



# **Estácio**

**FACULDADE ESTÁCIO**

**CÂMPUS MANAUS – AM**

**DESENVOLVIMENTO FULL STACK**

**DISCIPLINA – INICIANDO O CAMINHO PELO JAVA**

**TURMA – 2023.2**

**SEMESTRE – 3**

**MANAUS, JUNHO 2024.**

**DESENVOLVIMENTO FULL STACK**

**DISCIPLINA – INICIANDO O CAMINHO PELO JAVA**

**TURMA – 2023.2**

**SEMESTRE – 3**

**ALUNO – ALEX BARROSO PAZ**

**TUTOR – MARIA MANSO**

**GITHUB - <https://github.com/finntroll89/iniciandoOcaminhoPeloJava1>**

**MANAUS, JUNHO 2024.**

## RESUMO

O projeto CadastroPOO tem como objetivo desenvolver um sistema de cadastro de pessoas físicas e jurídicas utilizando princípios de Programação Orientada a Objetos (POO) na linguagem Java. O projeto foi estruturado no NetBeans como uma aplicação Java do tipo Ant. Foi criado um pacote chamado "model" contendo as classes `Pessoa`, `PessoaFisica`, e `PessoaJuridica`, onde cada uma dessas classes implementa a interface `Serializable`. A classe `Pessoa` possui os campos `id` e `nome`, enquanto `PessoaFisica` e `PessoaJuridica` herdam de `Pessoa` e adicionam os campos `cpf` e `idade`, e `cnpj`, respectivamente.

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

O projeto garante a integridade e a organização dos dados, facilitando a manipulação e persistência das informações de maneira eficiente. O código-fonte do projeto, juntamente com a documentação completa, foi armazenado em um repositório Git para facilitar o acesso e a revisão.

Palavras-chave: Programação Orientada a Objetos (POO), Java Application, Modelagem de Dados, Serialização, Gerenciamento de Dados, Persistência de Dados, Teste de Funcionalidades, Armazenamento em Git, Eficiência e Organização.

## SUMÁRIO

### 1 INTRODUÇÃO

4

1.1 CADASTROPOO	5
1.2 OBJETIVO DA PRÁTICA	5
1.3 O QUE FAZER	5
1.4 COMO FAZER	5

### 2 CADASTROPOO

6

2.1 CODIGO CADASTROPOO	6
------------------------	---

### 3 MODEL

7

3.1 PESSOA.JAVA	8
3.2 PESSOA FISICA.JAVA	9
3.3 PESSOA FISICA REPO.JAVA	10
3.4 PESSOA JURIDICA.JAVA	12
3.5 PESSOA JURIDICA REPO.JAVA	13

### 4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS

14

### 5 ANALISE

14

5.1 VANTAGENS E DESVANTAGENS DO USO DE HERANÇA	14
5.2 POR QUE A INTERFACE SERIALIZABLE É NECESSÁRIA AO EFETUAR PERSISTÊNCIA EM ARQUIVOS BINÁRIOS?	15

5.3 COMO O PARADIGMA FUNCIONAL É UTILIZADO PELA API STREAM NO JAVA?	15
---	----

5.4 PADRÃO DE DESENVOLVIMENTO ADOPTADO NA PERSISTÊNCIA DE DADOS EM ARQUIVOS	15
---	----

## **6 CONCLUSÃO**

**17**

## **7 REFERÊNCIAS**

**18**

# **1 INTRODUÇÃO**

## **1.1 CADASTROPOO**

A prática desenvolvida no projeto CadastroPOO visa aplicar os princípios da Programação Orientada a Objetos (POO) em um sistema de cadastro de pessoas físicas e jurídicas utilizando 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

