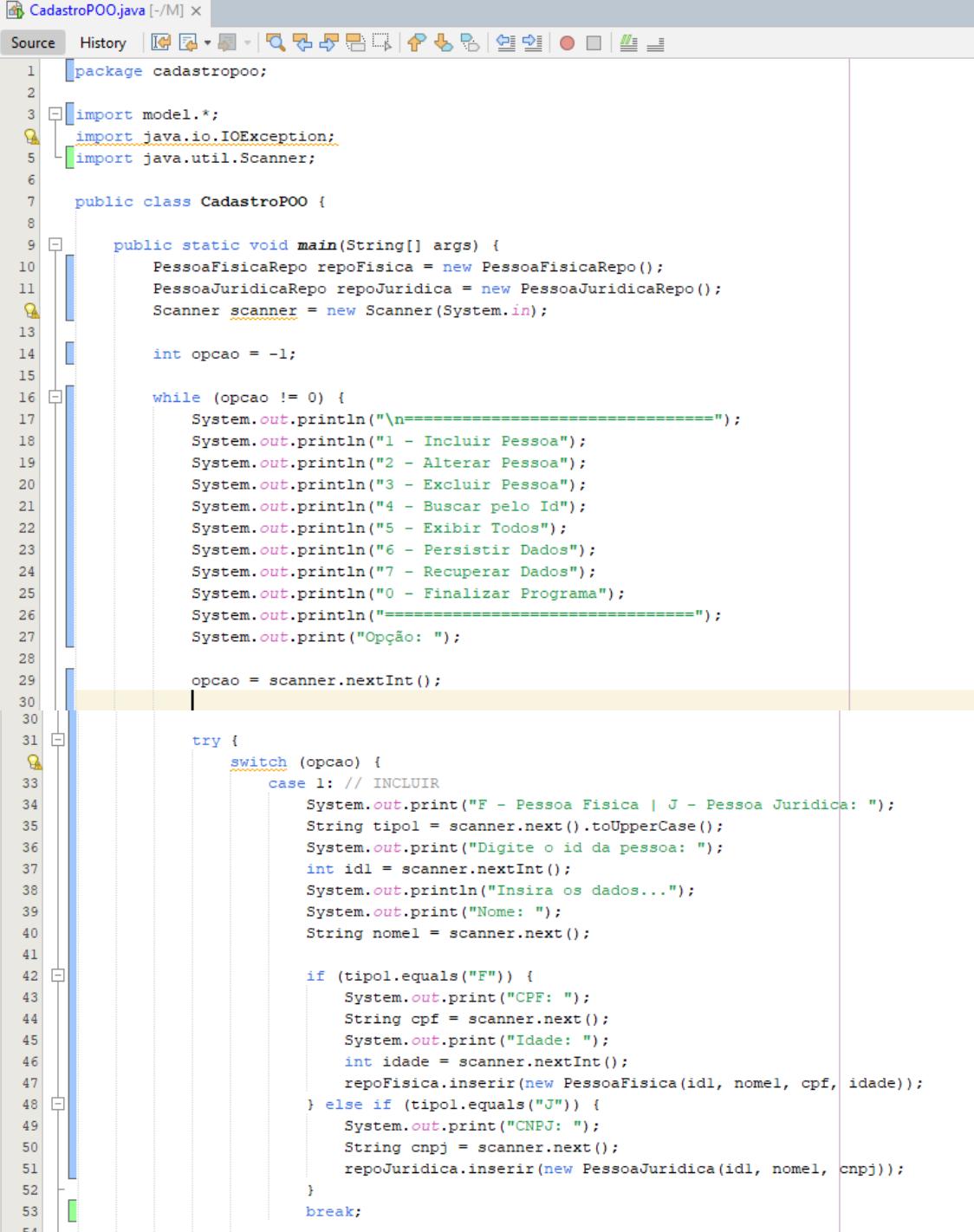
- Como o paradigma funcional é utilizado pela API stream no Java?

Ele é utilizado através de Expressões Lambda e da API de Streams para manipular coleções de forma declarativa e concisa. No projeto, utilizamos esse conceito no método excluir dos repositórios: lista.removeIf(p -> p.getId() == id);. Nesse caso, passa-se uma função (predicado) como argumento para o método removeIf, evitando a necessidade de escrever laços de repetição explícitos e focando no "o que" deve ser feito, e não no "como".

- d. Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?

Neste projeto, adotou-se o padrão DAO (Data Access Object) ou Repository Pattern. Criou-se classes específicas (PessoaFisicaRepo e PessoaJuridicaRepo) cuja única responsabilidade é gerenciar o acesso aos dados (CRUD) e a persistência (salvar/recuperar). Isso promove uma separação clara entre a lógica de persistência e o restante da aplicação, facilitando a organização e a manutenção do código.

