UNIVERSIDADE ESTÁCIO DE SÁ

TECNOLOGIA EMDESENVOLVIMENTO FULL STACK

RPG0014- INICIANDO O CAMINHO PELO JAVA

CARLOS ALEXANDRE PAULINO DE OLIVEIRA

RELATÓRIO DA MISSÃO PRÁTICA

EUSÉBIO 2024

1. INTRODUÇÃO

Este relatório tem como objetivo documentar e analisar a prática realizada no desenvolvimento de sistemas em Java, focando na implementação de um cadastro de clientes em modo texto com persistência em arquivos. O exercício propõe a aplicação de conceitos fundamentais, como herança, polimorfismo, persistência de objetos em arquivos binários e tratamento de exceções na linguagem Java.

A atividade visa consolidar o aprendizado teórico através da execução prática de um projeto. O desenvolvimento do sistema cadastral em Java envolve a criação de entidades, a implementação de gerenciadores, o uso da interface Serializable, além da execução de testes e ajustes necessários para garantir o correto funcionamento do sistema.

Espera-se, ao final da prática, não apenas a entrega de um sistema funcional, mas também a compreensão e aplicação de boas práticas de programação orientada a objetos, organização do código e persistência de dados. A prática, realizada de forma individual, foca no fortalecimento das habilidades fundamentais para o desenvolvimento em Java, especialmente na persistência de dados em arquivos binários.

Este relatório apresentará os conceitos utilizados, os procedimentos executados, os resultados obtidos e as análises sobre o desenvolvimento do sistema, culminando em uma conclusão que abordará as vantagens e desvantagens das técnicas aplicadas, além de reflexões sobre os conceitos assimilados durante a prática.

2. PROGRAMAÇÃO ORIENTADA A OBJETOS

A Programação Orientada a Objetos (POO) é um paradigma que se baseia na criação de objetos que encapsulam dados e métodos para operar sobre esses dados. Essa abordagem visa modelar o mundo real de forma mais intuitiva, facilitando a compreensão e manutenção do código (VERSOLATTO, 2023, p. 3).

2.1 Pilares da Programação Orientada a Objetos

De acordo com Silva (2023) e Versolatto (2023), os pilares essenciais da POO incluem abstração, encapsulamento, herança e polimorfismo.

- **Abstração:** Refere-se à simplificação de sistemas complexos, destacando apenas os aspectos essenciais através da criação de classes e objetos.
- **Encapsulamento:** Agrupa dados e métodos relacionados em uma única unidade chamada classe, protegendo os detalhes internos e controlando o acesso aos dados.
- **Herança:** Permite que uma classe derive características de outra, promovendo a reutilização de código e facilitando a criação de hierarquias de classes.
- **Polimorfismo:** Refere-se à habilidade de um objeto assumir diferentes formas, permitindo que objetos de tipos diferentes sejam tratados de maneira uniforme.

3. LINGUAGEM DE PROGRAMAÇÃO JAVA

Java, criada por James Gosling em 1995 na Sun Microsystems, é uma linguagem de programação orientada a objetos. Ela oferece diversos benefícios, como modularidade, reutilização e facilidade de manutenção (PALMEIRA, 2012; LEMOS, 2009).

- **Classe:** A unidade fundamental de organização do código em Java,

responsável por definir a estrutura e comportamento dos objetos.

- **Objetos:** Instâncias de classes que encapsulam dados e comportamentos, facilitando a modelagem de entidades reais.
- **Herança:** Java permite que classes herdem atributos e métodos de outras, promovendo a reutilização de código e a criação de hierarquias.
- **Polimorfismo:** Java suporta o polimorfismo por meio de interfaces, classes abstratas e sobrecarga de métodos, permitindo que objetos de diferentes tipos sejam manipulados de forma uniforme.
- **Encapsulamento:** Java utiliza modificadores de acesso como public, private e protected para controlar a visibilidade dos atributos e métodos de uma classe.