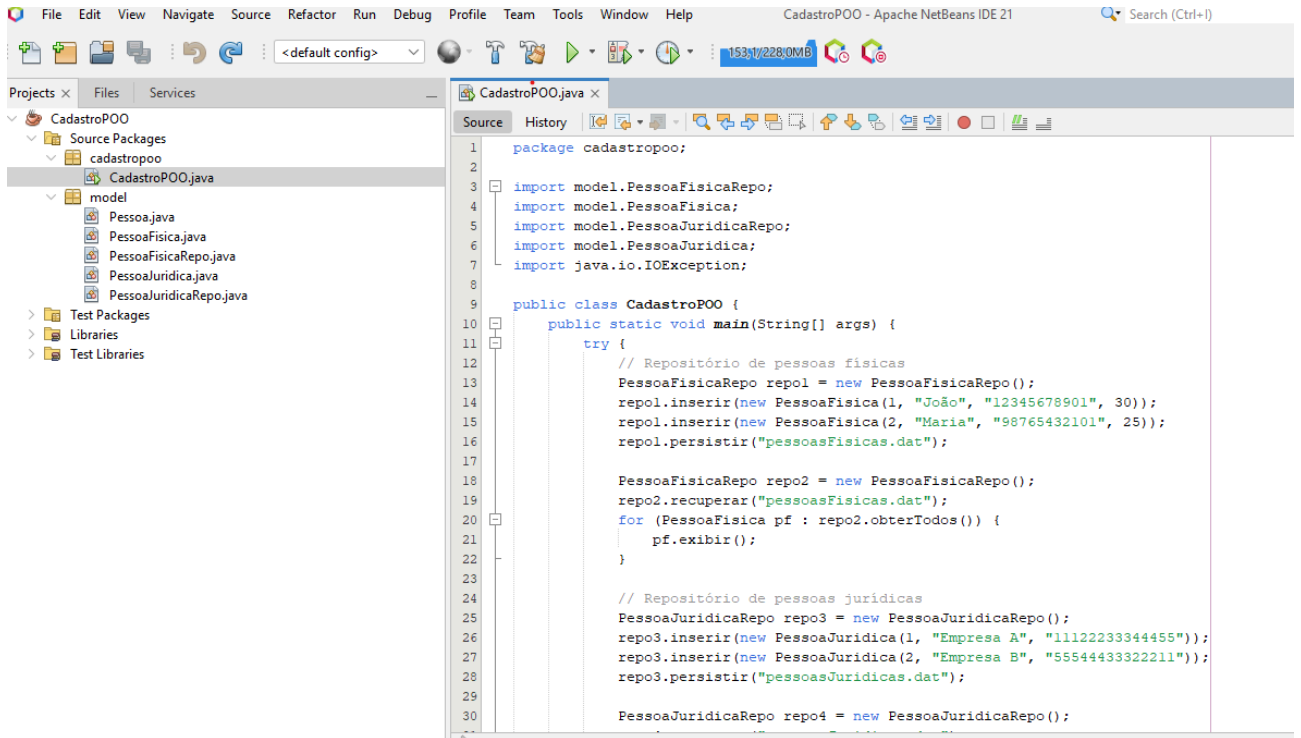
Será desenvolvido um conjunto de classes no pacote "model", incluindo **Pessoa**, **PessoaFisica** e **PessoaJuridica**, cada uma implementando a interface **Serializable** para permitir a persistência em arquivos. Adicionalmente, serão criados os gerenciadores **PessoaFisicaRepo** e **PessoaJuridicaRepo**,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

### 1.4 COMO FAZER

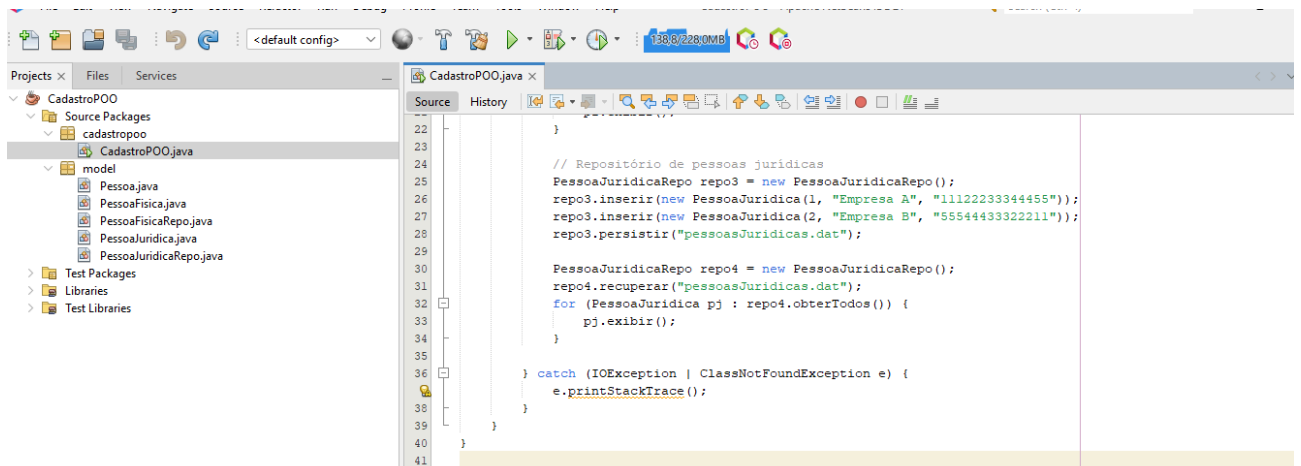
Inicialmente, será criado o projeto no NetBeans e estruturado o pacote "model" para as entidades e gerenciadores. As classes **Pessoa**, **PessoaFisica** e **PessoaJuridica** serão implementadas com seus respectivos atributos, métodos construtores, getters, setters e métodos de exibição. Em seguida,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

