Campus: Nova Friburgo - Olaria **Curso:** 7124 - Desenvolvimento Full Stack

Disciplina: DGT2821 - Programação Back-end com Java **Turma:** 9001

Semestre: 2025.2

Aluno: Joilson Til Vieira - **202409182973**

RELATÓRIO DE PRÁTICA: CADASTRO POO

1. Título da Prática

Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java.

2. Objetivo da Prática

Utilizar herança e polimorfismo na definição de entidades, aplicar a persistência de objetos em arquivos binários, implementar uma interface cadastral em modo texto e utilizar o controle de exceções da plataforma Java. O objetivo final é demonstrar habilidades na organização de código utilizando repositórios e o padrão MVC simplificado.

3. Link do Repositório (GitHub)

Abaixo encontra-se o link para o repositório contendo todo o código-fonte do projeto:

https://github.com/JoilsonVieira/CadastroPOO

4. Códigos Desenvolvidos

Abaixo estão os códigos fonte das classes implementadas no pacote `model` e no pacote principal `cadastropoo`.

4.1. Classe Pessoa (Pessoa.java)

```
package model;

import java.io.Serializable;


public class Pessoa implements Serializable {
   private int id;
   private String nome;


   public Pessoa() { }


   public Pessoa(int id, String nome) {
      this.id = id;
      this.nome = nome;
   }


   public int getId() { return id; }
   public void setId(int id) { this.id = id; }
   public String getNome() { return nome; }
   public void setNome(String nome) { this.nome = nome; }


   public void exibir() {
      System.out.println("ID: " + id);
      System.out.println("Nome: " + nome);
   }
```

```
}
```

4.2. Classe PessoaFisica (PessoaFisica.java):

```
package model;
import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private String cpf;
    private int idade;

    public PessoaFisica() { }

    public PessoaFisica(int id, String nome, String cpf, int idade) {
        super(id, nome);
        this.cpf = cpf;
        this.idade = idade;
    }

    public String getCpf() { return cpf; }
    public void setCpf(String cpf) { this.cpf = cpf; }
    public int getIdade() { return idade; }
    public void setIdade(int idade) { this.idade = idade; }

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

```
        System.out.println("Idade: " + idade);
    }
}
```

4.3. Classe PessoaJuridica (PessoaJuridica.java):

```
package model;
import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;

    public PessoaJuridica() { }

    public PessoaJuridica(int id, String nome, String cnpj) {
        super(id, nome);
        this.cnpj = cnpj;
    }

    public String getCnpj() { return cnpj; }
    public void setCnpj(String cnpj) { this.cnpj = cnpj; }

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

4.4. Repositório Pessoa Física (PessoaFisicaRepo.java):

```
package model;
import java.io.*;
import java.util.ArrayList;

public class PessoaFisicaRepo {
   private ArrayList<PessoaFisica> listaPessoas = new ArrayList<>();

   public void inserir(PessoaFisica pessoa) { listaPessoas.add(pessoa); }

   public void alterar(PessoaFisica pessoaAlterada) {
      for (int i = 0; i < listaPessoas.size(); i++) {
         if (listaPessoas.get(i).getId() == pessoaAlterada.getId()) {
            listaPessoas.set(i, pessoaAlterada);
            return;
         }
      }
   }

   public void excluir(int id) { listaPessoas.removeIf(p -> p.getId() == id); }

   public PessoaFisica obter(int id) {
      for (PessoaFisica p : listaPessoas) {
         if (p.getId() == id) return p;
      }
      return null;
```

```
    }

    public ArrayList<PessoaFisica> obterTodos() { return listaPessoas; }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            out.writeObject(listaPessoas);
            System.out.println("Dados de Pessoa Fisica armazenados.");
        }
    }

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoas = (ArrayList<PessoaFisica>) in.readObject();
            System.out.println("Dados de Pessoa Fisica recuperados.");
        }
    }
}
```

