**Universidade Estácio de Sá**

**Polo Estácio EAD - Batatais / SP**

**Curso:** Desenvolvimento Full Stack  
**Disciplina:** Iniciando o Caminho pelo Java  
**Missão Prática - Nível 1  
Turma:** RPG0014  
**Semestre Letivo:** 2024.3  
 **Integrante:** Daiana Maira de Oliveira Lascala

**Título**

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

**Objetivo da Prática**

* Utilizar herança e polimorfismo na definição de entidades: Criar classes que herdem atributos e comportamentos, utilizando polimorfismo para exibir diferentes comportamentos em classes herdadas.
* Utilizar persistência de objetos em arquivos binários: Implementar a funcionalidade de salvar e recuperar dados de entidades para arquivos binários.
* Implementar uma interface cadastral em modo texto: Criar um cadastro de clientes simples utilizando a linha de comando (modo texto).
* Utilizar o controle de exceções da plataforma Java: Lidar com possíveis erros durante a execução, como ao salvar ou carregar dados.
* No final do projeto, implementar um sistema cadastral em Java, utilizando os recursos de Programação Orientada a Objetos (POO) e persistência em arquivos binários.

**Códigos Implementados**

Os códigos desenvolvidos para esta prática foram divididos nas seguintes partes:

**1. Classes de Entidades**

* Pessoa.java: Classe base com atributos como id e nome, e métodos como exibir, além de getters e setters.
* PessoaFisica.java: Classe herdada de Pessoa, com atributos adicionais como cpf e idade.
* PessoaJuridica.java: Classe herdada de Pessoa, com atributo adicional cnpj.

**2. Repositórios**

* PessoaFisicaRepo.java: Classe que gerencia um ArrayList de PessoaFisica, com métodos de inserir, alterar, excluir, persistir e recuperar.
* PessoaJuridicaRepo.java: Classe semelhante para gerenciar entidades de PessoaJuridica.

**3. Interface Cadastral**

* O programa foi testado por meio da classe Principal.java, onde foi instanciado repositórios e feita a persistência e recuperação de dados de clientes, tanto físicos quanto jurídicos.

**Resultados**

**Pessoa.java**

package model;

import java.io.Serializable;

public class Pessoa implements Serializable {

    private int id;

    private String nome;

    // Construtor padrão

    public Pessoa() {}

    // Construtor completo

    public Pessoa(int id, String nome) {

        this.id = id;

        this.nome = nome;

    }

    // Getters e Setters

    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;

    }

    // Exibir

    public void exibir() {

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

    }

}

**PessoaFisica.java**

package model;

public class PessoaFisica extends Pessoa {

    private String cpf;

    private int idade;

    // Construtor padrão

    public PessoaFisica() {}

    // Construtor completo

    public PessoaFisica(int id, String nome, String cpf, int idade) {

        super(id, nome); // Chama o construtor da superclasse (Pessoa)

        this.cpf = cpf;

        this.idade = idade;

    }

    // Getters e Setters

    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;

    }

    // Exibir

    @Override

    public void exibir() {

        super.exibir(); // Chama o método exibir da superclasse

        System.out.println("CPF: " + cpf + ", Idade: " + idade);

    }

}

**PessoaJuridica.java**

package model;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    // Construtor padrão

    public PessoaJuridica() {}

    // Construtor completo

    public PessoaJuridica(int id, String nome, String cnpj) {

        super(id, nome); // Chama o construtor da superclasse (Pessoa)

        this.cnpj = cnpj;

    }

    // Getters e Setters

    public String getCnpj() {

        return cnpj;

    }

    public void setCnpj(String cnpj) {

        this.cnpj = cnpj;

    }

    // Exibir

    @Override

    public void exibir() {

        super.exibir(); // Chama o método exibir da superclasse

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

    }

}

**PessoaFisicaRepo.java**

package model;

import java.io.\*;

import java.util.ArrayList;

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> listaPessoaFisica = new ArrayList<>();

    //Inserir

    public void inserir(PessoaFisica pessoaFisica) {

        listaPessoaFisica.add(pessoaFisica);

    }

    //Alterar

    public void alterar(PessoaFisica pessoaFisica) {

        PessoaFisica existente = obter(pessoaFisica.getId());

        if (existente != null) {

            existente.setNome(pessoaFisica.getNome());

            existente.setCpf(pessoaFisica.getCpf());

            existente.setIdade(pessoaFisica.getIdade());

        }

    }

    //Excluir

    public void excluir(int id) {

        PessoaFisica pessoaFisica = obter(id);

        if (pessoaFisica != null) {

            listaPessoaFisica.remove(pessoaFisica);

        }

    }

    //Obter (por id)

    public PessoaFisica obter(int id) {

        for (PessoaFisica pessoaFisica : listaPessoaFisica) {

            if (pessoaFisica.getId() == id) {

                return pessoaFisica;

            }

        }

        return null;

    }

    //ObterTodos

    public ArrayList<PessoaFisica> obterTodos() {

        return listaPessoaFisica;

    }

    //Persistir

    public void persistir(String nomeArquivo) throws IOException {

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {

            oos.writeObject(listaPessoaFisica);

        }

    }

    //Recuperar

    @SuppressWarnings("unchecked")

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {

            listaPessoaFisica = (ArrayList<PessoaFisica>) ois.readObject();

        }

    }

}

**PessoaJuridicaRepo.java**

package model;

import java.io.\*;

import java.util.ArrayList;

public class PessoaJuridicaRepo {

    private ArrayList<PessoaJuridica> listaPessoaJuridica = new ArrayList<>();

    // Inserir

    public void inserir(PessoaJuridica pessoaJuridica) {

        listaPessoaJuridica.add(pessoaJuridica);

