



# **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 – JOSYANE SOUZA**

**GITHUB - <https://github.com/finntroll89/JavaTrabalho1Parte2>**

**MANAUS, JUNHO 2024.**

## RESUMO

O código apresentado implementa um sistema de cadastro de pessoas físicas e jurídicas utilizando a linguagem Java. O sistema permite incluir, alterar, excluir, exibir pelo ID, exibir todos, salvar e recuperar dados de pessoas físicas e jurídicas. Para isso, são utilizadas classes de repositório (**PessoaFisicaRepo** e **PessoaJuridicaRepo**) que armazenam os dados e oferecem métodos para manipulá-los. A classe **CadastroPOO** contém o método **main**, que é responsável por exibir o menu de opções e gerenciar a interação com o usuário através da classe **Scanner**.

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

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>5</b>
<b>2 Cadastropoo.....</b>	<b>6</b>
2.1 Codigo CadastroPOO.....	7
<b>3 model.....</b>	<b>9</b>
3.1 Pessoa.java.....	10
3.2 Pessoa fisica.java.....	11
3.3 Pessoa fisica repo.java.....	12
3.4 Pessoa juridica.java.....	13
3.5 Pessoa juridica repo.java.....	14
<b>4 resultados da execução dos códigos.....</b>	<b>15</b>
4.1 incluir.....	15
4.2 Alterar.....	15
4.3 Excluir.....	16
4.4 Exibir pelo ID.....	16
4.5 Exibir todos.....	18
4.6 Salvar dados.....	18
<b>5 ANALISE.....</b>	<b>21</b>
5.1 O que são elementos estáticos e qual o motivo para o método main adotar esse modificador?.....	21
5.2 Para que serve a classe Scanner?.....	21
5.3 Como o uso de classes de repositório impactou na organização do código?.....	21
<b>6 CONCLUSÃO.....</b>	<b>23</b>
<b>7 REFERÊNCIAS.....</b>	<b>24</b>