Em resumo, Java incorpora profundamente os princípios da POO, tornando-a uma linguagem ideal para o desenvolvimento de sistemas escaláveis e modulares. O exercício prático ilustra a aplicação desses conceitos em um ambiente de programação real.

4. METODOLOGIA

A metodologia adotada iniciou-se com um estudo teórico dos conceitos-chave, como herança, polimorfismo, persistência de objetos em arquivos binários e uso da interface Serializable. Em seguida, foi realizada uma análise detalhada do roteiro da prática, identificando etapas e requisitos críticos.

A configuração do ambiente de desenvolvimento incluiu a instalação do JDK 17.0.6 e da IDE NetBeans, escolhidos por sua compatibilidade e facilidade de uso. O desenvolvimento foi conduzido de forma modular, iniciando com a definição das entidades e seus relacionamentos, seguido de testes unitários para garantir o correto funcionamento do sistema.

Após a implementação, foram realizados testes individuais para verificar a persistência e recuperação de dados, além da robustez do

tratamento de exceções. O projeto foi armazenado em um repositório Git, e a documentação gerada foi enviada para avaliação.

5. RESULTADOS E ANÁLISES

O projeto foi estruturado em pacotes, com destaque para o pacote model, que contém as entidades e gerenciadores. A abordagem modular adotada facilita a manutenção e organização do código. No pacote model, as classes **Pessoa**, **PessoaFisica** e **PessoaJuridica** foram implementadas, sendo **Pessoa** a classe base com campos comuns, enquanto as derivadas incluem campos específicos.

Todas as classes implementam a interface Serializable, permitindo a persistência segura em arquivos binários. A seguir, são apresentados trechos de código que ilustram essa implementação.

Código 1: Classe Pessoa

```
package model;

import java.io.Serializable;

public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    private int id;
    private String nome;


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


    public int getId() {
        return id;
    }


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

```
    public void setNome(String nome) {
        this.nome = nome;
    }

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


    public void exibir(){

        System.out.println("ID: " + this.id);
        System.out.println("Nome: " + this.nome);

    }
}
```

Código2: ClassePessoaFisica

```
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    private String cpf;
    private int idade;

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

Código3: ClassePessoaJuridica

```
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    private String cnpj;

    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() {
        System.out.println("CNPJ: " + this.cnpj);

    }

}
```

Os gerenciadores, como Pessoa Fisica Repo e Pessoa Juridica Repo, são responsáveis por manipular as entidades e realizar operações como inserção, alteração, exclusão, recuperação e persistência. Apresença dos métodos persistire recuperar, que lançam exceções, garante um controle robusto das operações de armazenamento e recuperação de dados. Como apresentados a seguir:

Código 4: Classe Pessoa Física Repo

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {
    private final List<PessoaFisica> listaPessoasFisicas = new ArrayList<>();

    public void inserir(PessoaFisica pessoaFisica) {
        listaPessoasFisicas.add(pessoaFisica);
    }

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

    public void excluir(int id) {
        for (int i = 0; i < listaPessoasFisicas.size(); i++) {
            if (listaPessoasFisicas.get(i).getId() == id) {
                listaPessoasFisicas.remove(i);
                return;
            }
        }
    }

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

```
    public List<PessoaFisica> obterTodos() {
        return listaPessoasFisicas;
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(listaPessoasFisicas);
        } catch (IOException e) {
            throw e;
        }
    }

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasFisicas.clear();
            List<PessoaFisica> listaRecuperada = (List<PessoaFisica>)
inputStream.readObject();
            listaPessoasFisicas.addAll(listaRecuperada);
        } catch (IOException | ClassNotFoundException e) {
            throw e;
        }
    }
}
```

Código 4: Classe Pessoa Jurídica Repo

```
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaRepo {
    private final List<PessoaJuridica> listaPessoasJuridicas = new ArrayList<>();

    public void inserir(PessoaJuridica pessoaJuridica) {
        listaPessoasJuridicas.add(pessoaJuridica);
    }

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

