- **Campus: Polo Méier**
- **Curso: Desenvolvimento Full Stack**
- **Disciplina: Introdução à Programação OO em Java**
- **Número da Turma: EAD**
- **Semestre Letivo: 2024.4**
- **Nome do Integrante da Prática: Gabriel dos Reis Ferreira**

Cadastro de Pessoas Físicas e Jurídicas com Persistência e Recuperação de Dados

Objetivo da Prática

Desenvolver um sistema de cadastro para pessoas físicas e jurídicas, que permita realizar operações de inclusão, alteração, exclusão, exibição e recuperação de dados, com persistência em arquivos. A prática também teve como objetivo reforçar conceitos de orientação a objetos, como herança, polimorfismo e encapsulamento, além do uso de classes repositório para organização do código e da manipulação de arquivos em Java.

Códigos Solicitados

Os códigos solicitados incluem a implementação de classes para representar pessoas físicas e jurídicas, repositórios para gerenciar dados, e funcionalidades de persistência e recuperação de dados. Abaixo estão os trechos de código implementados.

Código da Classe **Pessoa**

```
package model;

import java.io.Serializable;

public abstract class Pessoa implements Serializable {

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 abstract void exibir();package model;
import java.io.Serializable;
public abstract class Pessoa implements Serializable {
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 abstract void exibir();
}
}
```

Código da Classe **PessoaFisica**

```
package model;
public class PessoaFisica extends Pessoa {
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 void exibir() {
System.out.println("ID: " + getId() + ", Nome: " + getNome());
System.out.println("CPF: " + cpf + ", Idade: " + idade);
}
}
```

Código da Classe **PessoaJuridica**

```
package model;
public class PessoaJuridica extends Pessoa {
private String cnpj;
public PessoaJuridica(int id, String nome, String cnpj) {
super(id, nome);
this.cnpj = cnpj;
}
public void exibir() {
System.out.println("ID: " + getId() + ", Nome: " + getNome());
System.out.println("CNPJ: " + cnpj);
}
}
```

Código da Classe **PessoaFisicaRepo**

```
package model;
import java.io.*;
import java.util.ArrayList;
public class PessoaFisicaRepo {
```

```
private ArrayList<PessoaFisica> lista = new ArrayList<>();
// Método inserir
public void inserir(PessoaFisica pessoaFisica) {
lista.add(pessoaFisica);
}
// Método alterar
public void alterar(PessoaFisica pessoaFisica) {
for (int i = 0; i < lista.size(); i++) {
if (lista.get(i).getId() == pessoaFisica.getId()) {
lista.set(i, pessoaFisica);
return;
}
}
}
// Método excluir
public void excluir(int id) {
lista.removeIf(pessoa -> pessoa.getId() == id);
}
// Método obter
public PessoaFisica obter(int id) {
for (PessoaFisica pessoa : lista) {
if (pessoa.getId() == id) {
return pessoa;
}
}
return null;
}
// Método obterTodos
```

```
public ArrayList<PessoaFisica> obterTodos() {

return new ArrayList<>(lista);

}

// Método persistir (salvar no disco)

public void persistir(String nomeArquivo) throws IOException {

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

oos.writeObject(lista);

}

}

// Método recuperar (carregar do disco)

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

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

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

}

}

}
```

Código da Classe **PessoaJuridicaRepo**

```
package model;

import java.io.*;

import java.util.ArrayList;

public class PessoaJuridicaRepo {

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

public void inserir(PessoaJuridica pessoaJuridica) {

lista.add(pessoaJuridica);

}

public void persistir(String nomeArquivo) throws IOException {

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

```
oos.writeObject(lista);
}
}
public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
lista = (ArrayList<PessoaJuridica>) ois.readObject();
}
}

public ArrayList<PessoaJuridica> obterTodos() {
return new ArrayList<>(lista);
}
}
```

Código da Classe **Main**

```
package model;
import java.io.IOException;
public class Main {
public static void main(String[] args) {
try {
System.setOut(new java.io.PrintStream(System.out, true, "UTF-8"));
System.out.println("=== Cadastro de Pessoas ===");
// Repositório de pessoas físicas
PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
repo1.inserir(new PessoaFisica(1, "João Silva", "123.456.789-00", 30));
repo1.inserir(new PessoaFisica(2, "Maria Oliveira", "987.654.321-00", 25));
repo1.persistir("pessoas_fisicas.dat");
// Recuperando dados de pessoas físicas
PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
```