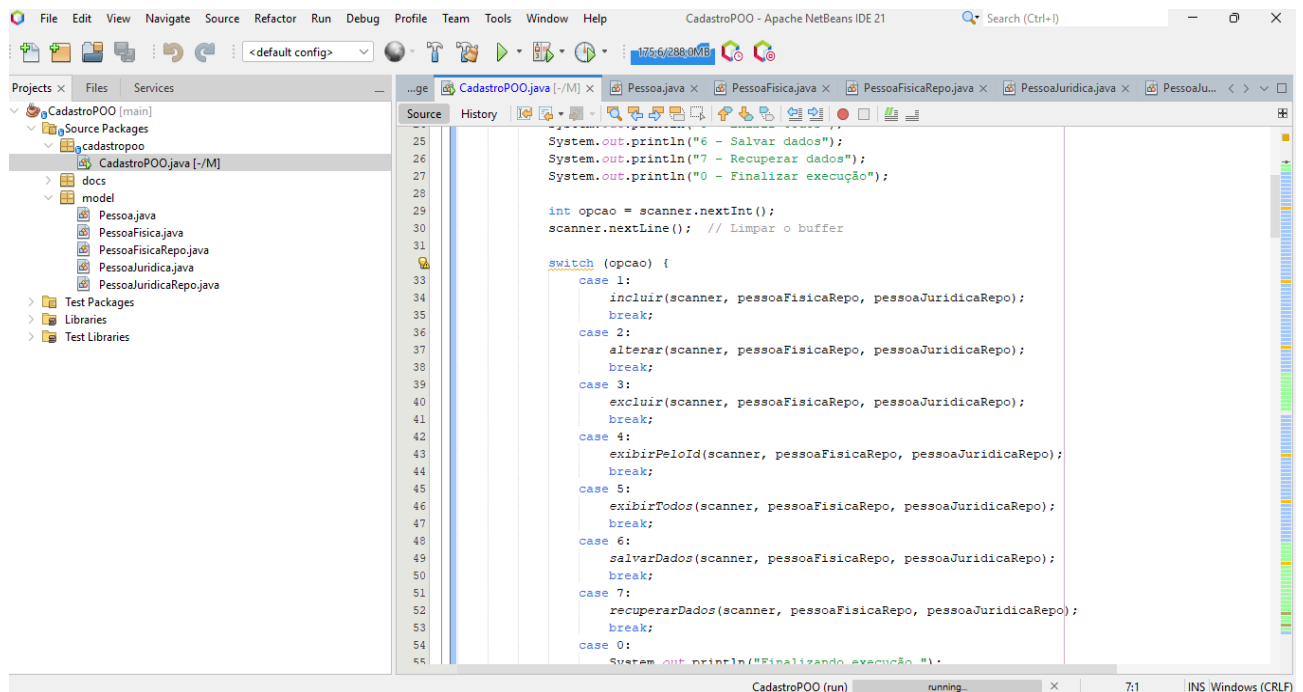
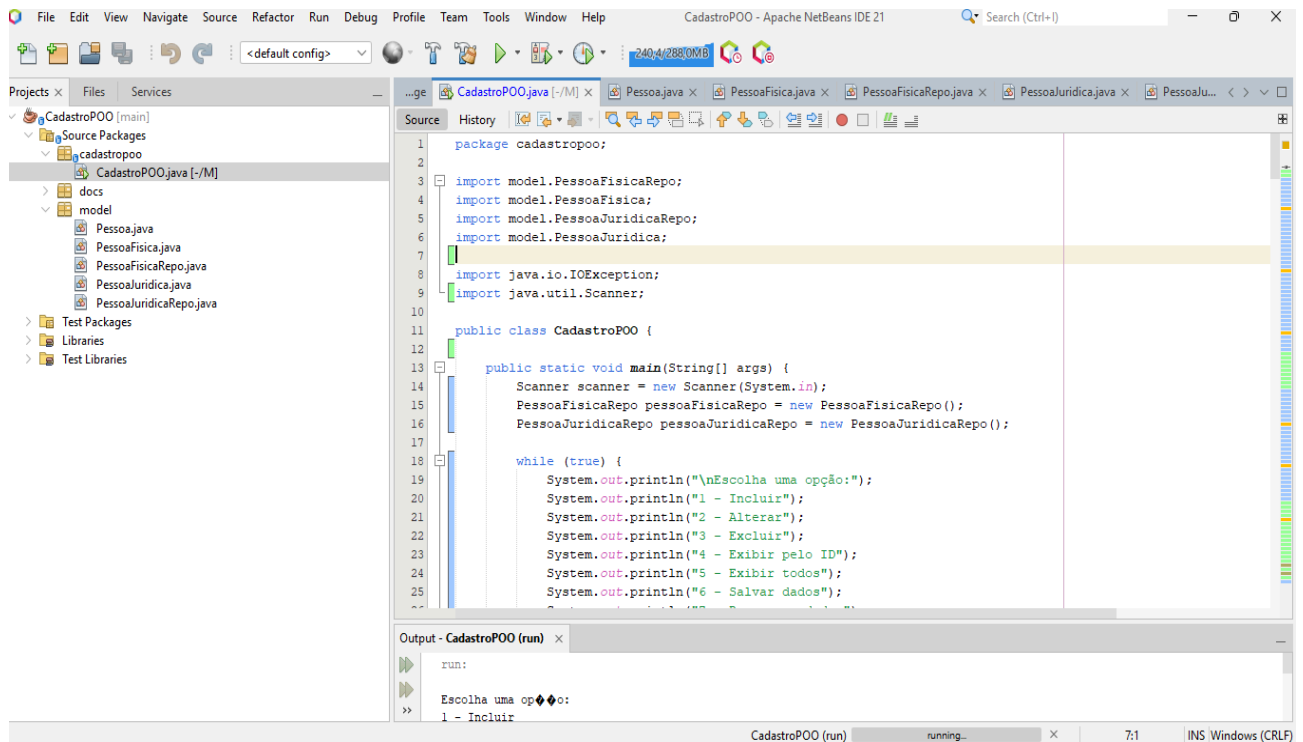
## 1 INTRODUÇÃO

Este trabalho apresenta o desenvolvimento de um sistema de cadastro de pessoas físicas e jurídicas utilizando a linguagem de programação Java. O sistema é projetado para operar com eficiência e clareza, permitindo a inclusão, alteração, exclusão, exibição e persistência de dados. A implementação faz uso de conceitos fundamentais de programação orientada a objetos (POO), como encapsulamento, modularização e reutilização de código, o que facilita a manutenção e expansão do sistema. O ponto de entrada da aplicação é o método `main`, marcado como `static`, permitindo que a JVM (Java Virtual Machine) o invoque diretamente sem a necessidade de instanciar a classe `CadastroPOO`. A interação com o usuário é gerida pela classe `Scanner`, que captura entradas do usuário e facilita a navegação pelas diversas opções do menu.