    }

    // Alterar

    public void alterar(PessoaJuridica pessoaJuridica) {

        PessoaJuridica existente = obter(pessoaJuridica.getId());

        if (existente != null) {

            existente.setNome(pessoaJuridica.getNome());

            existente.setCnpj(pessoaJuridica.getCnpj());

        }

    }

    // Excluir

    public void excluir(int id) {

        PessoaJuridica pessoaJuridica = obter(id);

        if (pessoaJuridica != null) {

            listaPessoaJuridica.remove(pessoaJuridica);

        }

    }

    // Obter (por id)

    public PessoaJuridica obter(int id) {

        for (PessoaJuridica pessoaJuridica : listaPessoaJuridica) {

            if (pessoaJuridica.getId() == id) {

                return pessoaJuridica;

            }

        }

        return null;

    }

    // ObterTodos

    public ArrayList<PessoaJuridica> obterTodos() {

        return listaPessoaJuridica;

    }

    // Persistir

    public void persistir(String nomeArquivo) throws IOException {

        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {

            oos.writeObject(listaPessoaJuridica);

        }

    }

    // Recuperar

    @SuppressWarnings("unchecked")

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {

        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {

            listaPessoaJuridica = (ArrayList<PessoaJuridica>) ois.readObject();

        }

    }

}

**Principal.java**

package model;

import java.io.IOException;

public class Principal {

    public static void main(String[] args) {

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        // Adicionar pessoas físicas

        PessoaFisica pessoa1 = new PessoaFisica(1, "Ana Silva", "123.456.789-00", 30);

        PessoaFisica pessoa2 = new PessoaFisica(2, "Maria Oliveira", "987.654.321-00", 25);

        repo1.inserir(pessoa1);

        repo1.inserir(pessoa2);

        // Persistência fornecendo o nome de arquivo

        String nomeArquivoFisica = "pessoasFisicas.dat";

        try {

            repo1.persistir(nomeArquivoFisica);

        } catch (IOException e) {

            System.out.println("Erro ao persistir os dados de pessoas físicas: " + e.getMessage());

        }

        // Repositório de pessoas físicas (repo2)

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        // Recuperação fornecendo o mesmo nome de arquivo

        try {

            repo2.recuperar(nomeArquivoFisica);

        } catch (IOException | ClassNotFoundException e) {

            System.out.println("Erro ao recuperar os dados de pessoas físicas: " + e.getMessage());

        }

        // Todas as pessoas físicas recuperadas

        for (PessoaFisica pf : repo2.obterTodos()) {

            System.out.println("ID: " + pf.getId() + ", Nome: " + pf.getNome() + ", CPF: " + pf.getCpf() + ", Idade: " + pf.getIdade());

        }

        // Repositório de pessoas jurídicas

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        PessoaJuridica empresa1 = new PessoaJuridica(1, "Jump Company", "00.111.222/0001-33");

        PessoaJuridica empresa2 = new PessoaJuridica(2, "IdeiaFood", "11.222.333/0001-44");

        repo3.inserir(empresa1);

        repo3.inserir(empresa2);

        // Persistência fornecendo o nome de arquivo

        String nomeArquivoJuridica = "pessoasJuridicas.dat";

        try {

            repo3.persistir(nomeArquivoJuridica);

        } catch (IOException e) {

            System.out.println("Erro ao persistir os dados de pessoas jurídicas: " + e.getMessage());

        }

        // Instanciar outro repositório de pessoas jurídicas

        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

        // Recuperação fornecendo o mesmo nome de arquivo

        try {

            repo4.recuperar(nomeArquivoJuridica);

        } catch (IOException | ClassNotFoundException e) {

            System.out.println("Erro ao recuperar os dados de pessoas jurídicas: " + e.getMessage());

        }

        // Todas as pessoas jurídicas recuperadas

        for (PessoaJuridica pj : repo4.obterTodos()) {

            System.out.println("ID: " + pj.getId() + ", Nome: " + pj.getNome() + ", CNPJ: " + pj.getCnpj());

        }

    }

}

**Análise e Conclusão**

**Quais as vantagens e desvantagens do uso de herança?**

Vantagens:

* Reutilização de código: A herança permite que classes filhas reutilizem métodos e atributos das classes pai, evitando duplicação de código.
* Organização: Facilita a estruturação do código ao agrupar comportamentos semelhantes em uma classe base.

Desvantagens:

* Acoplamento: A herança pode gerar um forte acoplamento entre classes, dificultando a manutenção e a evolução do código.
* Rigidez: A hierarquia de classes pode se tornar rígida, limitando a flexibilidade ao modificar comportamentos.

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

A interface Serializable é usada para indicar que um objeto pode ser convertido em uma sequência de bytes para que possa ser salvo em um arquivo e posteriormente reconstruído. Sem implementar essa interface, o Java não pode garantir que os objetos poderão ser gravados e recuperados corretamente de arquivos binários.

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

O paradigma funcional é utilizado pela API Stream do Java para realizar operações de processamento de dados de forma mais declarativa. Em vez de iterar coleções manualmente, podemos usar map(), filter(), e forEach() para aplicar transformações ou filtragens em fluxos de dados, de maneira funcional e concisa.

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

Na persistência de dados em arquivos, muitas vezes seguimos o padrão DAO (Data Access Object). Esse padrão encapsula as operações de acesso a dados, permitindo que a lógica de persistência seja separada da lógica de negócios. No projeto, isso foi aplicado através dos repositórios (PessoaFisicaRepo e PessoaJuridicaRepo), que gerenciam a persistência e recuperação dos objetos.

**Endereço do Repositório Git**

https://github.com/DaianaLascala/Missao-Nivel-1-Java-