****

**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 3](#_Toc733925444)

[1.1 Cadastro poo 3](#_Toc571868236)

[1.2 objetivo da prática 3](#_Toc1459267218)

[1.3 o que fazer 4](#_Toc1355471134)

[1.4 como fazer 4](#_Toc1167411024)

[2 Cadastro poo 4](#_Toc996482083)

[2.1 Código Cadastro POO 4](#_Toc1899138761)

[3 model 5](#_Toc2066760964)

[3.1 Pessoa.java 5](#_Toc537881417)

[3.2 Pessoa fisica.java 6](#_Toc1086154136)

[3.3 Pessoa fisica repo.java 6](#_Toc1312030586)

[3.4 Pessoa juridica.java 7](#_Toc1817026934)

[3.5 Pessoa juridica repo.java 8](#_Toc245356233)

[4 resultados da execução dos códigos 9](#_Toc1045316012)

[5 ANALISE 10](#_Toc1249282022)

[5.1 Vantagens e Desvantagens do Uso de Herança 10](#_Toc400539961)

[5.2 Por que a Interface Serializable é Necessária ao Efetuar Persistência em Arquivos Binários? 10](#_Toc1103941822)

[5.3 Como o Paradigma Funcional é Utilizado pela API Stream no Java? 10](#_Toc2135216386)

[5.4 Padrão de Desenvolvimento Adotado na Persistência de Dados em Arquivos 11](#_Toc782725728)

[6 Conclusão 11](#_Toc1827198510)

[7 REFERÊNCIAS 12](#_Toc1738367447)

# INTRODUÇÃO

## 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.

## 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.

## o que fazer

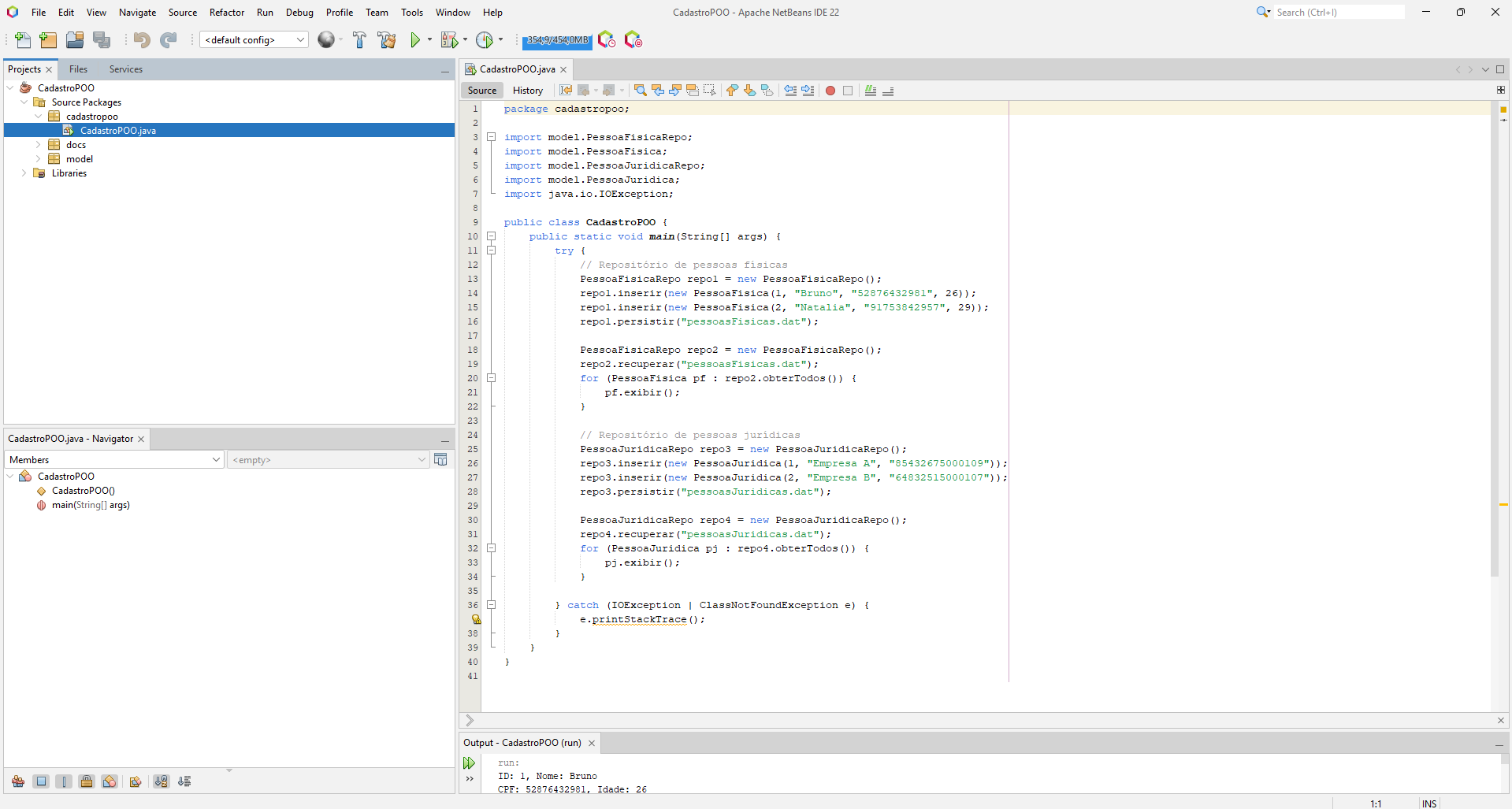
Será desenvolvido um conjunto de classes no pacote "model", incluindo Pessoa, PessoaFisica e PessoaJuridica, cada uma implementando a interface Serializable para permitir a persistência em arquivos. Adicionalmente, serão criados os gerenciadores PessoaFisicaRepo e PessoaJuridicaRepo, responsáveis por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