```
    public void excluir(int id) {
        for (int i = 0; i < listaPessoasJuridicas.size(); i++) {
            if (listaPessoasJuridicas.get(i).getId() == id) {
                listaPessoasJuridicas.remove(i);
                return;
            }
        }
    }

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

    public List<PessoaJuridica> obterTodos() {
        return listaPessoasJuridicas;
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(listaPessoasJuridicas);
        } catch (IOException e) {
            throw e;
        }
    }

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasJuridicas.clear();
            List<PessoaJuridica> listaRecuperada = (List<PessoaJuridica>)
inputStream.readObject();
            listaPessoasJuridicas.addAll(listaRecuperada);
        } catch (IOException | ClassNotFoundException e) {
            throw e;
        }
    }
}
```

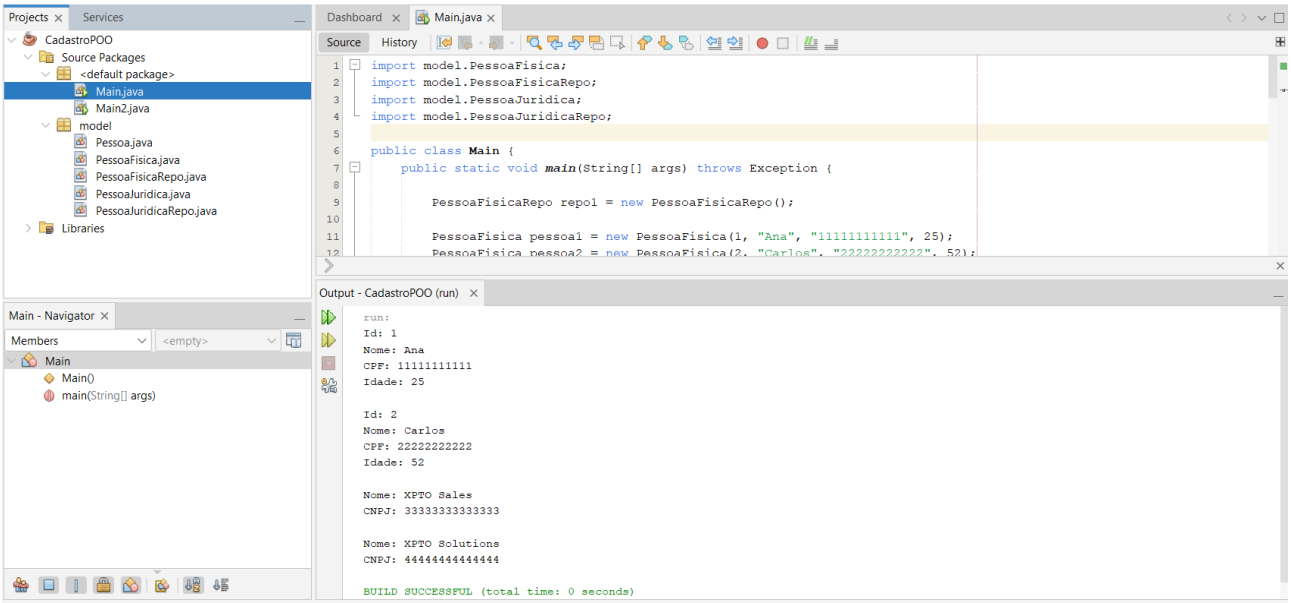
O método main da classe principal foi implementado para instanciar repositórios, adicionar dados, persistir e recuperar esses dados (Código6). Os resultados esperados incluem uma execução organizada e a correta persistência e recuperação dos dados (Figura1).

Código 6: Classe Main

```
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;

public class Main {
    public static void main(String[] args) throws Exception {