As operações de manipulação de dados são centralizadas em classes de repositório (`PessoaFisicaRepo` e `PessoaJuridicaRepo`), que encapsulam a lógica de acesso a dados e promovem uma separação clara entre a lógica de negócio e a lógica de interface com o usuário. Esta estrutura não só melhora a legibilidade do código, mas também segue o princípio da responsabilidade única, tornando cada classe responsável por uma única parte do sistema. Este documento também discutirá os elementos estáticos e seu papel no método `main`, a funcionalidade da classe `Scanner`, e o impacto positivo do uso de classes de repositório na organização do código.



## 2.1 CODIGO CADASTROPOO



```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
CadastroPOO - Apache NetBeans IDE 21
Search (Ctrl+I)

rojects x Files Services
CadastroPOO [main]
  Source Packages
  cadastropo
    CadastroPOO.java [-/M]
    docs
    model
      Pessoa.java
      PessoaFisica.java
      PessoaFisicaRepo.java
      PessoaJuridica.java
      PessoaJuridicaRepo.java
  Test Packages
  Libraries
  Test Libraries

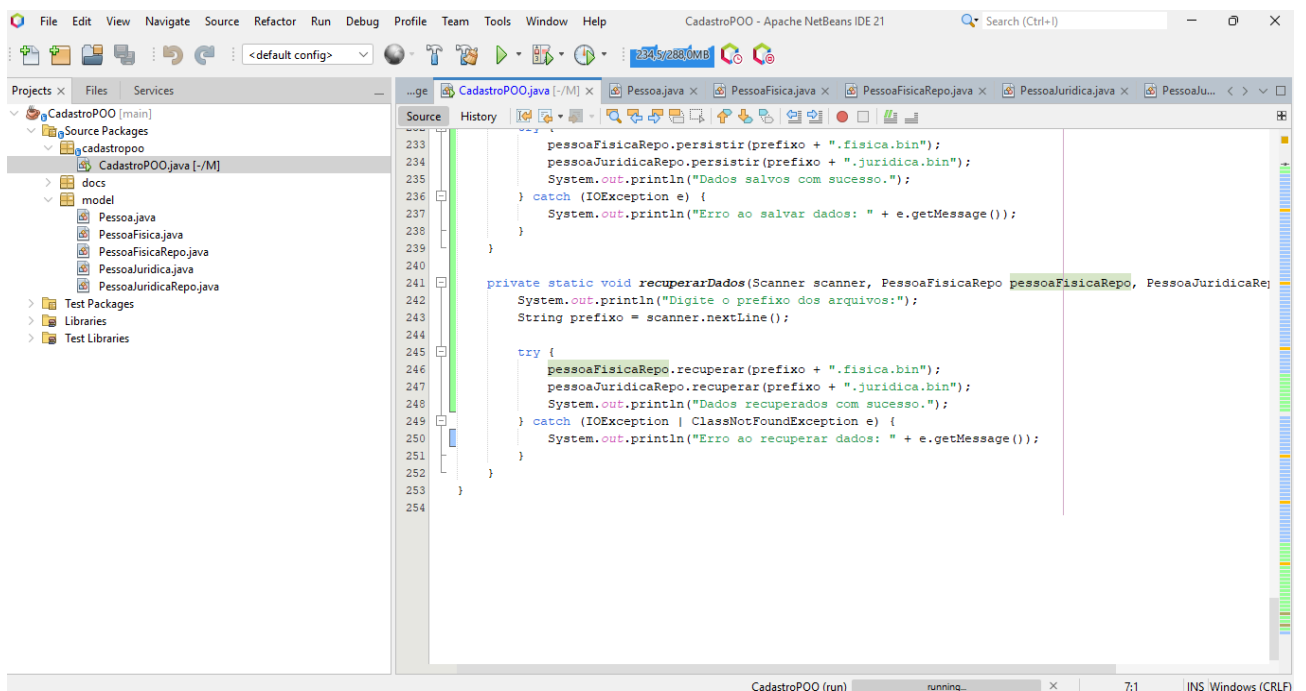
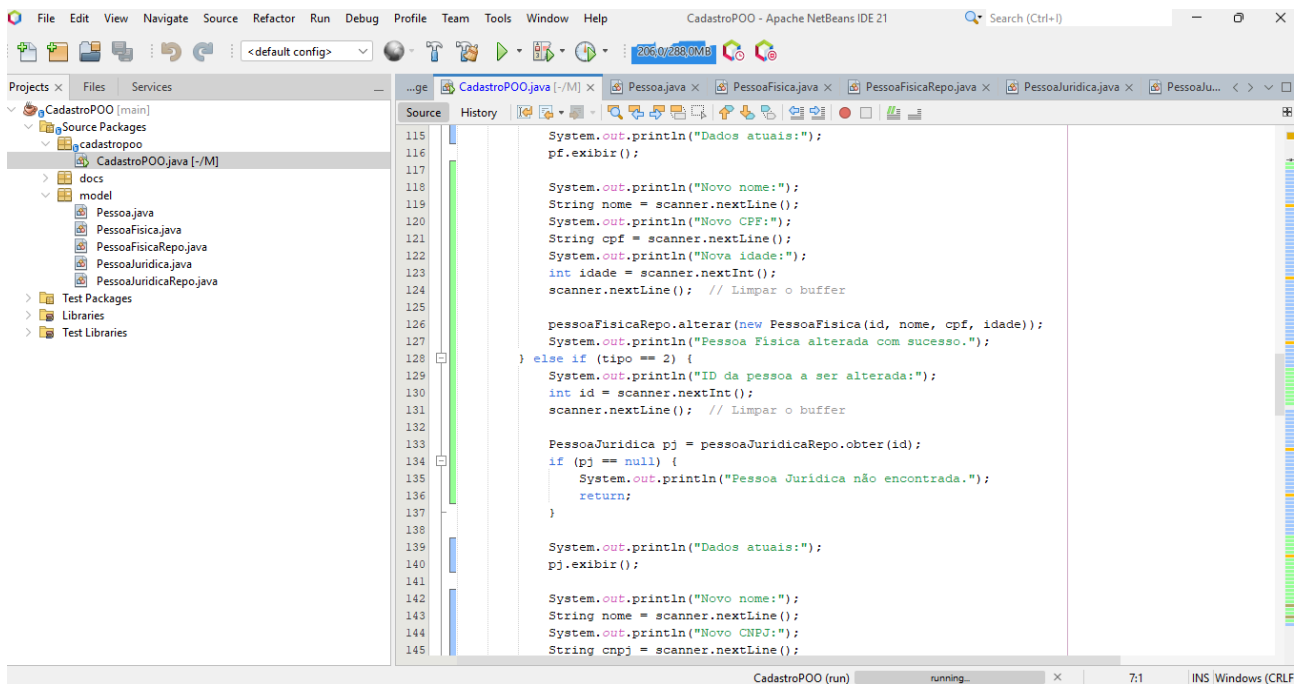
Source History
55      System.out.println("Finalizando execução.");
56      scanner.close();
57      return;
58      default:
59          System.out.println("Opção inválida.");
60      }