## 2º Procedimento | Criação do Cadastro em Modo Texto



```

1 package cadastropoo;
2
3 import model.*;
4 import java.io.IOException;
5 import java.util.Scanner;
6
7 public class CadastroPOO {
8
9     public static void main(String[] args) {
10         PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
11         PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();
12         Scanner scanner = new Scanner(System.in);
13
14         int opcao = -1;
15
16         while (opcao != 0) {
17             System.out.println("\n=====");
18             System.out.println("1 - Incluir Pessoa");
19             System.out.println("2 - Alterar Pessoa");
20             System.out.println("3 - Excluir Pessoa");
21             System.out.println("4 - Buscar pelo Id");
22             System.out.println("5 - Exibir Todos");
23             System.out.println("6 - Persistir Dados");
24             System.out.println("7 - Recuperar Dados");
25             System.out.println("0 - Finalizar Programa");
26             System.out.println("=====");
27             System.out.print("Opção: ");
28
29             opcao = scanner.nextInt();
30
31             try {
32                 switch (opcao) {
33                     case 1: // INCLUIR
34                         System.out.print("F - Pessoa Física | J - Pessoa Jurídica: ");
35                         String tipol = scanner.next().toUpperCase();
36                         System.out.print("Digite o id da pessoa: ");
37                         int idl = scanner.nextInt();
38                         System.out.println("Insira os dados...");
39                         System.out.print("Nome: ");
40                         String nomel = scanner.next();
41
42                         if (tipol.equals("F")) {
43                             System.out.print("CPF: ");
44                             String cpf = scanner.next();
45                             System.out.print("Idade: ");
46                             int idade = scanner.nextInt();
47                             repoFisica.inserir(new PessoaFisica(idl, nomel, cpf, idade));
48                         } else if (tipol.equals("J")) {
49                             System.out.print("CNPJ: ");
50                             String cnpj = scanner.next();
51                             repoJuridica.inserir(new PessoaJuridica(idl, nomel, cnpj));
52                         }
53                         break;
54                 }
55             } catch (IOException e) {
56                 System.out.println("Erro ao ler o arquivo.");
57             }
58         }
59     }
60 }

```

```

54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130

        case 2: // ALTERAR
        System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
        String tipo2 = scanner.next().toUpperCase();
        System.out.print("Digite o id da pessoa: ");
        int id2 = scanner.nextInt();

        if (tipo2.equals("F")) {
            PessoaFisica p = repoFisica.obter(id2);
            if (p != null) {
                System.out.println("Dados atuais:"); p.exibir();
                System.out.print("Novo Nome: "); String n = scanner.next();
                System.out.print("Novo CPF: "); String c = scanner.next();
                System.out.print("Nova Idade: "); int i = scanner.nextInt();
                repoFisica.alterar(new PessoaFisica(id2, n, c, i));
            }
        } else if (tipo2.equals("J")) {
            PessoaJuridica p = repoJuridica.obter(id2);
            if (p != null) {
                System.out.println("Dados atuais:"); p.exibir();
                System.out.print("Novo Nome: "); String n = scanner.next();
                System.out.print("Novo CNPJ: "); String c = scanner.next();
                repoJuridica.alterar(new PessoaJuridica(id2, n, c));
            }
        }
        break;

    case 3: // EXCLUIR
        System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
        String tipo3 = scanner.next().toUpperCase();
        System.out.print("Digite o id da pessoa: ");
        int id3 = scanner.nextInt();
        if (tipo3.equals("F")) repoFisica.excluir(id3);
        else if (tipo3.equals("J")) repoJuridica.excluir(id3);
        break;

    case 4: // BUSCAR
        System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
        String tipo4 = scanner.next().toUpperCase();
        System.out.print("Digite o id da pessoa: ");
        int id4 = scanner.nextInt();
        if (tipo4.equals("F")) {
            PessoaFisica p = repoFisica.obter(id4);
            if(p!=null) p.exibir();
        } else if (tipo4.equals("J")) {
            PessoaJuridica p = repoJuridica.obter(id4);
            if(p!=null) p.exibir();
        }
        break;

    case 5: // EXIBIR TODOS
        System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
        String tipo5 = scanner.next().toUpperCase();
        if (tipo5.equals("F")) for(PessoaFisica p : repoFisica.obterTodos()) { p.exibir(); System.out.println(); }
        else if (tipo5.equals("J")) for(PessoaJuridica p : repoJuridica.obterTodos()) { p.exibir(); System.out.println(); }
        break;

    case 6: // SALVAR
        System.out.print("Digite o prefixo dos arquivos: ");
        String prefixoSalvar = scanner.next();
        repoFisica.persistir(prefixoSalvar + ".fisica.bin");
        repoJuridica.persistir(prefixoSalvar + ".juridica.bin");
        System.out.println("Dados salvos.");
        break;

    case 7: // RECUPERAR
        System.out.print("Digite o prefixo dos arquivos: ");
        String prefixoRec = scanner.next();
        repoFisica.recuperar(prefixoRec + ".fisica.bin");
        repoJuridica.recuperar(prefixoRec + ".juridica.bin");
        System.out.println("Dados recuperados.");
        break;

    case 0:
        System.out.println("Finalizando...");
        break;