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        PessoaFisica pessoa1 = new PessoaFisica(1, "Ana", "11111111111", 25);
        PessoaFisica pessoa2 = new PessoaFisica(2, "Carlos", "22222222222", 52);
        repo1.inserir(pessoa1);
        repo1.inserir(pessoa2);

        repo1.persistir("dados-pf1");

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        repo2.recuperar("dados-pf1");

        for (PessoaFisica pessoa : repo2.obterTodos()) {
            System.out.println("Id: " + pessoa.getId());
            System.out.println("Nome: " + pessoa.getNome());
            System.out.println("CPF: " + pessoa.getCpf());
            System.out.println("Idade: " + pessoa.getIdade());
            System.out.println();
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        PessoaJuridica pessoaJuridica1 = new PessoaJuridica(1, "XPTO Sales",
"33333333333333");
        PessoaJuridica pessoaJuridica2 = new PessoaJuridica(2, "XPTO Solutions",
"44444444444444");
        repo3.inserir(pessoaJuridica1);
        repo3.inserir(pessoaJuridica2);

        repo3.persistir("dados-pj1");

        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

        repo4.recuperar("dados-pj1");
```

```
        for (PessoaJuridica pessoaJuridica : repo4.obterTodos()) {
            System.out.println("Nome: " + pessoaJuridica.getNome());
            System.out.println("CNPJ: " + pessoaJuridica.getCnpj());
            System.out.println();
        }
    }
}
```

Figura 1- Execução da classe Main



A segunda parte do desenvolvimento, apresentada no método main2, introduz uma interação textual com o usuário. As opções incluem incluir, alterar, excluir, buscar e exibir dados, além de persistir e recuperar informações. As exceções são tratadas de maneira adequada para garantir a robustez do sistema, conforme solicitado no segundo procedimento da atividade prática (Código 7 e Figura 2).

Código 7: Classe Main2

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;
```

```
public class Main2 {

    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String opcao = "";
        PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
        PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();

        while (!"0".equals(opcao)) {
            System.out.println("====================================");
            System.out.println("1 - Incluir Pessoa");
            System.out.println("2 - Alterar Pessoa");
            System.out.println("3 - Excluir Pessoa");
            System.out.println("4 - Buscar pelo Id");
            System.out.println("5 - Exibir Todos");
            System.out.println("6 - Persistir Dados");
            System.out.println("7 - Recuperar Dados");
            System.out.println("0 - Finalizar Programa");
            System.out.println("====================================");

            try {
                opcao = reader.readLine();

                switch (opcao) {
                    case "1":
                        System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
                        String tipoPessoa = reader.readLine();
                        switch (tipoPessoa) {
                            case "F":
                                PessoaFisica pf = lerDadosPessoaFisica(reader);
                                repoFisica.inserir(pf);
                                System.out.println("Pessoa Física incluída com sucesso.");
                                break;
                            case "J":
                                PessoaJuridica pj = lerDadosPessoaJuridica(reader);
                                repoJuridica.inserir(pj);
                                System.out.println("Pessoa Jurídica incluída com sucesso.");
                                break;
                            default:
                                System.out.println("Tipo inválido.");
                        }
                        break;

                    case "2":
                        System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
                        String tipoPessoaAlterar = reader.readLine();
                        switch (tipoPessoaAlterar) {
                            case "F":
                                alterarPessoa(repoFisica, reader);
                                System.out.println("Pessoa Física alterada com sucesso.");
                                break;
                            case "J":
```

```
                        alterarPessoa(repoJuridica, reader);
                        System.out.println("Pessoa Jurídica alterada com sucesso.");
                        break;
                    default:
                        System.out.println("Tipo inválido.");
                }
                break;

            case "3":
                System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
                String tipoPessoaExcluir = reader.readLine();
                switch (tipoPessoaExcluir) {
                    case "F":
                        excluirPessoa(repoFisica, reader);
                        System.out.println("Pessoa Física excluída com sucesso.");
                        break;
                    case "J":
                        excluirPessoa(repoJuridica, reader);
                        System.out.println("Pessoa Jurídica excluída com sucesso.");
                        break;
                    default:
                        System.out.println("Tipo inválido.");
                }
                break;

            case "4":
                System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
                String tipoPessoaBuscar = reader.readLine();
                switch (tipoPessoaBuscar) {
                    case "F":
                        buscarPessoa(repoFisica, reader);
                        System.out.println("Pessoa Física encontrada com sucesso.");
                        break;
                    case "J":
                        buscarPessoa(repoJuridica, reader);
                        System.out.println("Pessoa Jurídica encontrada com sucesso.");
                        break;
                    default:
                        System.out.println("Tipo inválido.");
                }
                break;

            case "5":
                System.out.println("F - Pessoa Física | J - Pessoa Jurídica");
                String tipoPessoaExibirTodos = reader.readLine();
                switch (tipoPessoaExibirTodos) {
                    case "F":
                        exibirTodasPessoas(repoFisica);
                        System.out.println("Listagem de todas as Pessoas Físicas
cadastradas.");
                        break;
                    case "J":
```

