Estácio

FACULDADE ESTÁCIO

CÂMPUS VOLTA REDONDA – RJ

DESENVOLVIMENTO FULL STACK

DISCIPLINA – INICIANDO O CAMINHO PELO JAVA

TURMA – 2023.2

SEMESTRE – 3

VOLTA REDONDA, JUNHO 2024.

DESENVOLVIMENTO FULL STACK

DISCIPLINA – INICIANDO O CAMINHO PELO JAVA

TURMA – 2023.2

SEMESTRE – 3

ALUNO – BRUNO SAMPAIO BASTOS

TUTOR – MARIA MANSO

GITHUB - https://github.com/BrunoTI-Code?tab=repositories

VOLTA REDONDA, JUNHO 2024.

RESUMO

O projeto Cadastro POO visa criar um sistema de cadastro para pessoas físicas e jurídicas utilizando os princípios da Programação Orientada a Objetos (POO) na linguagem Java. Desenvolvido no NetBeans como uma aplicação Java do tipo Ant, o projeto inclui um pacote "model" que contém as classes Pessoa, PessoaFisica, e PessoaJuridica, todas implementando a interface Serializable. A classe Pessoa possui os campos id e nome, enquanto PessoaFisica e PessoaJuridica herdam de Pessoa, adicionando os campos cpf e idade, e cnpj, respectivamente.

Para gerenciar essas entidades, foram criadas as classes PessoaFisicaRepo e PessoaJuridicaRepo, que utilizam listas para armazenar as entidades e têm métodos para inserir, alterar, excluir, obter e persistir os dados em arquivos. O método main da classe principal foi modificado para testar as funcionalidades dos repositórios, incluindo a persistência e recuperação de dados, demonstrando a correta manipulação e armazenamento das informações.

O projeto assegura a integridade e a organização dos dados, facilitando a manipulação e persistência das informações de maneira eficiente. O código-fonte e a documentação completa do projeto foram armazenadas em um repositório Git, facilitando o acesso e a revisão.

Palavras-chave: Programação Orientada a Objetos (POO), Java, NetBeans, Manipulação de dados, Serialização, Gerenciamento de Dados, Persistência de Dados, Teste de Funcionalidades, Repositório Git, Eficiência e Organização.

SUMARIO

1 INTRODUÇÃO

1.1 CADASTRO POO

O objetivo da prática desenvolvida no projeto CadastroPOO é aplicar os princípios da Programação Orientada a Objetos (POO) em um sistema de cadastro de pessoas físicas e jurídicas usando a linguagem Java. Este projeto foi criado no ambiente de desenvolvimento NetBeans, utilizando o Ant para automatização do processo de compilação.

1.2 OBJETIVO DA PRÁTICA

O objetivo principal é implementar um sistema que permita a criação, edição, exclusão e consulta de registros de pessoas físicas e jurídicas, aplicando conceitos fundamentais de POO como herança, encapsulamento e polimorfismo. Além disso, será explorada a persistência de dados utilizando o mecanismo de serialização em arquivos, garantindo que as informações sejam armazenadas de forma segura e acessível.