## 2 CADASTROPOO

## 2.1 CODIGO CADASTROPOO



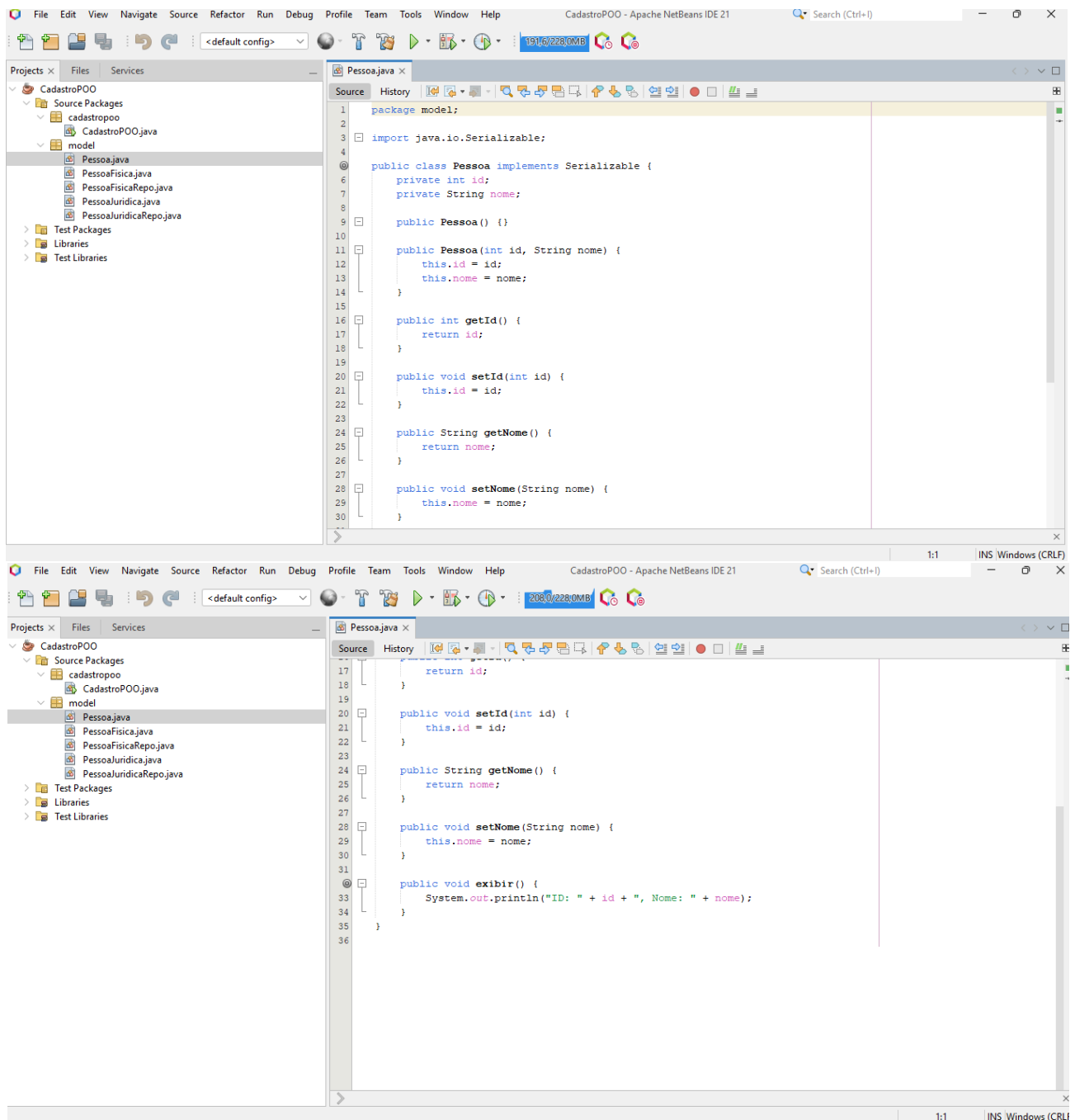
```
1 package cadastrapoo;
2
3 import model.PessoaFisicaRepo;
4 import model.PessoaFisica;
5 import model.PessoaJuridicaRepo;
6 import model.PessoaJuridica;
7 import java.io.IOException;
8
9 public class CadastroPOO {
10     public static void main(String[] args) {
11         try {
12             // Repositório de pessoas físicas
13             PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
14             repo1.inserir(new PessoaFisica(1, "João", "12345678901", 30));
15             repo1.inserir(new PessoaFisica(2, "Maria", "98765432101", 25));
16             repo1.persistir("pessoasFisicas.dat");
17
18             PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
19             repo2.recuperar("pessoasFisicas.dat");
20             for (PessoaFisica pf : repo2.obterTodos()) {
21                 pf.exibir();
22             }
23
24             // Repositório de pessoas jurídicas
25             PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
26             repo3.inserir(new PessoaJuridica(1, "Empresa A", "11122233344455"));
27             repo3.inserir(new PessoaJuridica(2, "Empresa B", "55544433322211"));
28             repo3.persistir("pessoasJuridicas.dat");
29
30             PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
```



```
31             repo4.recuperar("pessoasJuridicas.dat");
32             for (PessoaJuridica pj : repo4.obterTodos()) {
33                 pj.exibir();
34             }
35         } catch (IOException | ClassNotFoundException e) {
36             e.printStackTrace();
37         }
38     }
39 }
40
41
```