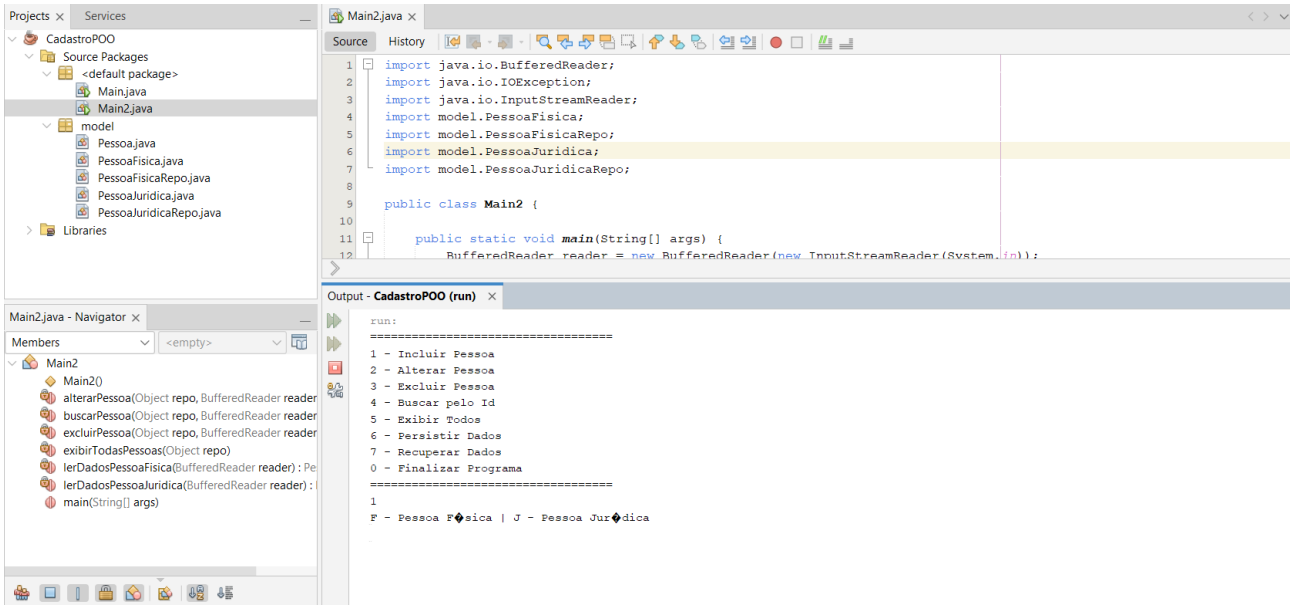
```
                        exibirTodasPessoas(repoJuridica);
                        System.out.println("Listagem de todas as Pessoas Jurídicas
cadastradas.");
                        break;
                    default:
                        System.out.println("Tipo inválido.");
                }
                break;

            case "6":
                System.out.print("Qual o nome dos arquivos? ");
                String arquivoP = reader.readLine();
                try {
                    repoFisica.persistir(arquivoP + ".fisica.bin");
                    repoJuridica.persistir(arquivoP + ".juridica.bin");
                    System.out.println("Dados salvos com sucesso.");
                } catch (IOException e) {
                    System.out.println("Erro ao salvar os dados: " + e.getMessage());
                }
                break;

            case "7":
                System.out.print("Qual o nome dos arquivos? ");
                String arquivoR = reader.readLine();
                try {
                    repoFisica.recuperar(arquivoR + ".fisica.bin");
                    repoJuridica.recuperar(arquivoR + ".juridica.bin");
                    System.out.println("Dados recuperados com sucesso.");
                } catch (IOException | ClassNotFoundException e) {
                    System.out.println("Erro ao recuperar os dados: " + e.getMessage());
                }
                break;

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

            default:
                System.out.println("Opção inválida!");
                break;
            }
        } catch (IOException e) {
            System.out.println("Erro de entrada/saída: " + e.getMessage());
        }
    }
}

    private static PessoaFisica lerDadosPessoaFisica(BufferedReader reader) throws
IOException {
        System.out.println("Digite o id da pessoa: ");
        int id = Integer.parseInt(reader.readLine());
        System.out.println("Insira os dados...");
```

```
        System.out.print("Nome: ");
        String nome = reader.readLine();
        System.out.print("CPF: ");
        String cpf = reader.readLine();
        System.out.print("Idade: ");
        int idade = Integer.parseInt(reader.readLine());
        return new PessoaFisica(id, nome, cpf, idade);
    }

    private static PessoaJuridica lerDadosPessoaJuridica(BufferedReader reader) throws
IOException {
        System.out.println("Digite o id da pessoa: ");
        int id = Integer.parseInt(reader.readLine());
        System.out.println("Insira os dados...");
        System.out.print("Nome: ");
        String nome = reader.readLine();
        System.out.print("CNPJ: ");
        String cnpj = reader.readLine();
        return new PessoaJuridica(id, nome, cnpj);
    }

    private static void alterarPessoa(Object repo, BufferedReader reader) throws
IOException {
        System.out.println("Digite o id da pessoa: ");
        int id = Integer.parseInt(reader.readLine());
        if (repo instanceof PessoaFisicaRepo) {
            PessoaFisica pf = ((PessoaFisicaRepo) repo).obter(id);
            if (pf != null) {
                System.out.println("Nome atual: " + pf.getNome());
                System.out.print("Novo nome: ");
                pf.setNome(reader.readLine());