1.3 O QUE FAZER

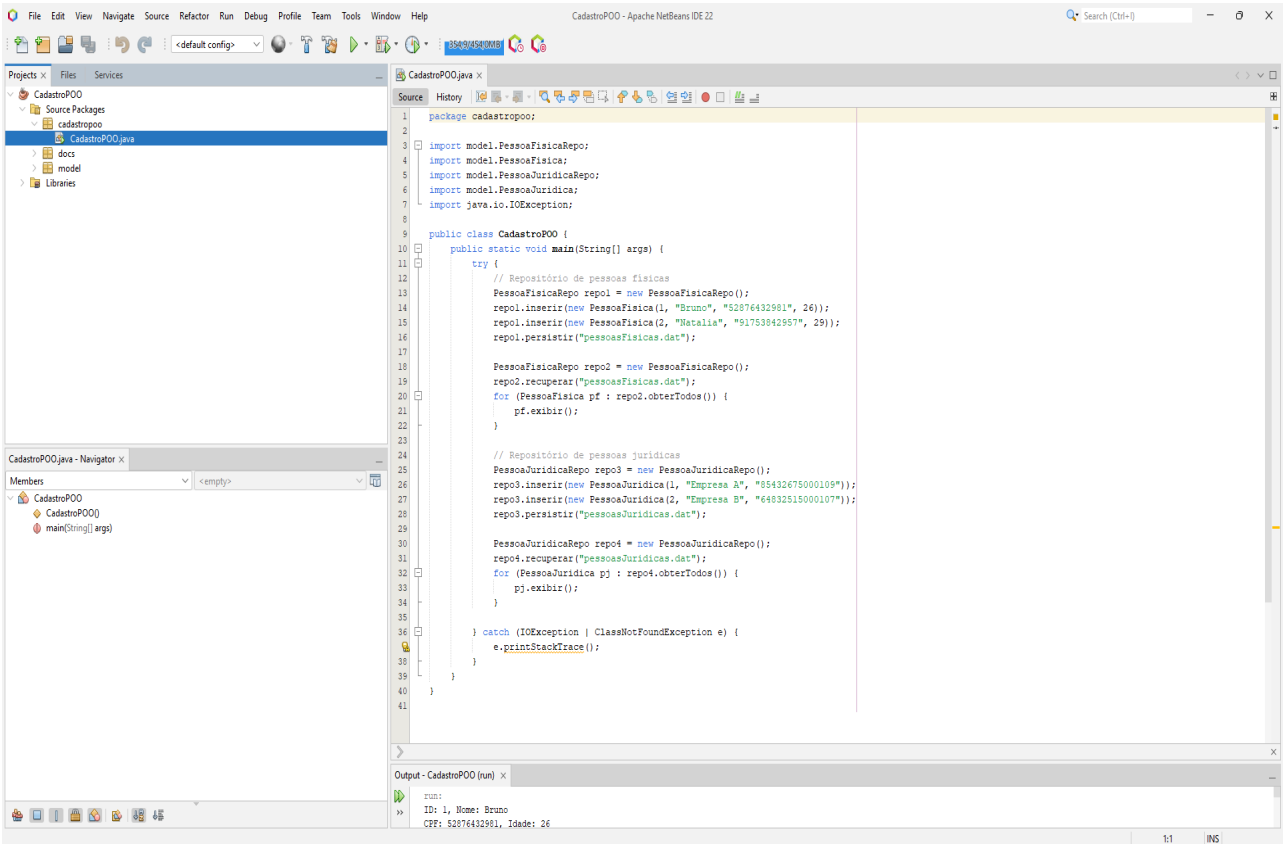
Será desenvolvido um conjunto de classes no pacote "model", incluindo Pessoa, PessoaFisica e PessoaJuridica, cada uma implementando a interface Serializable para permitir a persistência em arquivos. Adicionalmente, serão criados os gerenciadores PessoaFisicaRepo e PessoaJuridicaRepo, responsáveis por manipular as listas de entidades e realizar operações como inserção, alteração, exclusão e consulta.

1.4 COMO FAZER

Inicialmente, o projeto será criado no NetBeans e o pacote "model" será estruturado para as entidades e gerenciadores. As classes Pessoa, PessoaFisica e PessoaJuridica serão implementadas com seus respectivos atributos, métodos construtores, getters, setters e métodos de exibição. Em seguida, os gerenciadores PessoaFisicaRepo e PessoaJuridicaRepo serão desenvolvidos, integrando métodos para manipulação das listas de entidades e operações de persistência em arquivos.