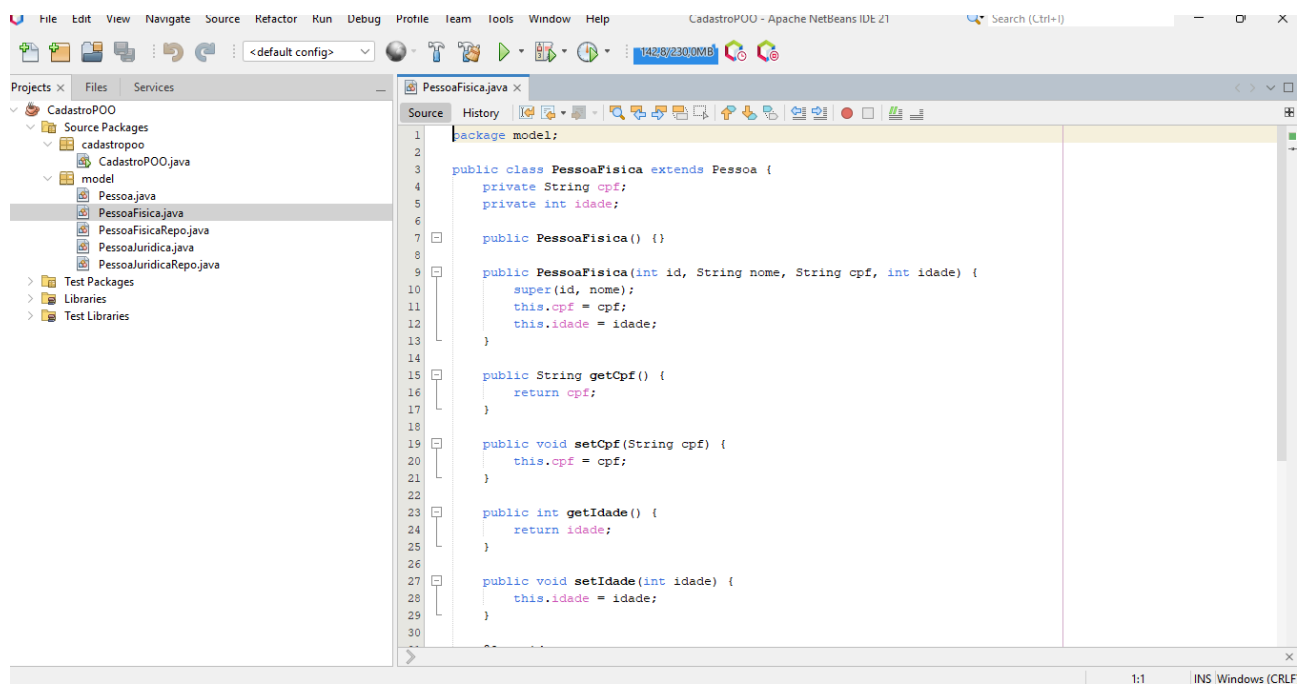
### 3 MODEL

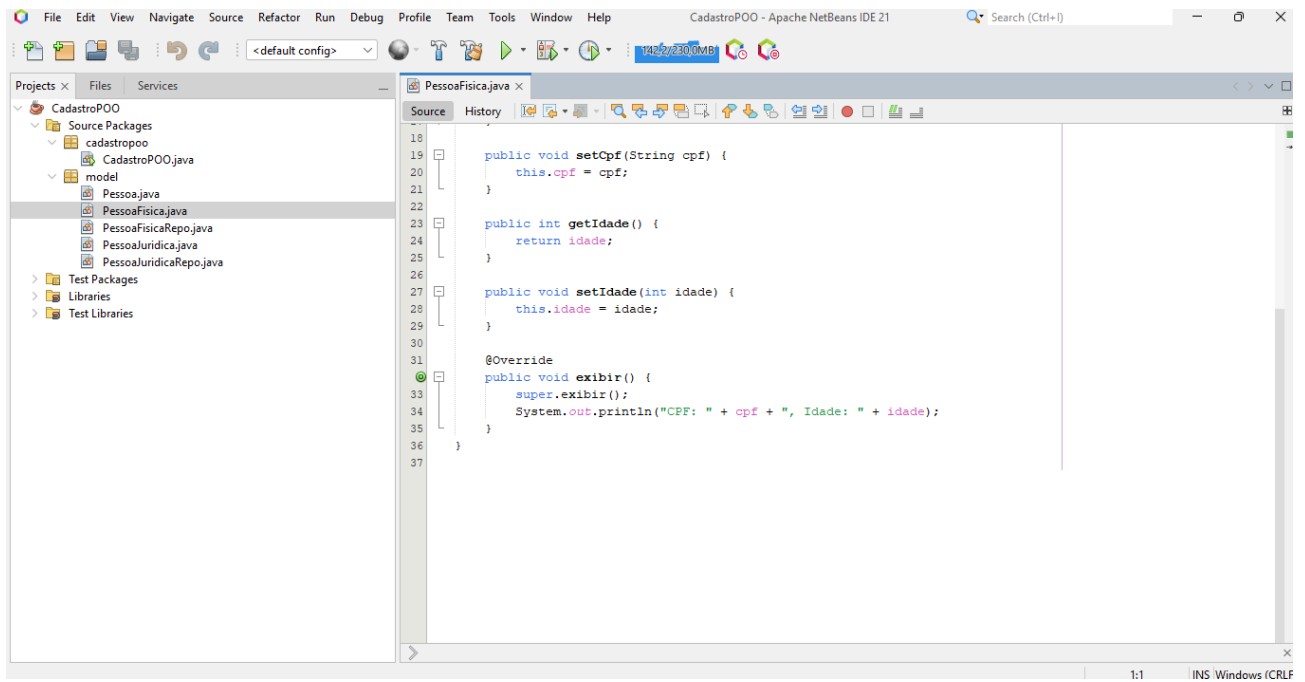