## 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.

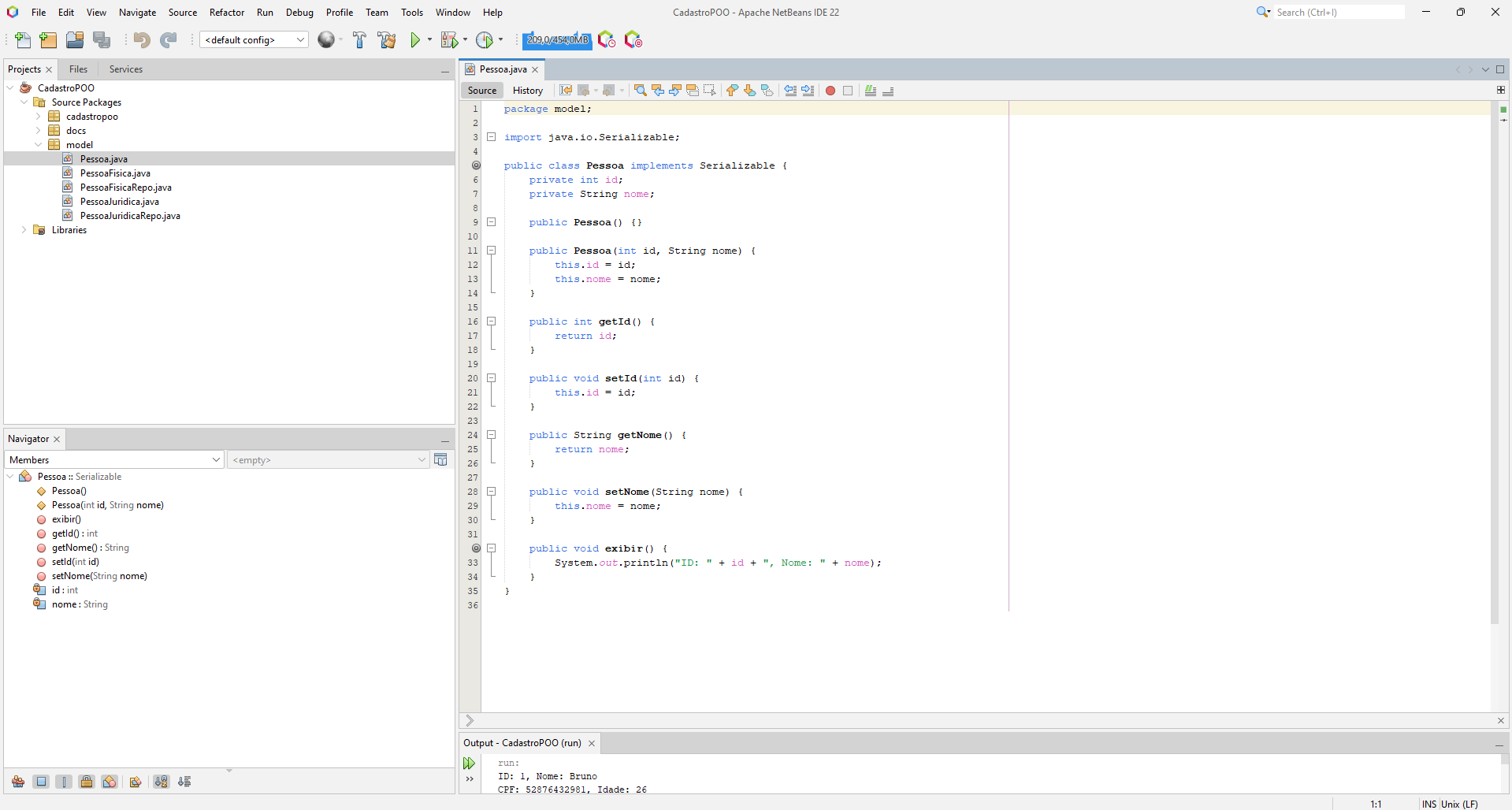
# Cadastro poo

## Código Cadastro POO