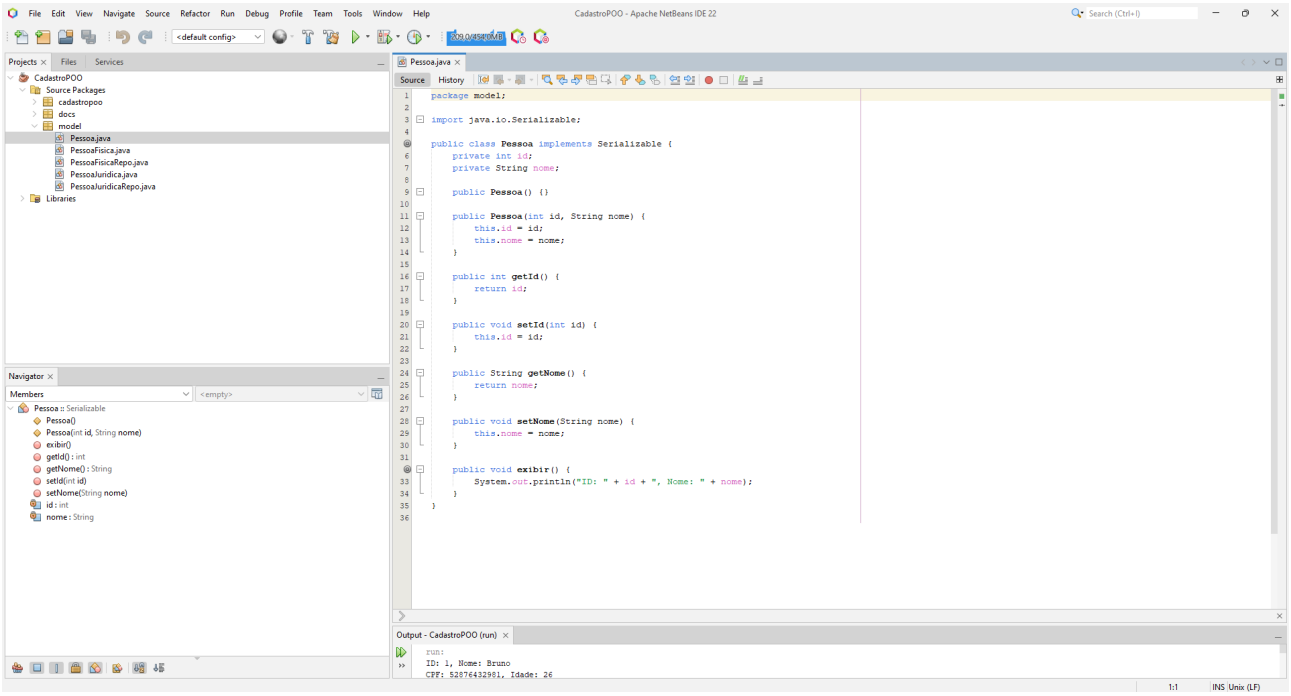
2 CADASTRO POO

2.1 CÓDIGO CADASTRO POO

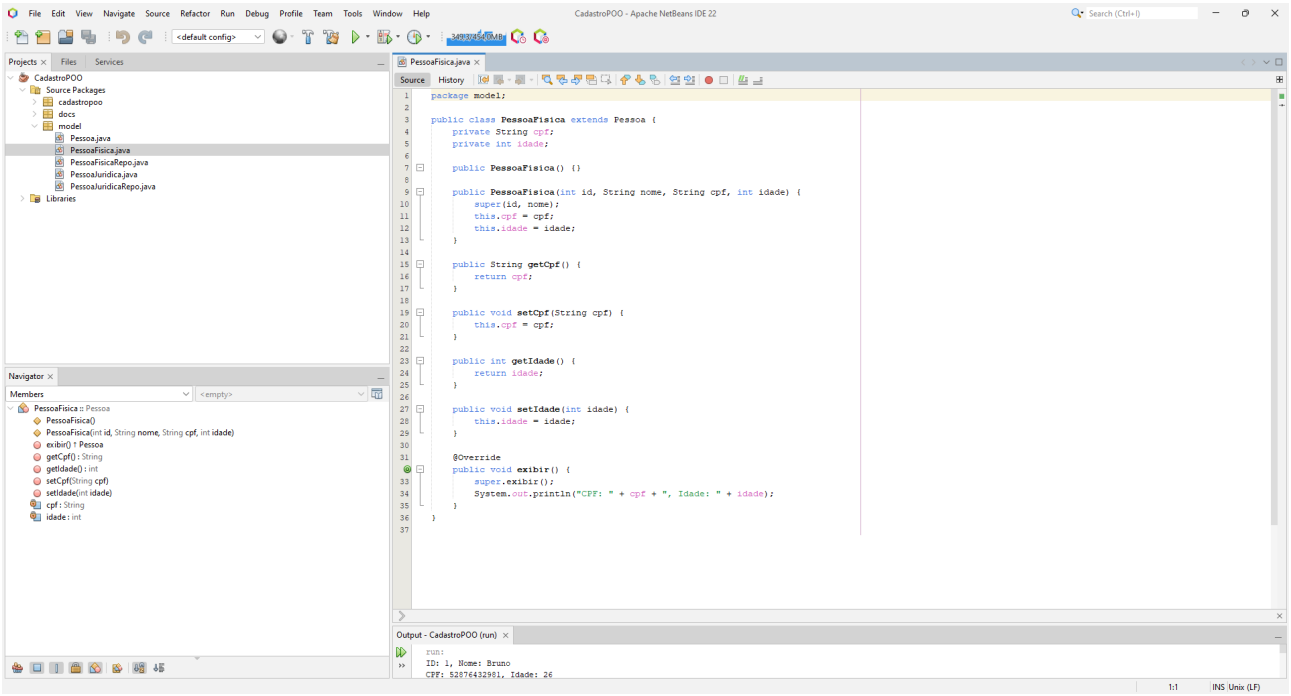


3 MODEL

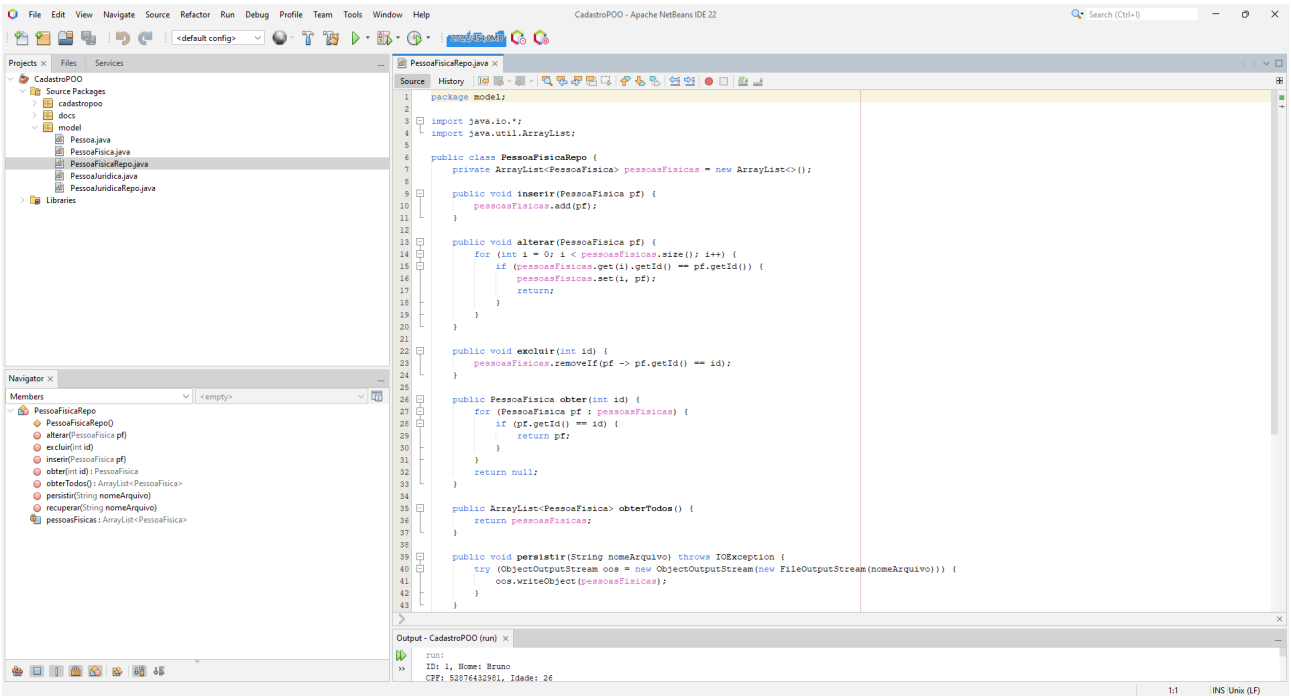
3.1 PESSOA.JAVA

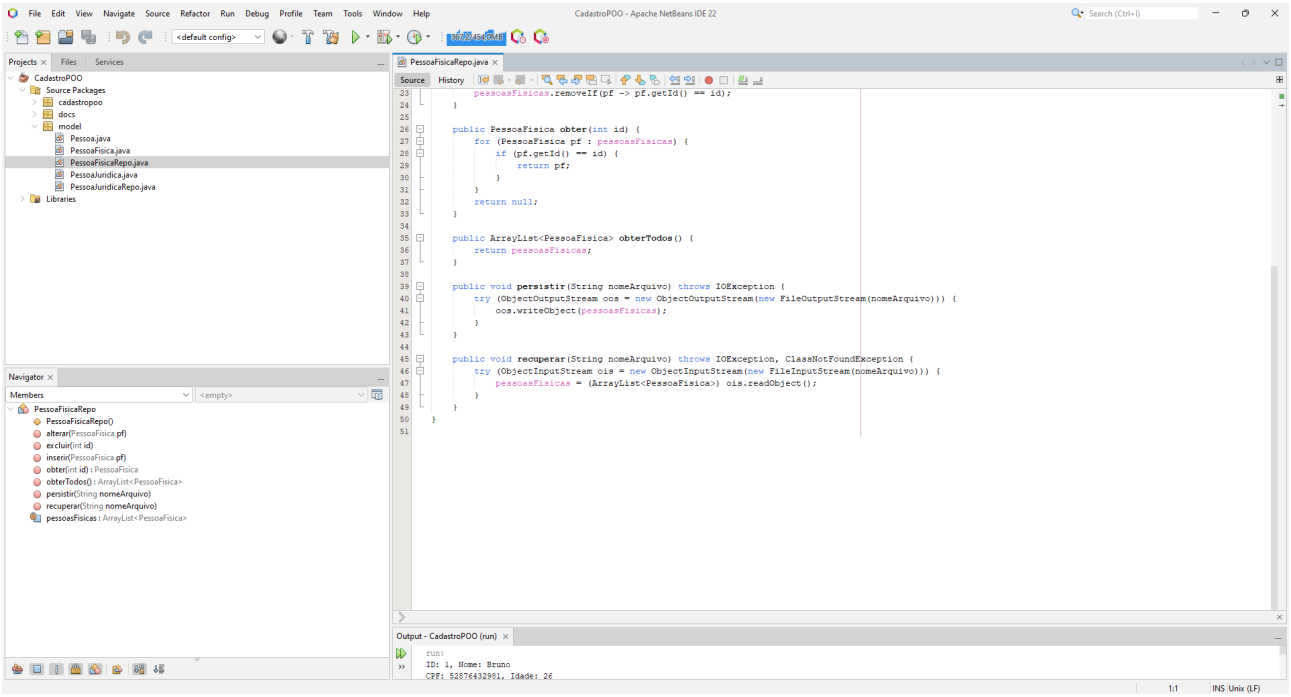


3.2 PESSOA FISICA.JAVA

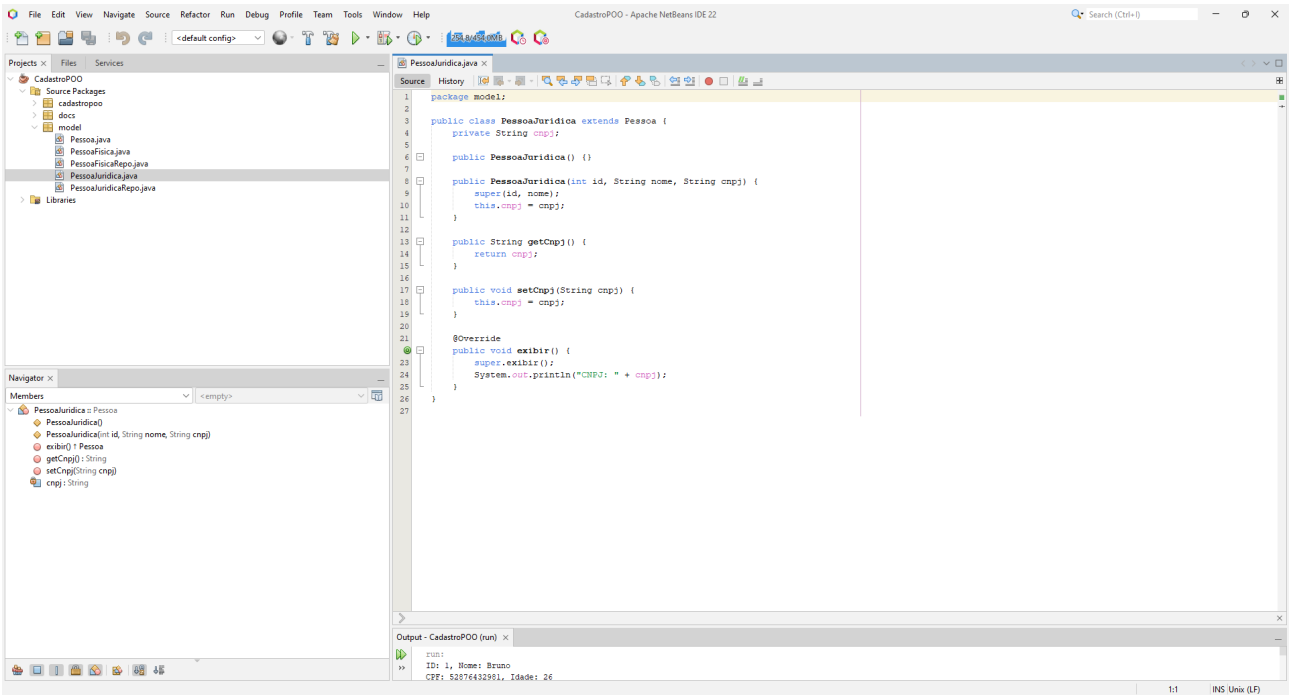


3.3 PESSOA FISICA REPO.JAVA

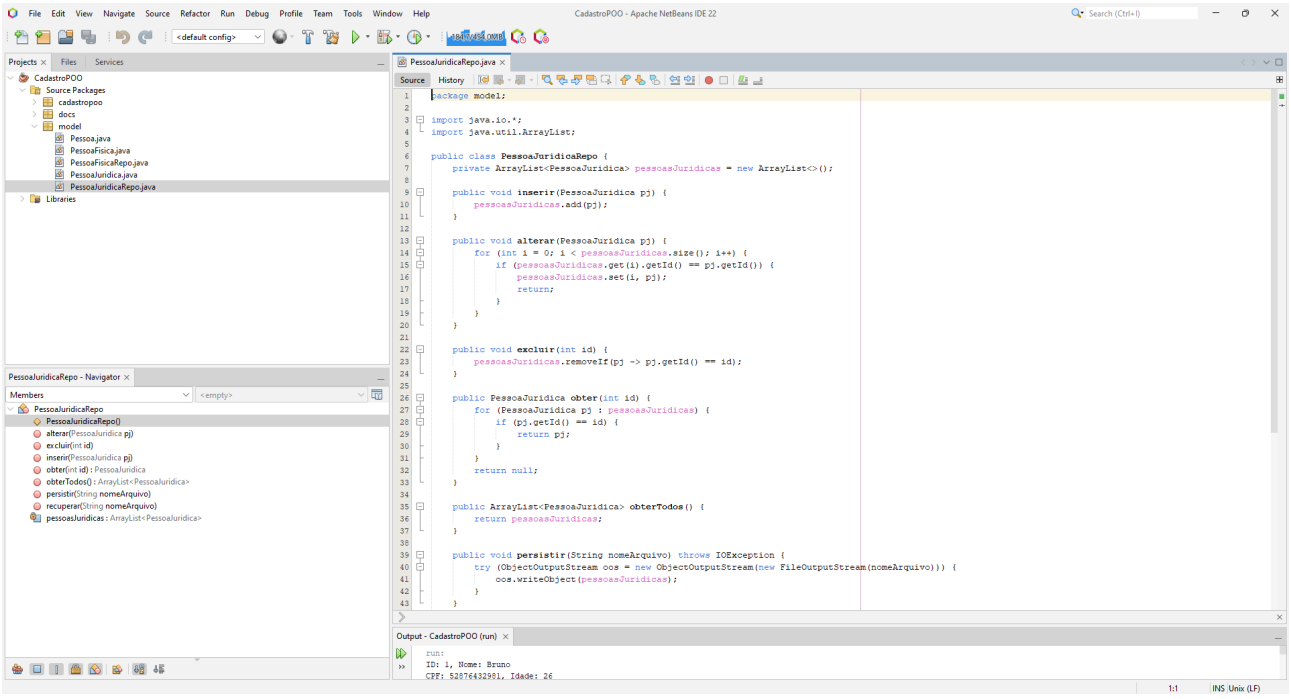




3.4 PESSOA JURIDICA.JAVA

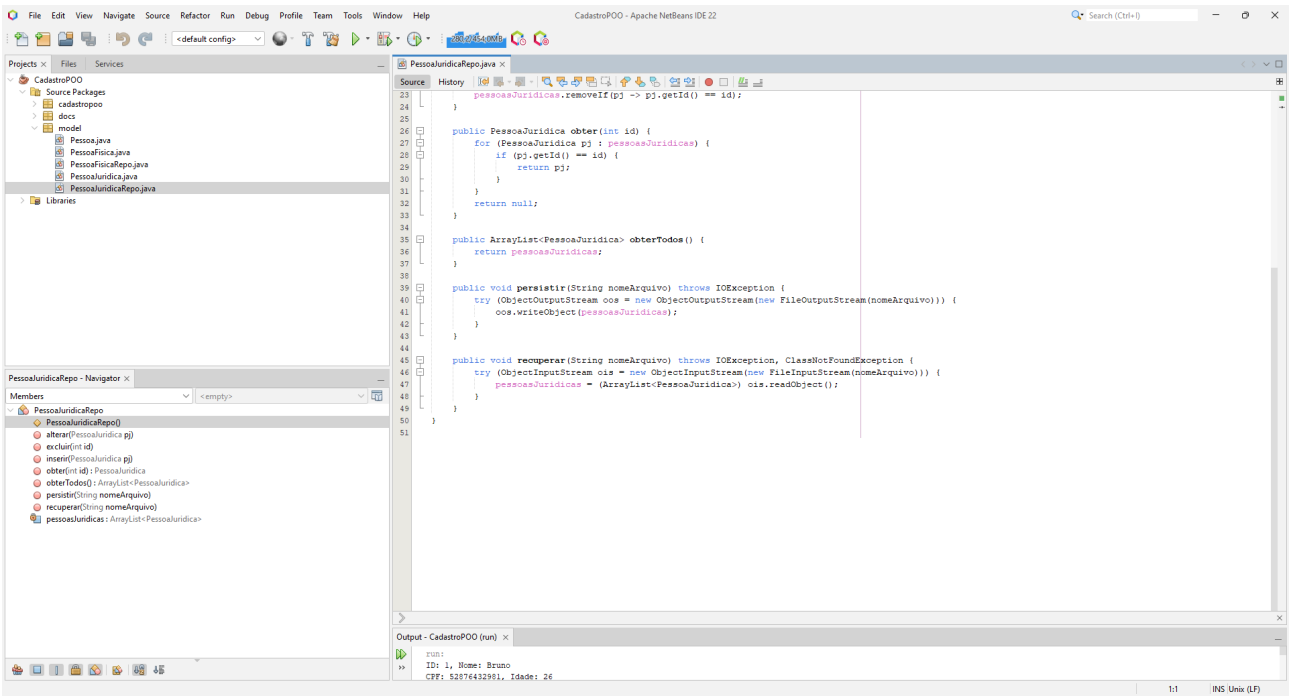


3.5 PESSOA JURIDICA REPO.JAVA



```
package model;

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

public class PessoaJuridicaRepo {