#### 3.1 PESSOA.JAVA



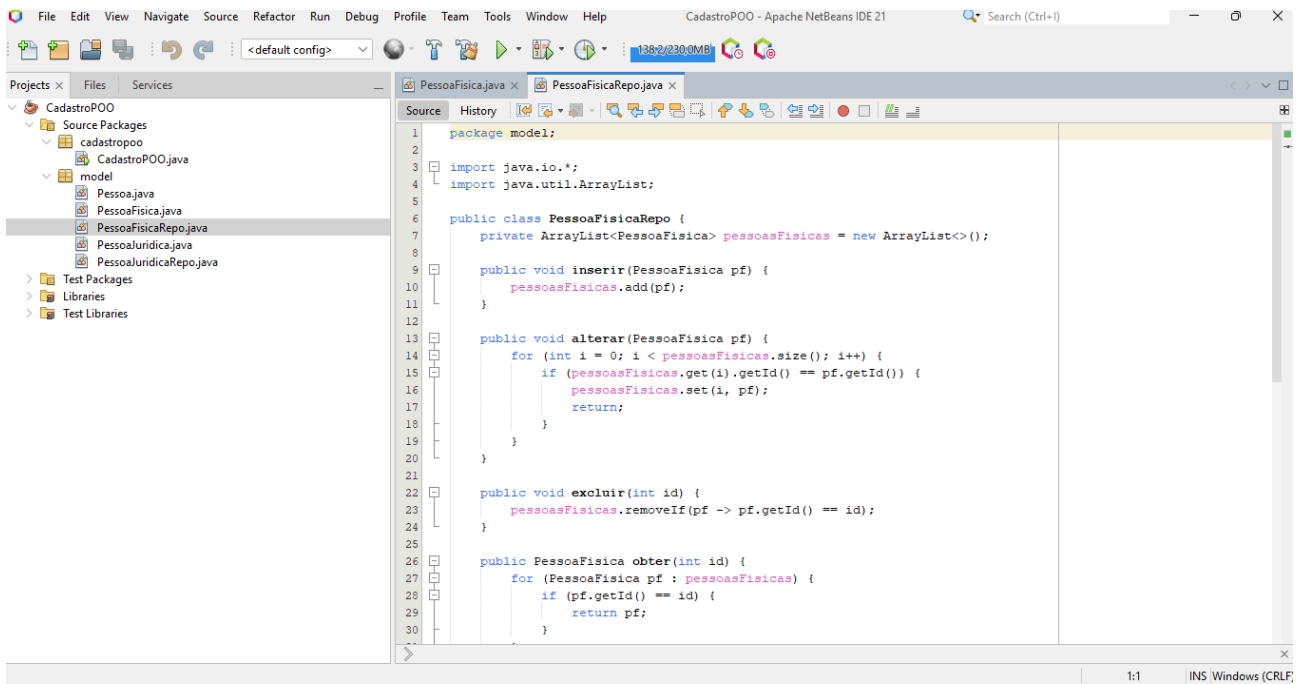


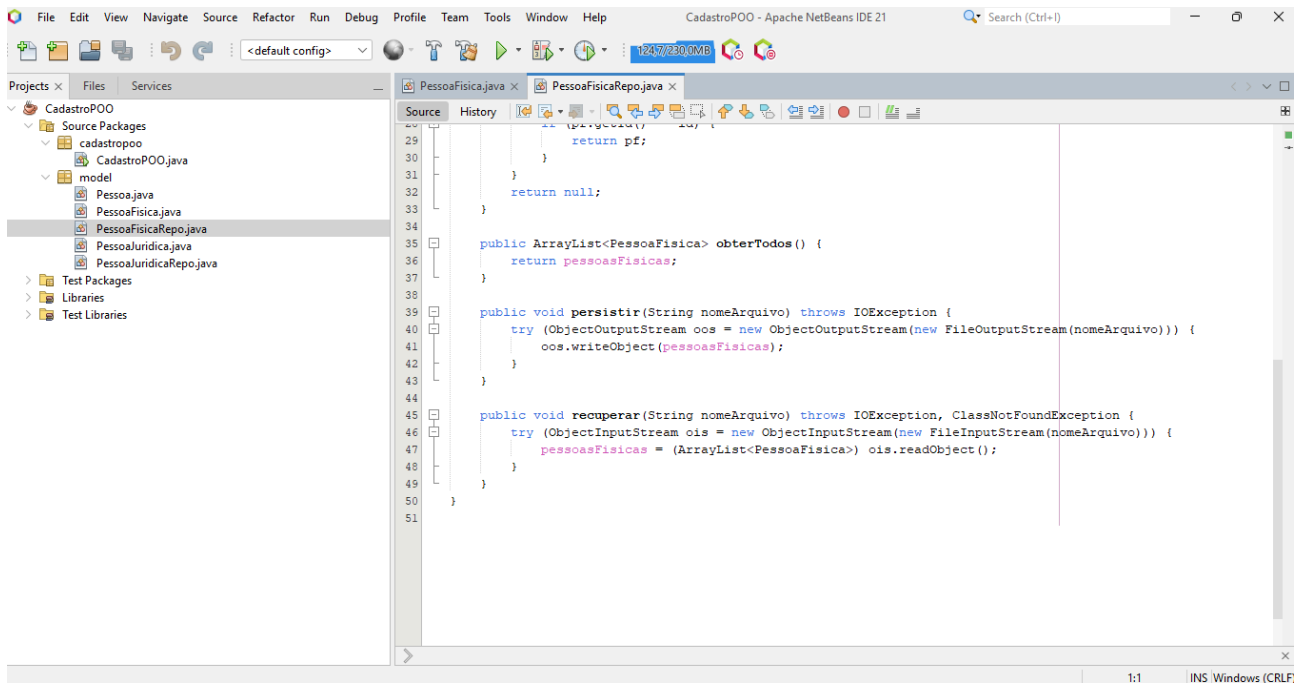