                System.out.println("CPF atual: " + pf.getCpf());
                System.out.print("Novo CPF: ");
                pf.setCpf(reader.readLine());
                System.out.println("Idade atual: " + pf.getIdade());
                System.out.print("Nova idade: ");
                pf.setIdade(Integer.parseInt(reader.readLine()));
                ((PessoaFisicaRepo) repo).alterar(pf);
            } else {
                System.out.println("Pessoa Física não encontrada.");
            }
        } else if (repo instanceof PessoaJuridicaRepo) {
            PessoaJuridica pj = ((PessoaJuridicaRepo) repo).obter(id);
            if (pj != null) {
                System.out.println("Nome atual: " + pj.getNome());
                System.out.print("Novo nome: ");
                pj.setNome(reader.readLine());
                System.out.println("CNPJ atual: " + pj.getCnpj());
                System.out.print("Novo CNPJ: ");
                pj.setCnpj(reader.readLine());
                ((PessoaJuridicaRepo) repo).alterar(pj);
            } else {
```

```
            System.out.println("Pessoa Jurídica não encontrada.");
        }
    }
}

private static void excluirPessoa(Object repo, BufferedReader reader) throws
IOException {
    System.out.print("Digite o Id do usuário: ");
    int id = Integer.parseInt(reader.readLine());
    if (repo instanceof PessoaFisicaRepo) {
        ((PessoaFisicaRepo) repo).excluir(id);
    } else if (repo instanceof PessoaJuridicaRepo) {
        ((PessoaJuridicaRepo) repo).excluir(id);
    }
}

private static void buscarPessoa(Object repo, BufferedReader reader) throws
IOException {
    System.out.print("Digite o id da pessoa: ");
    int id = Integer.parseInt(reader.readLine());
    if (repo instanceof PessoaFisicaRepo) {
        PessoaFisica pf = ((PessoaFisicaRepo) repo).obter(id);
        if (pf != null) {
            System.out.println("Id: " + pf.getId());
            System.out.println("Nome: " + pf.getNome());
            System.out.println("CPF: " + pf.getCpf());
            System.out.println("Idade: " + pf.getIdade());
        } else {
            System.out.println("Pessoa Física não encontrada.");
        }
    } else if (repo instanceof PessoaJuridicaRepo) {
        PessoaJuridica pj = ((PessoaJuridicaRepo) repo).obter(id);
        if (pj != null) {
            System.out.println("Id: " + pj.getId());
            System.out.println("Nome: " + pj.getNome());
            System.out.println("CNPJ: " + pj.getCnpj());
        } else {
            System.out.println("Pessoa Jurídica não encontrada.");
        }
    }
}

private static void exibirTodasPessoas(Object repo) {
    if (repo instanceof PessoaFisicaRepo) {
        for (PessoaFisica pf : ((PessoaFisicaRepo) repo).obterTodos()) {
            System.out.println("Id: " + pf.getId());
            System.out.println("Nome: " + pf.getNome());
            System.out.println("CPF: " + pf.getCpf());
            System.out.println("Idade: " + pf.getIdade());
        }
    } else if (repo instanceof PessoaJuridicaRepo) {
        for (PessoaJuridica pj : ((PessoaJuridicaRepo) repo).obterTodos()) {
```

```
            System.out.println("Id: " + pj.getId());
            System.out.println("Nome: " + pj.getNome());
            System.out.println("CNPJ: " + pj.getCnpj());
        }
    }
}
}
```