    private ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

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

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

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

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

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

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

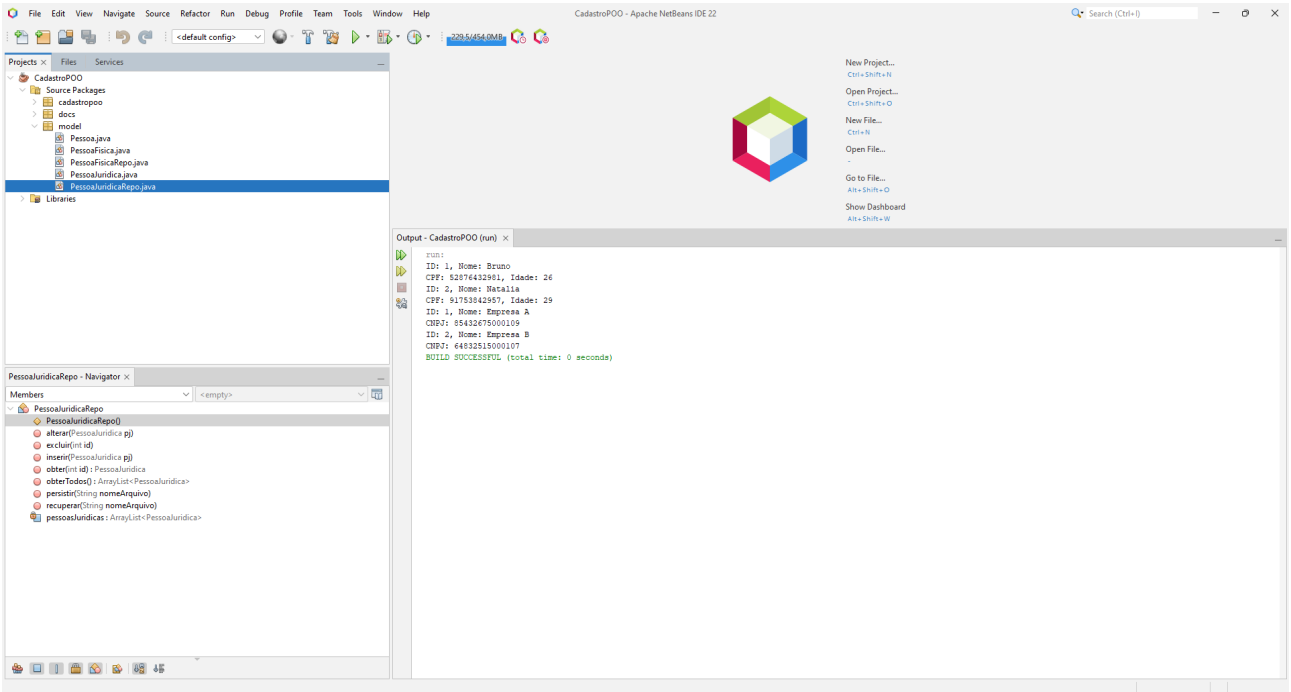
```
Output - CadastroPOO (run)
run:
ID: 1, Nome: Bruno
CPF: 52876432981, Idade: 26
```



```
    }

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

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