## 3.2 PESSOA FISICA.JAVA



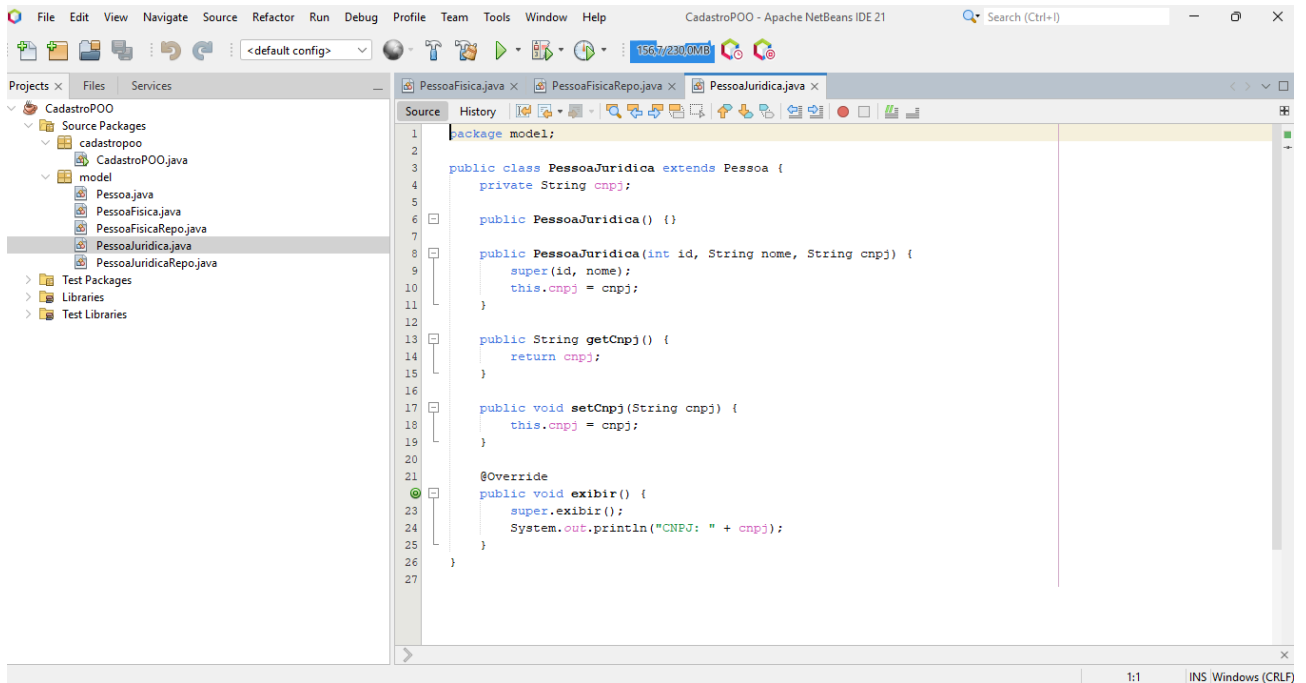


### 3.3 PESSOA FISICA