Centro Universitário Estácio

Curso: Desenvolvimento Full Stack

Disciplina: Iniciando o Caminho Pelo Java

Semestre: 3°

Aluno: Jonnatha Walben Saldanha da Silva

Criação das Entidades e Sistema de Persistência

Objetivo da Prática:

O objetivo desta prática foi aplicar conceitos de programação orientada a objetos em Java para desenvolver um sistema de cadastro de clientes com persistência em arquivos binários. Os principais objetivos incluíram o uso de herança e polimorfismo na definição de entidades, a implementação de interfaces Serializable para persistência em arquivos, e a utilização da API Stream para manipulação de dados.

Códigos Solicitados:

Implementação das entidades

- **Pessoa.**

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

- **PessoaFisica.**

```
package model;

public class PessoaFisica extends Pessoa {
    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;
    }

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

- **PessoaJuridica**

```
package model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

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

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

    // Getters e setters
    public String getCnpj() {
        return cnpj;
    }

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

Implementação dos gerenciadores:

- **PessoaFisicaRepo**

```
package model;

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

public class PessoaFisicaRepo {
   private List<PessoaFisica> pessoasFisicas;

   public PessoaFisicaRepo() {
      this.pessoasFisicas = new ArrayList<>();
   }

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

   public void alterar(PessoaFisica pessoaFisica) {
      for (PessoaFisica pf : pessoasFisicas) {
         if (pf.getId() == pessoaFisica.getId()) {
            pf.setNome(pessoaFisica.getNome());
            pf.setCpf(pessoaFisica.getCpf());
            pf.setIdade(pessoaFisica.getIdade());
            return;
         }
      }
      throw new IllegalArgumentException("Pessoa Física com o ID " +
pessoaFisica.getId() + " não encontrada.");
   }

   public void excluir(int id) {
      Iterator<PessoaFisica> iterator = pessoasFisicas.iterator();
      while (iterator.hasNext()) {
         PessoaFisica pf = iterator.next();
         if (pf.getId() == id) {
            iterator.remove();
            return;
         }
      }
      throw new IllegalArgumentException("Pessoa Física com o ID " + id + " não
encontrada.");
   }

   public PessoaFisica obter(int id) {
      for (PessoaFisica pf : pessoasFisicas) {
         if (pf.getId() == id) {
```

```
            return pf;
        }
      }
      throw new IllegalArgumentException("Pessoa Física com o ID " + id + " não
encontrada.");
   }

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

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

   public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
      try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
         pessoasFisicas = (List<PessoaFisica>) inputStream.readObject();
      }
   }
}
```

- **PessoaJuridicaRepo**

```
package model;

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

public class PessoaJuridicaRepo {
   private List<PessoaJuridica> pessoasJuridicas;

   public PessoaJuridicaRepo() {
      this.pessoasJuridicas = new ArrayList<>();
   }

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

   public void alterar(int id, PessoaJuridica pessoaJuridica) {
      for (int i = 0; i < pessoasJuridicas.size(); i++) {
```

```
            PessoaJuridica pj = pessoasJuridicas.get(i);
            if (pj.getId() == id) {
                pessoasJuridicas.set(i, pessoaJuridica);
                return;
            }
        }
        throw new IllegalArgumentException("Pessoa Jurídica com o ID " + id + " não
encontrada.");
    }

    public void excluir(int id) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            PessoaJuridica pj = pessoasJuridicas.get(i);
            if (pj.getId() == id) {
                pessoasJuridicas.remove(i);
                return;
            }
        }
        throw new IllegalArgumentException("Pessoa Jurídica com o ID " + id + " não
encontrada.");
    }

    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pj : pessoasJuridicas) {
            if (pj.getId() == id) {
                return pj;
            }
        }
        throw new IllegalArgumentException("Pessoa Jurídica com o ID " + id + " não
encontrada.");
    }

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

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

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException
{
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoasJuridicas = (List<PessoaJuridica>) inputStream.readObject();
```

```
        }
    }

    public void alterar(PessoaJuridica pessoa) {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

Classe principal

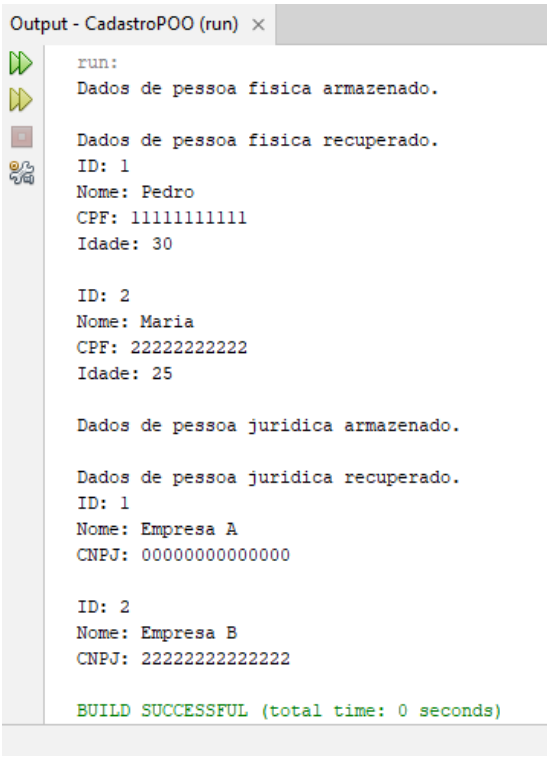
- **CadastroPOO**