```
repo2.recuperar("pessoas_fisicas.dat");
System.out.println("\nPessoas Físicas recuperadas:");
for (PessoaFisica pf : repo2.obterTodos()) {
pf.exibir();
}
// Repositório de pessoas jurídicas
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
repo3.inserir(new PessoaJuridica(1, "Minalba", "12.345.678/0001-90"));
repo3.inserir(new PessoaJuridica(2, "Natura", "98.765.432/0001-11"));
repo3.persistir("pessoas_juridicas.dat");
// Recuperando dados de pessoas jurídicas
PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
repo4.recuperar("pessoas_juridicas.dat");
System.out.println("\nPessoas Jurídicas recuperadas:");
for (PessoaJuridica pj : repo4.obterTodos()) {
pj.exibir();
}
} catch (IOException | ClassNotFoundException e) {
System.err.println("Erro: " + e.getMessage());
}
}
}
```

Resultados da Execução

Após a execução dos códigos, o sistema apresenta o seguinte resultado no console:

=== Cadastro de Pessoas ===

Pessoas Físicas recuperadas:

ID: 1, Nome: João Silva

CPF: 123.456.789-00, Idade: 30

ID: 2, Nome: Maria Oliveira

CPF: 987.654.321-00, Idade: 25

Pessoas Jurídicas recuperadas:

ID: 1, Nome: Minalba

CNPJ: 12.345.678/0001-90

ID: 2, Nome: Natura

CNPJ: 98.765.432/0001-11

Análise e Conclusão

1. Quais as vantagens e desvantagens do uso de herança?

Vantagens:

Reutilização de código: a classe filha pode reutilizar os métodos e atributos que são definidos em uma classe pai, removendo assim a duplicação de código.

Estrutura: fornece um relacionamento estrutural claro entre classes, tornando o fluxo do programa fácil de entender e alterar.

Extensível: recursos da classe pai podem ser adicionados ou substituídos em classes filhas. Facilita a flexibilidade.

Desvantagens:

Acoplamento aumentado: a classe filha é diretamente acoplada à implementação da classe pai, o que pode ser difícil para os desenvolvedores entenderem.

Complexidade: hierarquias profundas podem se tornar difíceis de manejar e compreender.

Reutilização limitada: apenas uma classe pode estender mais uma classe (Java não suporta heranças múltiplas de classes), então isso limita as maneiras como essa desvantagem pode ser útil.

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

A JVM precisa tomar nota dessa interface para permitir que um objeto seja convertido em um fluxo de bytes. É isso que podemos chamar de serialização. Para que serve?

Persistência: Você pode armazenar o estado de um objeto em um arquivo ou enviá-lo por uma rede.

Reconstrução: A capacidade de desserializar, lendo o fluxo de bytes para recriar o objeto.

Justificativa: Não questione a necessidade de Serializable e suas funções com relação à persistência de arquivos binários. Sem a implementação Serializable, a JVM não pode empacotar o objeto em bytes, então ela levanta uma exceção do tipo NotSerializableException.

3. Como o paradigma funcional é usado pela Stream API em Java?

O paradigma funcional é usado na Java Stream API da seguinte forma:

• **Funcionalidade de objeto de primeira classe:** fornece lambdas e interfaces funcionais para ajudar a implementar o conceito de mapeamento, filtragem e redução para operações de fluxo.

• **Imutabilidade:** as operações de fluxo não alteram nenhum dado original, mas criam novos resultados. Como uma versão aprimorada:

• **Operações declarativas:** permitem que você escreva código expressivo, declarando o que fazer, não como fazer.

• **Avaliação preguiçosa:** as operações são avaliadas somente quando necessário, o que otimiza o desempenho.

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

Um dos métodos de desenvolvimento mais utilizados para armazenar dados em arquivos no Java é o Data Access Object (DAO). Especificamente, este método consiste:

Divide a Lógica de Armazenamento de Dados em Etapas Distinctas : Separa as atividades de busca e modificação de dados em classes dedicadas.

- Simplifica a Manutenção - Concentra a lógica de armazenamento para possibilitar alterações no sistema de armazenamento sem interferir na lógica empresarial.

Abstração simplifica as operações de CRUD (Criar, Ler, Atualizar, Excluir) com uma interface fácil de entender.

Na prática cotidiana, os arquivos (Repositório de Pessoas Físicas e Repositório de PessoasJurídicas) adotam diretrizes semelhantes às do DAO, estruturando a obtenção e edição de informações de maneira encapsulada.

O código-fonte do projeto está disponível no repositório Git:
https://github.com/GabrielR3isS/GabrielR3isS-Miss-o-Pr-tica-N-vel-1-Mundo-3.git