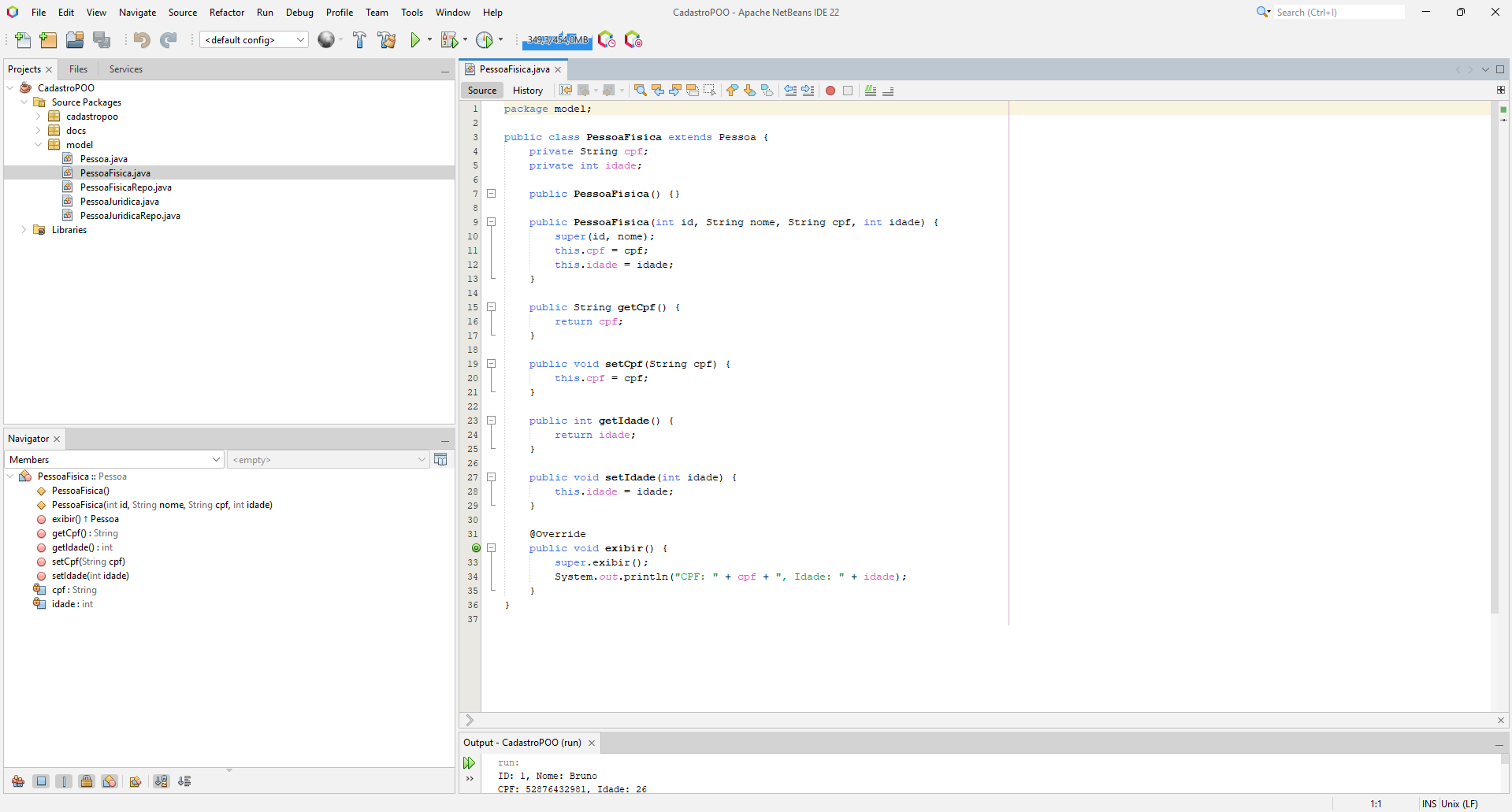


# model

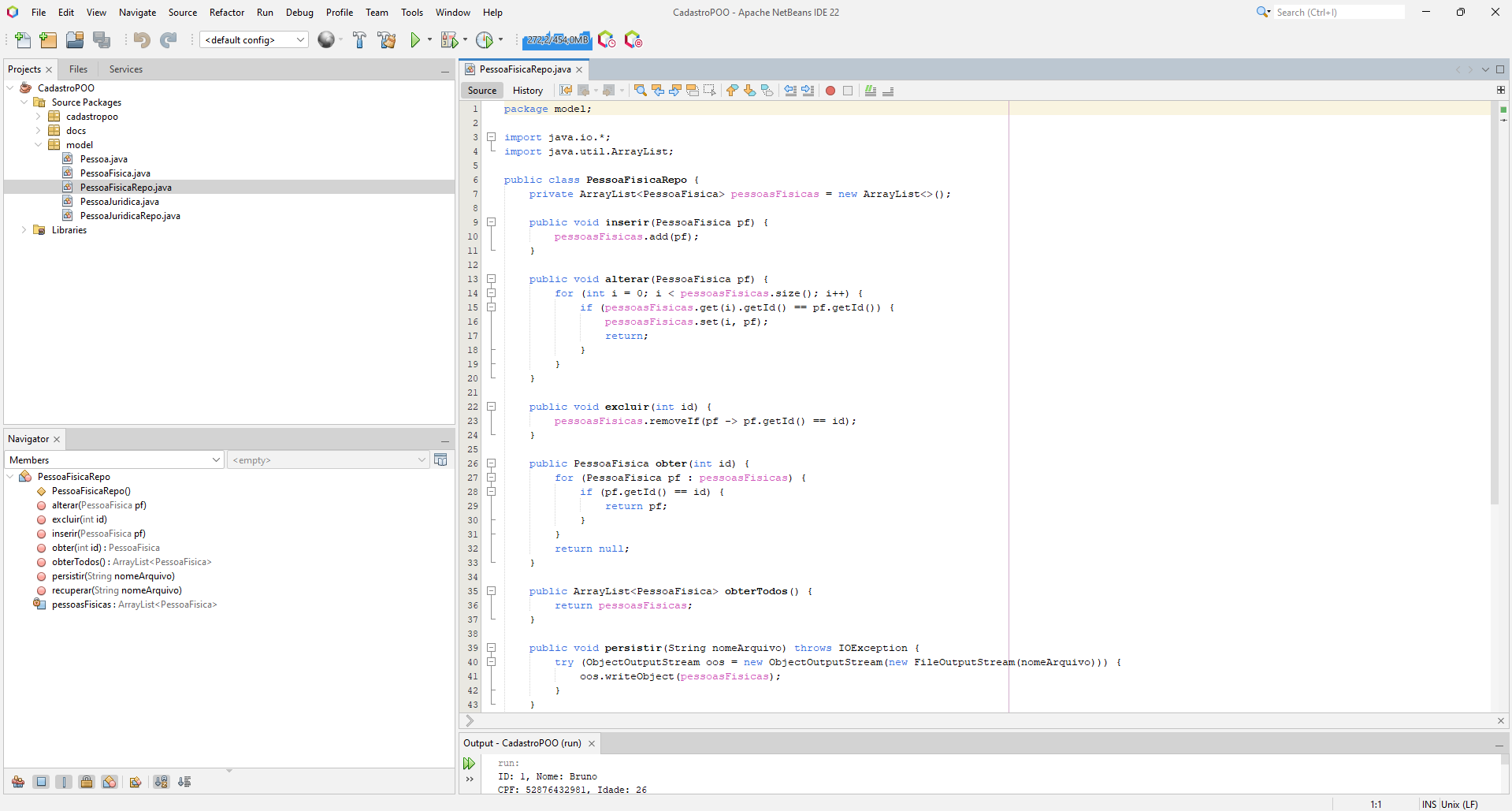
## Pessoa.java

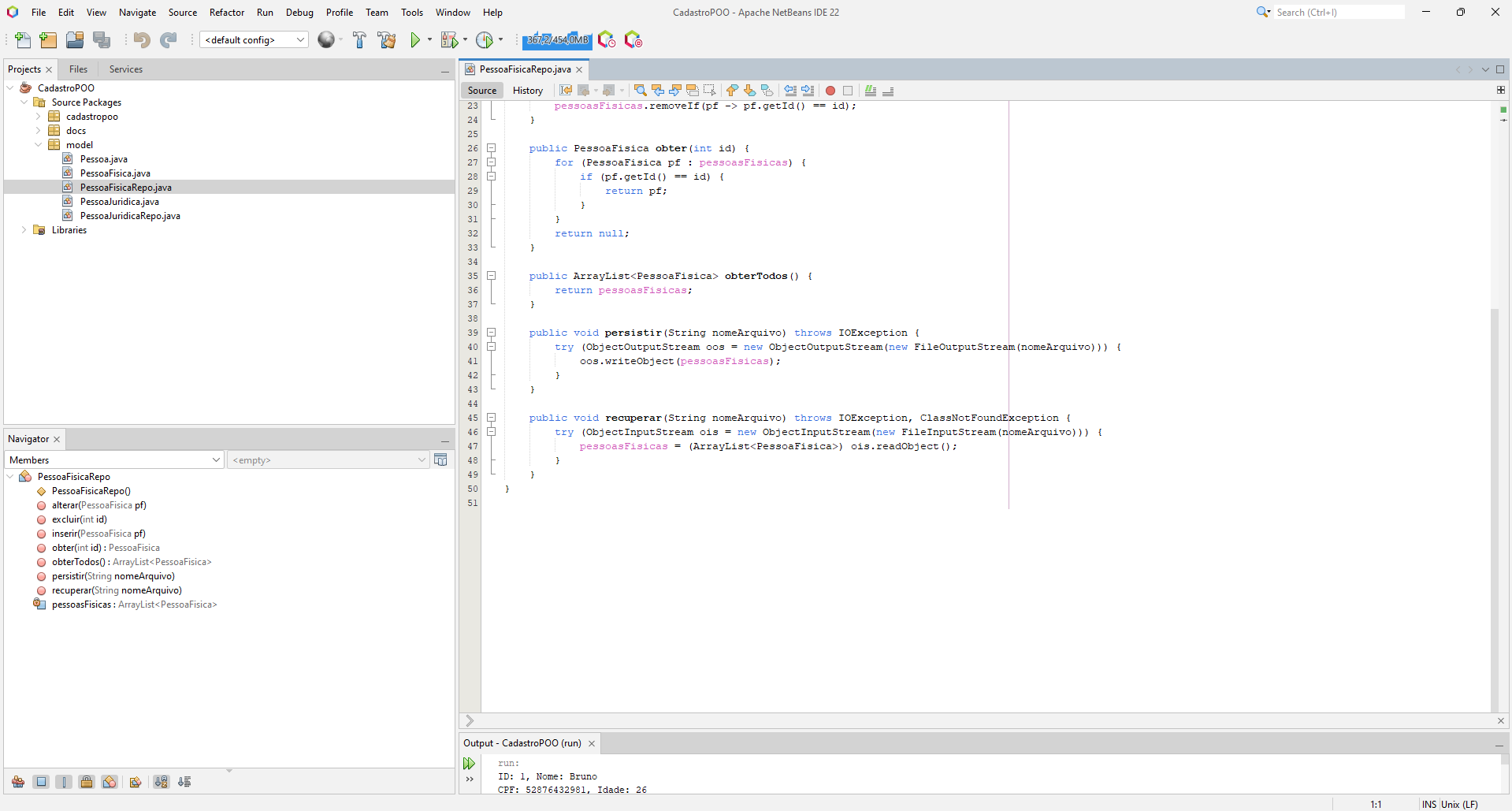