```
import model.*;

import java.io.IOException;
import java.util.List;

public class CadastroPOO {
    public static void main(String[] args) {
        testarPessoasFisicas();
        testarPessoasJuridicas();
    }
    public static void testarPessoasFisicas() {
        // Instanciar pessoas físicas.
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        // Adicionar duas pessoas físicas.
        repo1.inserir(new PessoaFisica(1, "Pedro", "11111111111", 30));
        repo1.inserir(new PessoaFisica(2, "Maria", "22222222222", 25));
        System.out.println("Dados de pessoa fisica armazenado.");

        try {
            repo1.persistir("pessoas_fisicas.dat");

                    PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoas_fisicas.dat");
            System.out.println("\nDados de pessoa fisica recuperado.");

            exibirPessoasFisicas(repo2.obterTodos());
        } catch (IOException | ClassNotFoundException e) {
        }
    }

    public static void testarPessoasJuridicas() {

        // Instanciar pessoas jurídicas.
        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
```

```
        // Adicionar duas pessoas jurídicas.
        repo3.inserir(new PessoaJuridica(1, "Empresa A", "00000000000000"));
        repo3.inserir(new PessoaJuridica(2, "Empresa B", "22222222222222"));
        System.out.println("Dados de pessoa juridica armazenado.");

        try {

            repo3.persistir("pessoas_juridicas.dat");

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar("pessoas_juridicas.dat");

            // Mostrar dados das pessoas jurídicas recuperadas.
            System.out.println("\nDados de pessoa juridica recuperado.");
            exibirPessoasJuridicas(repo4.obterTodos());
        } catch (IOException | ClassNotFoundException e) {
        }

    }

    public static void exibirPessoasFisicas(List<PessoaFisica> pessoasFisicas) {
        for (PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
            System.out.println(); // Pular linha entre cada pessoa física
        }
    }

    public static void exibirPessoasJuridicas(List<PessoaJuridica> pessoasJuridicas) {
        for (PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
            System.out.println();
        }
    }
}
```

Resultados da Execução dos Códigos:

Os dados das pessoas físicas e jurídicas foram inseridos nos repositórios e persistidos em arquivos binários.
Os dados foram recuperados dos arquivos e exibidos na tela com sucesso, demonstrando a funcionalidade de persistência em arquivos.



Análise e Conclusão:

Vantagens e Desvantagens do Uso de Herança:

- **Vantagens:** A herança permite a reutilização de código, promove uma estrutura mais organizada e hierárquica, e facilita a manutenção do software. Também permite a implementação do polimorfismo, que é útil em situações onde diferentes classes compartilham um comportamento comum.
- **Desvantagens:** A herança pode levar a um acoplamento excessivo entre classes, tornando o código mais difícil de entender e manter. Além disso, uma hierarquia de herança muito profunda pode levar a problemas de design e performance.

Necessidade da Interface Serializable para Persistência em Arquivos Binários: A interface Serializable é necessária ao efetuar persistência em arquivos binários porque ela marca uma classe como serializável, ou seja, permite que seus objetos sejam convertidos em uma sequência de bytes que podem ser salvos em um arquivo e posteriormente recuperados. Isso é essencial para armazenar objetos em arquivos de forma persistente.

Uso do Paradigma Funcional pela API Stream no Java: A API Stream no Java utiliza o paradigma funcional para permitir operações de processamento de dados de forma mais declarativa e expressiva. Com a API Stream, é possível manipular coleções de dados de maneira eficiente e concisa, utilizando operações como map, filter, reduce, entre outras, sem a necessidade de loops explícitos.

Padrão de Desenvolvimento na Persistência de Dados em Arquivos em Java: No desenvolvimento Java, o padrão de persistência de dados em arquivos geralmente segue o padrão de design DAO (Data Access Object). Esse padrão separa a lógica de negócios da

lógica de acesso aos dados, fornecendo uma camada de abstração para interagir com o sistema de armazenamento de dados, seja ele um banco de dados ou arquivos. Isso facilita a manutenção, testabilidade e flexibilidade do sistema.

Conclusão:

A prática de implementação de cadastro de clientes com persistência em arquivos em Java proporcionou uma oportunidade valiosa para aplicar conceitos fundamentais de programação orientada a objetos, como herança, polimorfismo e interfaces, além de explorar técnicas modernas de persistência de dados. Através da análise dos resultados e das questões levantadas, foi possível compreender as vantagens e desvantagens de diferentes abordagens de programação e padrões de desenvolvimento, contribuindo para uma melhor compreensão da programação em Java e suas aplicações práticas.