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

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

```
Output - CadastroPOO (run)
run:
ID: 1, Nome: Bruno
CPF: 52876432981, Idade: 26
```

4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
CadastroPOO - Apache NetBeans IDE 22
Search (Ctrl+I)
<default config>
Projects
Files
Services
CadastroPOO
Source Packages
cadastropoo
docs
model
Pessoa.java
PessoaFisica.java
PessoaFisicaRepo.java
PessoaJuridica.java
PessoaJuridicaRepo.java
Libraries
New Project... Ctrl+Shift+N
Open Project... Ctrl+Shift+O
New File... Ctrl+N
Open File...
Go to File... Alt+Shift+O
Show Dashboard Alt+Shift+W
PessoaJuridicaRepo - Navigator
Members
<empty>
PessoaJuridicaRepo
PessoaJuridicaRepo()
alterar(PessoaJuridica pj)
excluir(int id)
inserir(PessoaJuridica pj)
obter(int id) : PessoaJuridica
obterTodos() : ArrayList<PessoaJuridica>
persistir(String nomeArquivo)
recuperar(String nomeArquivo)
pessoasJuridicas : ArrayList<PessoaJuridica>
Output - CadastroPOO (run)
run:
ID: 1, Nome: Bruno
CPF: 52876432981, Idade: 26
ID: 2, Nome: Natalia
CPF: 91753842957, Idade: 29
ID: 1, Nome: Empresa A
CNPJ: 85432675000109
ID: 2, Nome: Empresa B
CNPJ: 64832515000107
BUILD SUCCESSFUL (total time: 0 seconds)

5 ANALISE

5.1 VANTAGENS E DESVANTAGENS DO USO DE HERANÇA

O uso de herança em Programação Orientada a Objetos (POO) traz várias vantagens, como a reutilização de código, permitindo que classes derivadas (subclasses) herdem atributos e métodos de classes base (superclasses). Isso facilita a manutenção e a extensão do sistema, uma vez que alterações na classe base são propagadas para as subclasses, reduzindo a duplicação de código e promovendo a consistência. Além disso, a herança permite a criação de uma hierarquia de classes, melhorando a organização do código e possibilitando o uso de polimorfismo.