## Pessoa fisica.java

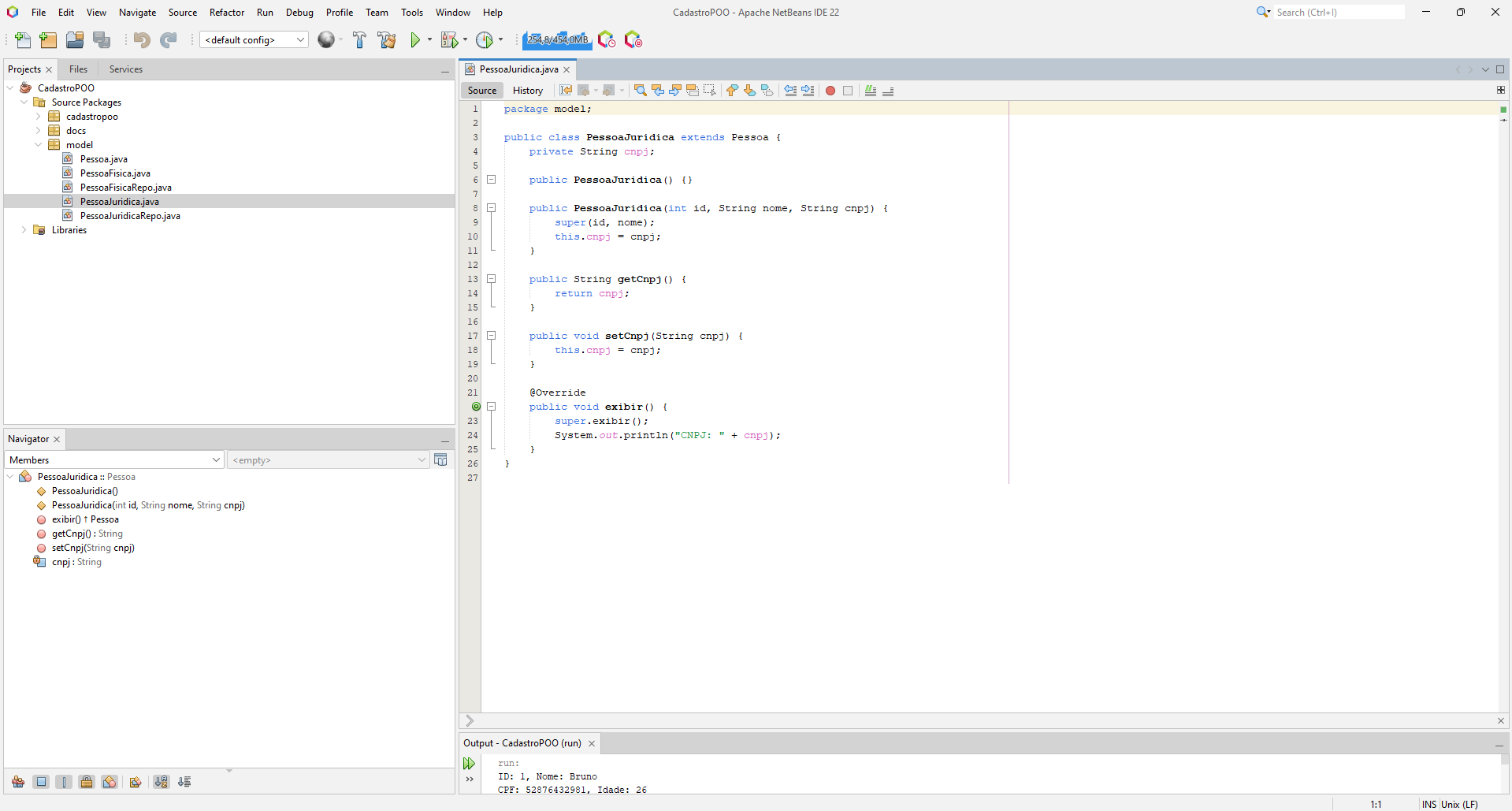


## Pessoa fisica repo.java

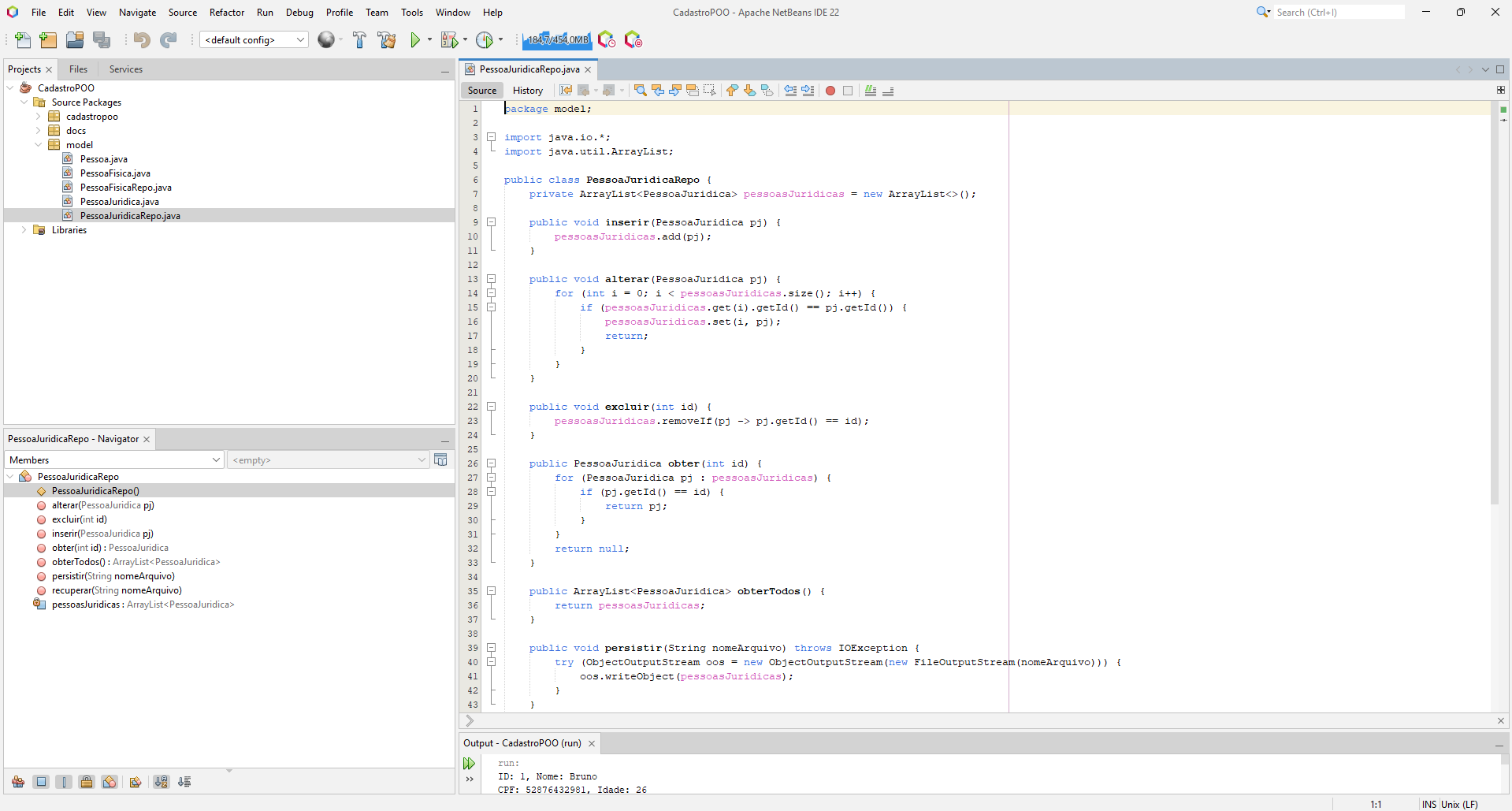


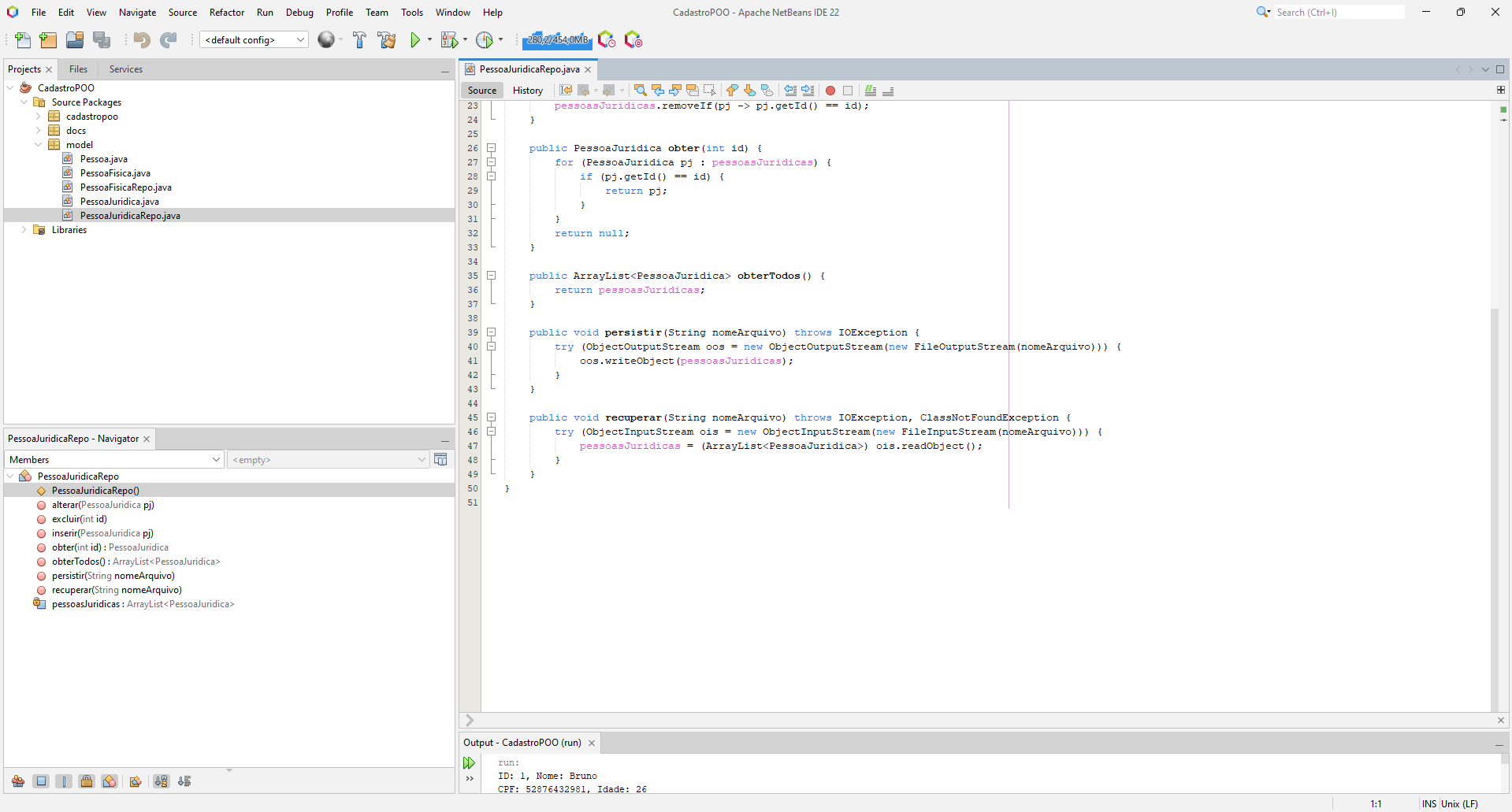