Figura 1- Execução da classe Main 2



Durante a execução do código, os resultados obtidos foram satisfatórios. Os repositórios para pessoas físicas e jurídicas foram criados, com dados adicionados, persistidos e recuperados com sucesso. A organização estruturada do código, junto ao uso eficaz das funcionalidades do NetBeans, facilitou a implementação bem-sucedida do sistema de cadastro.

6 DISCUSSÕES

A prática proporcionou uma compreensão aprofundada dos conceitos teóricos discutidos, permitindo a aplicação prática desses conhecimentos na construção do sistema cadastral. As vantagens da programação orientada a objetos, aliadas ao uso adequado de persistência em arquivos binários, resultaram em um código funcional.

6.1 Vantagens e desvantagens da herança

A herança e o polimorfismo foram fundamentais para definir entidades dentro da programação orientada a objetos. A classe base Pessoa exemplificou esses conceitos, com PessoaFisica e PessoaJuridica herdando

características da classe mãe, possibilitando a extensão e especialização de comportamentos.

6.1.1 Vantagens
A herança permite que as subclasses herdem atributos e métodos da classe base, promovendo a reutilização do código. Além disso, cria uma estrutura hierárquica que facilita a organização e compreensão do sistema, além de permitir o polimorfismo, onde objetos de classes diferentes podem ser tratados uniformemente.

6.1.2 Desvantagens
O uso excessivo de herança pode resultar em um acoplamento elevado entre classes, tornando o sistema mais complexo e difícil de manter. Além disso, alterações na classe mãe podem afetar as classes filhas, diminuindo a flexibilidade do sistema.