Entretanto, a herança também apresenta desvantagens. Uma das principais é o acoplamento forte entre classes, que pode dificultar a modificação e a evolução do sistema. Mudanças na classe base podem impactar todas as subclasses, potencialmente introduzindo erros. Além disso, a herança pode levar a hierarquias de classes complexas e difíceis de entender, especialmente em sistemas grandes. Outro problema é a restrição de herança única em Java, onde uma classe só pode herdar de uma única superclasse, limitando a flexibilidade do design.

5.2 POR QUE A INTERFACE SERIALIZABLE É NECESSÁRIA AO EFETUAR PERSISTÊNCIA EM ARQUIVOS BINÁRIOS?

A interface Serializable é essencial ao efetuar a persistência de objetos em arquivos binários porque ela permite que os objetos sejam convertidos em um formato de byte stream, que pode ser facilmente gravado e lido de um arquivo. Em Java, a serialização é o processo de transformar um objeto em uma sequência de bytes, que inclui os dados do objeto e informações sobre o tipo do objeto e os tipos de dados armazenados nele. Ao implementar Serializable, garantimos que a classe tem a capacidade de ser serializada e deserializada, permitindo a persistência e a recuperação dos objetos de forma eficiente e segura.

5.3 COMO O PARADIGMA FUNCIONAL É UTILIZADO PELA API STREAM NO JAVA?

O paradigma funcional é utilizado pela API Stream no Java para oferecer uma forma mais declarativa e expressiva de processar coleções de dados. A API Stream permite que os desenvolvedores utilizem funções lambda e métodos de referência para realizar operações como map, filter, reduce, collect, entre outras, de maneira concisa e eficiente. Isso promove um estilo de programação mais funcional, onde as operações são especificadas em termos de funções e transformações sobre os dados, em vez de loops e manipulações explícitas. Esse paradigma funcional facilita a paralelização e otimiza o processamento de grandes volumes de dados, melhorando a legibilidade e a manutenção do código.

5.4 PADRÃO DE DESENVOLVIMENTO ADOTADO NA PERSISTÊNCIA DE DADOS EM ARQUIVOS

Ao trabalhar com Java, um padrão de desenvolvimento comum adotado na persistência de dados em arquivos é o padrão Data Access Object (DAO). Esse padrão separa a lógica de acesso aos dados da lógica de negócio, encapsulando os detalhes de como os dados são armazenados e recuperados. No contexto da persistência de dados em arquivos, o DAO define métodos específicos para operações de CRUD (Create, Read, Update, Delete), promovendo uma interface clara e consistente para a manipulação dos dados. Isso melhora a modularidade, facilita a troca do mecanismo de persistência (por exemplo, mudando de arquivos para um banco de dados) e torna o código mais testável e manutenível.

6 CONCLUSÃO

O projeto Cadastro POO exemplifica a aplicação prática de vários conceitos fundamentais da Programação Orientada a Objetos e técnicas de persistência em Java. A utilização da herança facilita a reutilização de código e promove a extensibilidade do sistema, apesar das possíveis complexidades associadas ao forte acoplamento entre classes. A implementação da interface Serializable é essencial para permitir a persistência de objetos em arquivos binários, garantindo que o estado dos objetos possa ser salvo e recuperado de forma eficiente. A API Stream, ao incorporar o paradigma funcional, proporciona uma maneira declarativa e expressiva de manipular coleções de dados, promovendo uma programação mais limpa e permitindo a paralelização das operações. Por fim, a adoção do padrão Data Access Object (DAO) para a persistência de dados melhora a modularidade e manutenibilidade do código, separando a lógica de acesso a dados da lógica de negócios. Em conjunto, essas técnicas e padrões resultam em um sistema robusto, flexível e fácil de manter.

7 REFERÊNCIAS

SCIELO, *As Criptomoedas e os novos desafios aos Sistema monetário: Uma abordagem pós-Keynesiana. https://www.scielo.br/j/ecos/a/twmcnj944hvrsbbsn88jnhd. 2020*.

ORACLE, *Java Downloads. Acessado em https://www.oracle.com/java/technologies/downloads/ 2024.*

GeeksforGeeks , *Java Programming Language. Acessado em. https://www.geeksforgeeks.org/java/. 2024*

TutorialsPoint, *Java Tutorial. Acessado em 2024. https://www.tutorialspoint.com/java/index.htm.*

W3Schools, *Java Tutorial. Acessado em 2024. https://www.w3schools.com/java/.*