```

```
130
131
132
133
134     default:
135         System.out.println("Opção inválida.");
136     }
137 }
138 scanner.close();
139
140 }
```

## Output da Missão 2

```
Output - CadastroPOO (run) ×
run:
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====
Opção: 1
F - Pessoa Física | J - Pessoa Jurídica: f
Digite o id da pessoa: 120
Insira os dados...
Nome: Erica
CPF: 12345678933
Idade: 29

=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====
Opção: |
```

```
CadastroPOO (run) X CadastroPOO (run) #4 X

=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====

Opção: 5
F - Pessoa Física | J - Pessoa Jurídica: f
Id: 120
Nome: Erica
CPF: 12345678933
Idade: 29

=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====

Opção: 0
Finalizando...
BUILD SUCCESSFUL (total time: 4 minutes 21 seconds)
```

- a. O que são elementos estáticos e qual o motivo para o método main adotar esse modificador?

Elementos estáticos (static) em Java são membros (atributos ou métodos) que pertencem à classe em si, e não a uma instância específica (objeto) dessa classe. Isso significa que eles podem ser acessados sem a necessidade de criar um objeto. O método main adota esse modificador porque ele é o ponto de entrada da aplicação; a Java Virtual Machine (JVM) precisa invocá-lo para iniciar o programa antes que qualquer objeto da classe principal (CadastroPOO) tenha sido criado (instanciado).

- b. Para que serve a classe Scanner?

Está presente no pacote `java.util`, é utilizada para ler e analisar textos (strings) e tipos primitivos usando expressões regulares. Neste trabalho prático, ela foi utilizada para capturar os dados de entrada fornecidos pelo usuário através do console (fluxo de entrada padrão `System.in`), permitindo a interatividade do menu (leitura de opções numéricas, nomes, CPFs, etc.).

c. Como o uso de classes de repositório impactou na organização do código?

O uso das classes de repositório (`PessoaFisicaRepo` e `PessoaJuridicaRepo`) foi fundamental para aplicar o princípio de Separação de Responsabilidades. Elas encapsularam toda a lógica de manipulação de dados (inserir, excluir, buscar) e a complexidade da persistência em arquivos binários. Com isso, a classe principal (`CadastroPOO`) ficou responsável apenas pela interface com o usuário (`Menu`), tornando o código mais organizado, legível e facilitando a manutenção futura.

## Conclusão

A execução desta Missão Prática permitiu consolidar os fundamentos essenciais da Programação Orientada a Objetos (POO) e sua aplicação em sistemas de persistência de dados. A implementação do sistema "CadastroPOO" demonstrou a eficácia do uso de herança e polimorfismo, permitindo a criação de um código modular e reutilizável, onde a classe genérica `Pessoa` centralizou atributos comuns, facilitando a manutenção e a extensão para as entidades `PessoaFisica` e `PessoaJuridica`.

Do ponto de vista da persistência, o uso da interface `Serializable` em conjunto com os fluxos de entrada e saída (`ObjectInputStream` e  `ObjectOutputStream`) provou ser uma solução nativa e eficiente para o armazenamento de objetos em arquivos binários. Embora essa abordagem não substitua um Banco de Dados Relacional (SGBD) em cenários de alta complexidade ou concorrência, ela é fundamental para compreender como a Java Virtual Machine (JVM) manipula o estado dos objetos e a importância da serialização no tráfego de dados.

A organização do projeto, com a separação clara entre as classes de modelo (`Model`), as classes de repositório (lógica de acesso a dados) e a classe principal (interface com o usuário), evidenciou a importância de padrões de projeto e da separação de responsabilidades. Isso resultou em um código limpo, onde a lógica de negócios não se mistura com a lógica de apresentação.

Por fim, o desenvolvimento do menu interativo em modo texto proporcionou uma visão prática sobre o tratamento de exceções e o controle de fluxo, reforçando a importância de validar entradas do usuário para garantir a robustez da aplicação. O trabalho foi fundamental para servir de base sólida para futuras implementações.