REPO.JAVA

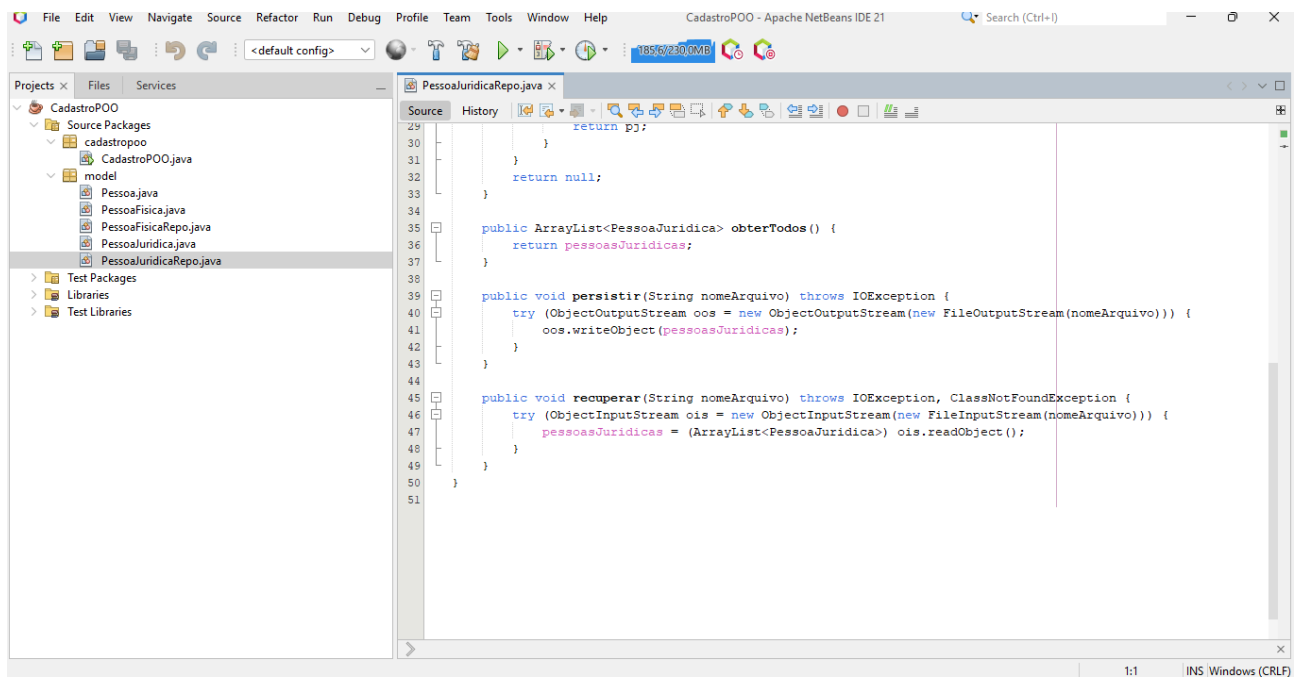
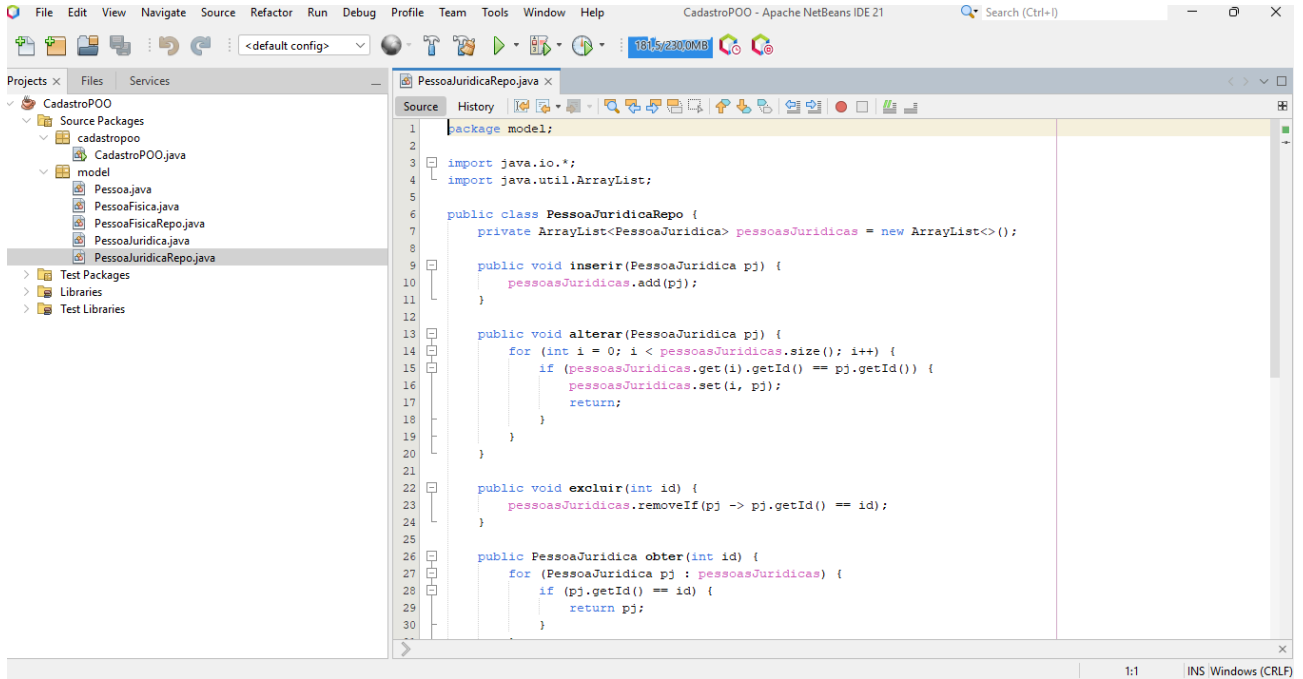