4.5. Repositório Pessoa Jurídica (PessoaJuridicaRepo.java):

```
package model;
import java.io.*;
import java.util.ArrayList;

public class PessoaJuridicaRepo {
```

```
  private ArrayList<PessoaJuridica> listaPessoas = new ArrayList<>();

  public void inserir(PessoaJuridica pessoa) { listaPessoas.add(pessoa); }

  public void alterar(PessoaJuridica pessoaAlterada) {
    for (int i = 0; i < listaPessoas.size(); i++) {
      if (listaPessoas.get(i).getId() == pessoaAlterada.getId()) {
        listaPessoas.set(i, pessoaAlterada);
        return;
      }
    }
  }

  public void excluir(int id) { listaPessoas.removeIf(p -> p.getId() == id); }

  public PessoaJuridica obter(int id) {
    for (PessoaJuridica p : listaPessoas) {
      if (p.getId() == id) return p;
    }
    return null;
  }

  public ArrayList<PessoaJuridica> obterTodos() { return listaPessoas; }

  public void persistir(String nomeArquivo) throws IOException {
    try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
      out.writeObject(listaPessoas);
```

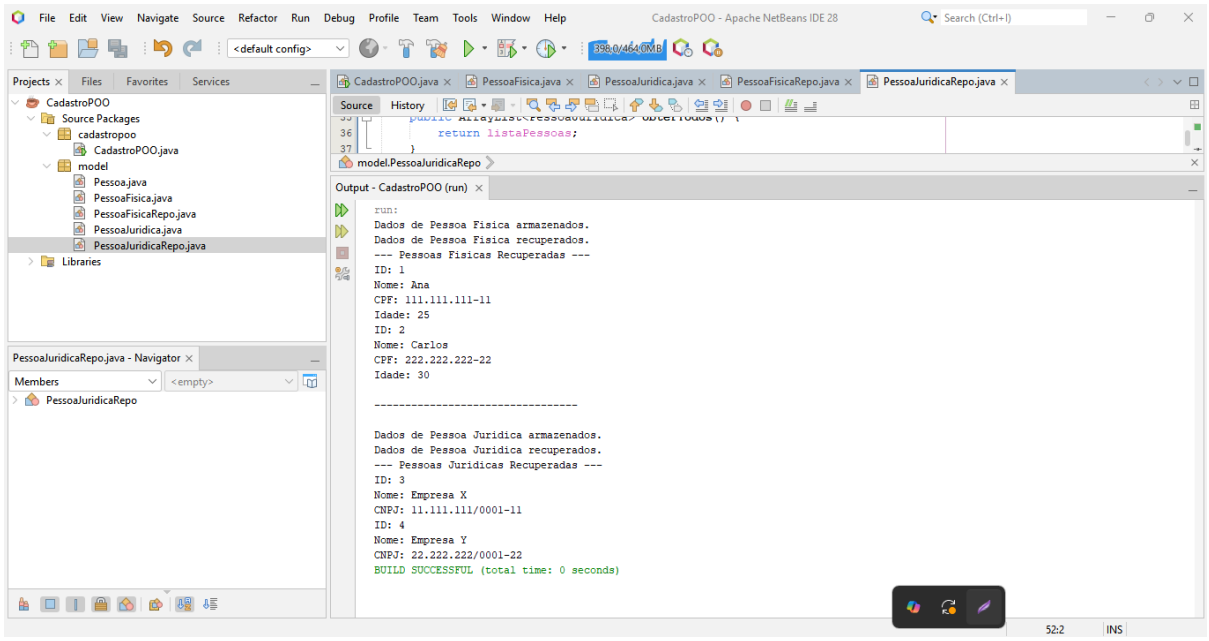
```
        System.out.println("Dados de Pessoa Juridica armazenados.");
    }
  }

  public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
    try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
        listaPessoas = (ArrayList<PessoaJuridica>) in.readObject();
        System.out.println("Dados de Pessoa Juridica recuperados.");
    }
  }
}
```

4.6. Classe Principal com Menu (CadastroPOO.java) *(Nota: O código completo do menu foi inserido no repositório, apresentando opções de incluir, alterar, excluir, exibir, salvar e recuperar dados).*