6.2 Importância da interface Serializable
A persistência de objetos em arquivos binários é essencial para a manipulação e armazenamento de dados. A interface Serializable foi utilizada em todas as classes do projeto, permitindo que os objetos fossem convertidos em sequências de bytes para armazenamento e recuperação posteriores. Essa interface marca as classes que podem ser serializadas, sendo crucial para operações de persistência.

6.3 Paradigma funcional e a API Stream no Java
A API Stream em Java incorpora o paradigma funcional nas operações de processamento de dados. Com o uso de expressões lambda, a API Stream permite operações poderosas e concisas em coleções, promovendo um código mais limpo e legível. As expressões lambda possibilitam a definição de funções anônimas de forma compacta, e operações como filter e map aplicam funções a elementos da coleção.

6.4 Padrão de desenvolvimento em persistência de dados
Ao utilizar Java, um padrão comum na persistência de dados em arquivos é a interface Serializable, já mencionada e utilizada nesta atividade prática. Essa abordagem é especialmente útil para garantir a persistência eficiente e estruturada de objetos, mantendo a integridade dos dados.

6.5 Elementos estáticos e o método main
Elementos estáticos em Java estão associados à classe, e não à instância de um objeto, significando que pertencem à classe como um todo. Métodos e variáveis estáticos, declarados com a palavra-chave static, podem ser acessados sem a necessidade de instanciar a classe. O método main em Java é o ponto de entrada para um programa e, por isso, utiliza o modificador static para indicar que pertence à classe e pode ser chamado diretamente pela JVM, sem criar um objeto da classe.

6.6 Função da classe Scanner
A classe Scanner em Java é usada para receber entradas do usuário pelo

console. Ela oferece métodos para ler diferentes tipos de dados, como inteiros, doubles e strings, facilitando a interação durante a execução do programa. No método main, utilizou-se a classe BufferedReader para obter entradas do usuário de maneira semelhante, permitindo uma interação dinâmica.

6.7 Impacto das classes de repositório na organização do código

A introdução de classes de repositório melhorou significativamente a organização do código. Essas classes atuam como intermediárias entre o programa e os dados persistidos, encapsulando a lógica de armazenamento e recuperação, resultando em um código mais modular e coeso. O uso dessas classes segue o princípio da responsabilidade única, tornando o código mais claro e alinhado com as boas práticas de desenvolvimento. Exemplos como PessoaFisicaRepo e PessoaJuridicaRepo demonstram esse impacto positivo, gerenciando a lógica de entidades e persistência de arquivos, enquanto o código principal (Main2) invoca métodos específicos para operações de CRUD.

7 CONSIDERAÇÕES FINAIS

O desenvolvimento desta atividade prática ofereceu uma imersão nos princípios fundamentais da Programação Orientada a Objetos (POO) e na manipulação de persistência de dados em Java. A implementação do sistema cadastral, focando nas entidades Pessoa, PessoaFisica e PessoaJuridica, proporcionou uma compreensão profunda sobre herança, polimorfismo e persistência de objetos em arquivos binários.

Ao discutir a herança, foram exploradas suas vantagens, como a reutilização de código e a criação de uma estrutura hierárquica. Contudo, também foram identificados desafios, como a complexidade crescente com hierarquias mais profundas. A interface Serializable foi crucial na persistência de objetos, permitindo a transformação eficiente de objetos em bytes por meio de ObjectOutputStream e ObjectInputStream, destacando a importância da segurança na transmissão e armazenamento.

O padrão de desenvolvimento adotado, utilizando a interface Serializable e classes de repositório, possibilitou uma organização eficaz do código. A modularização das operações de CRUD nas classes PessoaFisicaRepo e PessoaJuridicaRepo contribuiu para um código mais coeso e de fácil manutenção.

Dessa forma, concluímos que a atividade prática não apenas atingiu os objetivos propostos, mas também forneceu uma base sólida para compreender os princípios da POO em Java e as estratégias de persistência de dados, consolidando conhecimentos essenciais na programação orientada a objetos.