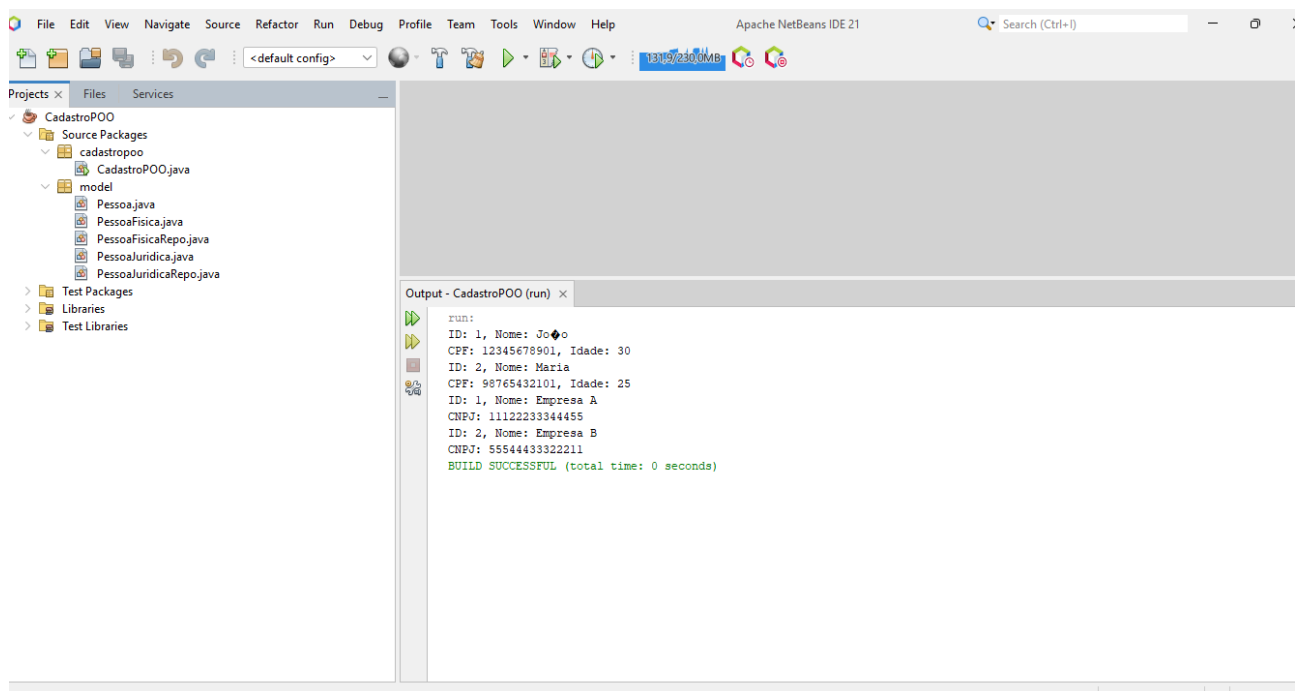
### 3.4 PESSOA JURIDICA.JAVA



### 3.5 PESSOA JURIDICA REPO.JAVA



## 4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS



## 5 ANALISE

### 5.1 VANTAGENS E DESVANTAGENS DO USO DE HERANÇA

O uso de herança em Programação Orientada a Objetos (POO) traz diversas 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 proporcionar 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.



O projeto CadastroPOO 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 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.

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

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

**ORACLE**, *Java Downloads* Acessado em 2024, [www.oracle.com/java/technologies/downloads/](http://www.oracle.com/java/technologies/downloads/) .

**GeeksforGeeks**, *Java Programming Language*, Acessado em 2024. [www.geeksforgeeks.org/java](http://www.geeksforgeeks.org/java).