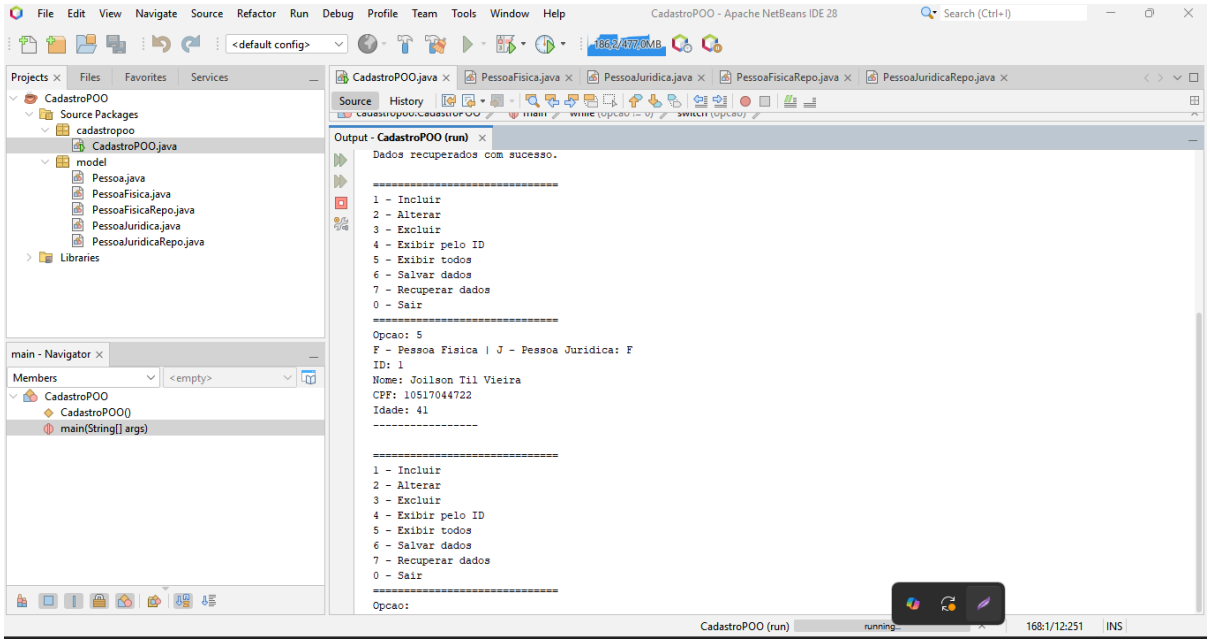
5. Resultados da Execução

5.1. Teste de Funcionalidade Básica (1º Procedimento) Abaixo, o print demonstrando a execução do teste hardcoded, onde dados foram inseridos, persistidos em arquivo e recuperados automaticamente.



5.2. Teste do Sistema com Menu (2º Procedimento)

Abaixo, o print demonstrando a interface em modo texto, com a inclusão de uma nova Pessoa Física e a exibição dos dados cadastrados.



6. Análise e Conclusão

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

Elementos estáticos (modificador `static`) em Java pertencem à classe em si, e não a uma instância específica (objeto) dessa classe. Isso significa que eles podem ser acessados sem que seja necessário criar um objeto `new Classe()`. O método `main` adota o modificador `static` porque ele é o ponto de entrada da aplicação. Quando a Máquina Virtual Java (JVM) inicia o programa, nenhum objeto foi criado ainda. Ao ser estático, a JVM consegue invocar `main` diretamente da classe para começar a execução.

b) Para que serve a classe Scanner?

A classe `Scanner`, pertencente ao pacote `java.util`, é utilizada para facilitar a entrada de dados. Ela é capaz de analisar (parse) strings e tipos primitivos usando expressões regulares. No contexto deste trabalho, ela foi fundamental para capturar os dados digitados pelo usuário no teclado (`System.in`), permitindo a leitura de inteiros (para o menu e IDs) e textos (para nomes e documentos) de forma interativa.

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

A utilização das classes de repositório (`PessoaFisicaRepo` e `PessoaJuridicaRepo`) teve um impacto muito positivo na organização, pois aplicou o princípio da Separação de Responsabilidades. Retiramos a responsabilidade de gerenciar a lista e lidar com arquivos das classes de entidade (`Pessoa`) e da classe principal (`Main`).

- **Entidades:** Ficaram apenas com os dados.
- **Repositórios:** Ficaram com a lógica de armazenamento (CRUD) e persistência.
- **Main:** Ficou apenas com a interface com o usuário (Menu). Isso torna o código mais limpo, fácil de manter e mais modular.