## Pessoa juridica.java

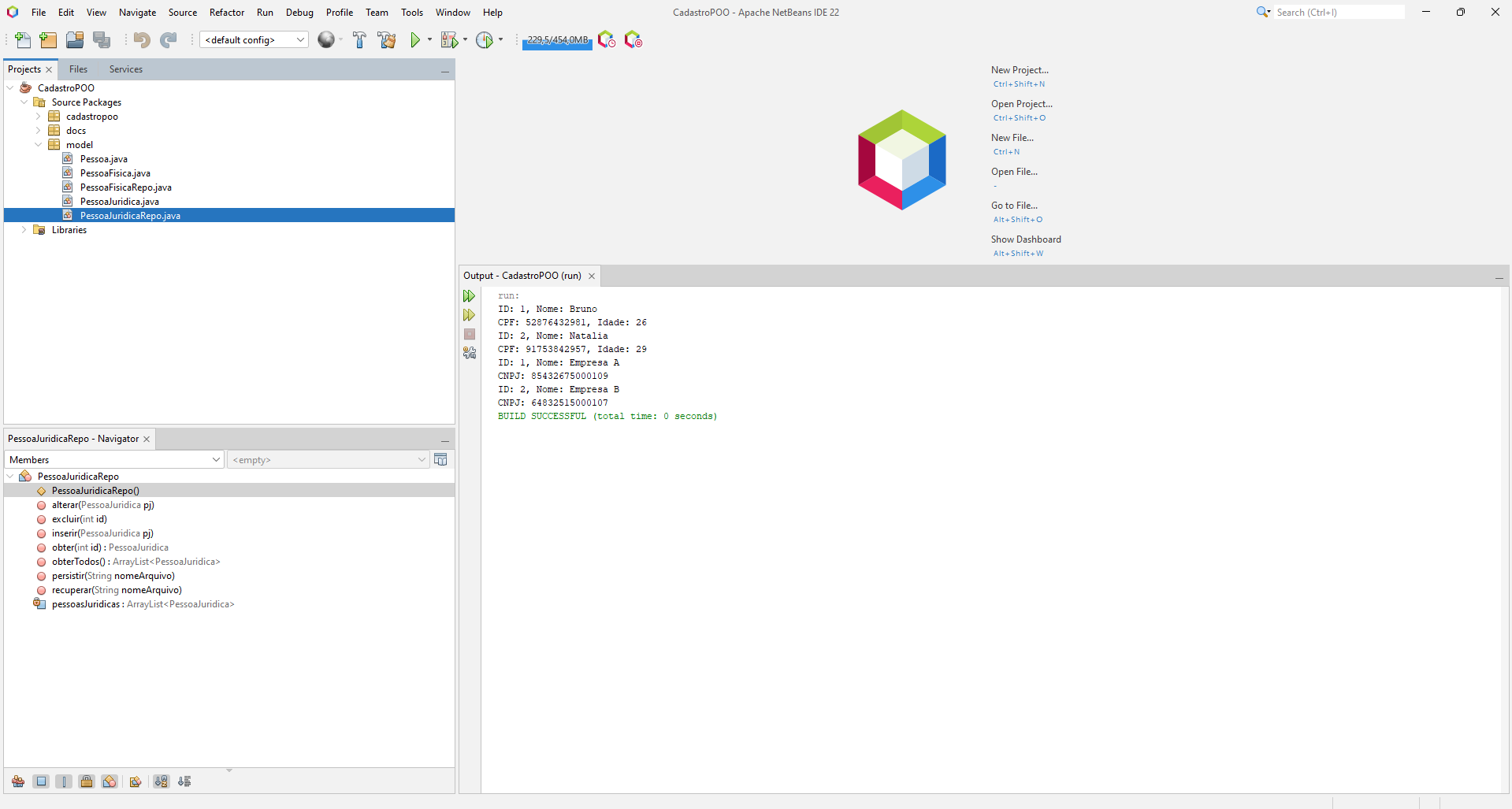


## Pessoa juridica repo.java





# resultados da execução dos códigos



# ANALISE

## 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.

## 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.

## 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.

## 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.

# 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.

# 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/*](https://www.oracle.com/java/technologies/downloads/) *2024.*

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