	**Campus**: Polo Cohama **Curso**: Desenvolvimento FullStack **Turma**: 9001 **Disciplina**: Iniciando o Caminho Pelo Java **Nome**: João Victor Sá de Araújo

Relatório de Prática: Criação das Entidades e Sistema de Persistência

2. Objetivo da Prática

O objetivo da prática é implementar e testar a manipulação de repositórios de dados utilizando conceitos de programação orientada a objetos, como herança, polimorfismo e serialização. Isso inclui a criação de repositórios para gerenciar entidades (`PessoaFisica` e `PessoaJuridica`), persistência de dados em arquivos, recuperação das informações e interação com listas dinâmicas para garantir o funcionamento correto das operações CRUD.

3. Códigos

CadastroPOO\src\cadastropoo\model\Pessoa.java/*

```
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package cadastropoo.model;

import java.io.Serializable;

/**
 *
 * @author joao_
 */
public class Pessoa implements Serializable{

      public int id;
      public String nome;
      private static final long serialVersionUID = 1L;
    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 Pessoa(int id, String nome) {
        this.id = id;
        this.nome = nome;
    }

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


}
```

CadastroPOO\src\cadastropoo\model\PessoaFisica.java:

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package cadastropoo.model;

import java.io.Serializable;

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

    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 + ", Idade: " + idade);
    }

}
```

CadastroPOO\src\cadastropoo\model\PessoaFisicaRepo.java:

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package cadastropoo.model;

/**
 *
 * @author joao_
 */
import java.io.*;
import java.util.ArrayList;
import java.util.List;
```

```java
// Classe PessoaFisicaRepo
public class PessoaFisicaRepo {
    private final List<PessoaFisica> pessoasFisicas = new
ArrayList<>();

    // Método inserir
    public void inserir(PessoaFisica pessoa) {
        pessoasFisicas.add(pessoa);
    }

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

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

    // Método obter
    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoa : pessoasFisicas) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null; // Retorna null caso não encontre a pessoa
    }

    // Método obterTodos
    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(pessoasFisicas);
    }

    // Método persistir
    public void persistir(String nomeArquivo) throws IOException {
```

```
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoasFisicas);
        }
    }

    // Método recuperar
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            List<PessoaFisica> recuperadas = (List<PessoaFisica>)
ois.readObject();
            pessoasFisicas.clear();
            pessoasFisicas.addAll(recuperadas);
        }
    }
}
```

CadastroPOO\src\cadastropoo\model\PessoaJuridica.java:

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package cadastropoo.model;

import java.io.Serializable;

/**
 *
 * @author joao_
 */
public class PessoaJuridica extends Pessoa implements Serializable   {
    private String cnpj;
    private static final long serialVersionUID = 1L;

    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);
    }

}
```

CadastroPOO\src\cadastropoo\model\PessoaJuridicaRepo.java:

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package cadastropoo.model;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

/**
 *
```

```java
 * @author joao_
 */
public class PessoaJuridicaRepo {
    private final List<PessoaJuridica> pessoasJuridicas = new
ArrayList<>();

    // Método inserir
    public void inserir(PessoaJuridica pessoa) {
        pessoasJuridicas.add(pessoa);
    }

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

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

    // Método obter
    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null; // Retorna null caso não encontre a pessoa
    }

    // Método obterTodos
    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(pessoasJuridicas);
    }

    // Método persistir
    public void persistir(String nomeArquivo) throws IOException {
```

```
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoasJuridicas);
        }
    }

    // Método recuperar
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            List<PessoaJuridica> recuperadas = (List<PessoaJuridica>)
ois.readObject();
            pessoasJuridicas.clear();
            pessoasJuridicas.addAll(recuperadas);
        }
    }
}
```

4. Resultados Obtidos

```
Dados de pessoas fisicas salvos no arquivo: pessoasFisicas.dat
Dados recuperados do arquivo: pessoasFisicas.dat
Pessoas Fisicas recuperadas:
ID: 1 , Nome: Ana
CPF: 123.456.789-00, Idade: 25
ID: 2 , Nome: Carlos
CPF: 987.654.321-00, Idade: 52
Dados de pessoas juridicas salvos no arquivo: pessoasJuridicas.dat
Dados recuperados do arquivo: pessoasJuridicas.dat
Pessoas Juridicas recuperadas:
ID: 1 , Nome: XPTO Sales
CNPJ: 12.345.678/0001-90
ID: 2 , Nome: XPTO Solutions
CNPJ: 98.765.432/0001-10
```

5. Análise e Conclusão

5.1. Vantagens e Desvantagens do Uso de Herança

- Vantagens:
    - Reuso de Código: Herança permite que classes derivadas reutilizem métodos e atributos definidos em classes base, promovendo economia de esforço e manutenção.

    - Organização Hierárquica: Facilita a modelagem de relações "é-um" (exemplo: `PessoaFisica` é uma `Pessoa`), tornando o design mais intuitivo e lógico.
    - Polimorfismo: Classes derivadas podem sobrescrever métodos da classe base para oferecer comportamentos específicos, aumentando a flexibilidade do código.
- Desvantagens:
    - Fragilidade da Base: Alterações na classe base podem impactar de forma indesejada todas as classes derivadas, dificultando a manutenção.
    - Aumento da Complexidade: Relações hierárquicas profundas podem tornar o código difícil de entender e depurar.
    - Adoção Prematura: Nem todas as relações "parecem ser" entre classes justificam a herança. O uso inadequado pode resultar em designs rígidos.

5.2. Por que a Interface `Serializable` é Necessária na Persistência em Arquivos Binários?

A interface `Serializable` é necessária porque ela instrui o mecanismo de serialização do Java a transformar objetos em uma sequência de bytes, que pode ser armazenada em um arquivo ou transmitida por rede. Sem implementá-la, o Java não sabe como processar os atributos do objeto para conversão. Além disso:

- Garante que os objetos possam ser reconstruídos (desserializados) na memória.
- Permite preservar o estado do objeto, incluindo os dados de seus atributos, durante a persistência.

5.3. Como o Paradigma Funcional é Utilizado pela API `Stream` no Java?

O paradigma funcional na API `Stream` do Java se manifesta no uso de funções como elementos de primeira classe, permitindo operações de alto nível sobre coleções de dados. Exemplos incluem:

- Operações Declarativas: Métodos como `filter`, `map` e `reduce` permitem especificar o "o quê" deve ser feito em vez do "como".
- Imutabilidade: Streams trabalham em pipelines que não modificam os dados originais.
- Funções Lambda: Funções anônimas são usadas para simplificar a manipulação dos dados.
- Concorrência Simplificada: Operações paralelas podem ser realizadas com `parallelStream`, otimizando o processamento.

5.4. Padrão de Desenvolvimento Adotado na Persistência de Dados em Arquivos no Java

Ao trabalhar com persistência de dados em arquivos no Java, o padrão mais comumente adotado é o Data Access Object (DAO), que separa a lógica de acesso aos dados da lógica de negócios. Esse padrão:

- Facilita a manutenção e escalabilidade do sistema.
- Centraliza as operações de leitura e gravação de arquivos em classes específicas.
- Garante que a lógica de persistência seja desacoplada do restante da aplicação.

Além disso, ao lidar com arquivos binários, o uso de APIs como `ObjectOutputStream` e `ObjectInputStream` representa um padrão específico para serialização, oferecendo um mecanismo padronizado e seguro para a persistência.