61  }
62  }
63
64  private static void incluir(Scanner scanner, PessoaFisicaRepo pessoaFisicaRepo, PessoaJuridicaRepo pess
65  System.out.println("Tipo de pessoa (1 - Fisica, 2 - Juridica):");
66  int tipo = scanner.nextInt();
67  scanner.nextLine(); // Limpar o buffer
68
69  if (tipo == 1) {
70      System.out.println("ID:");
71      int id = scanner.nextInt();
72      scanner.nextLine(); // Limpar o buffer
73      System.out.println("Nome:");
74      String nome = scanner.nextLine();
75      System.out.println("CPF:");
76      String cpf = scanner.nextLine();
77      System.out.println("Idade:");
78      int idade = scanner.nextInt();
79      scanner.nextLine(); // Limpar o buffer
80
81      pessoaFisicaRepo.inserir(new PessoaFisica(id, nome, cpf, idade));
82      System.out.println("Pessoa Fisica incluída com sucesso.");
83  } else if (tipo == 2) {
84      System.out.println("ID:");
85      int id = scanner.nextInt();
```

```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
CadastroPOO - Apache NetBeans IDE 21
Search (Ctrl+I)

rojects x Files Services
CadastroPOO [main]
  Source Packages
  cadastropo
    CadastroPOO.java [-/M]
    docs
    model
      Pessoa.java
      PessoaFisica.java
      PessoaFisicaRepo.java
      PessoaJuridica.java
      PessoaJuridicaRepo.java
  Test Packages
  Libraries
  Test Libraries

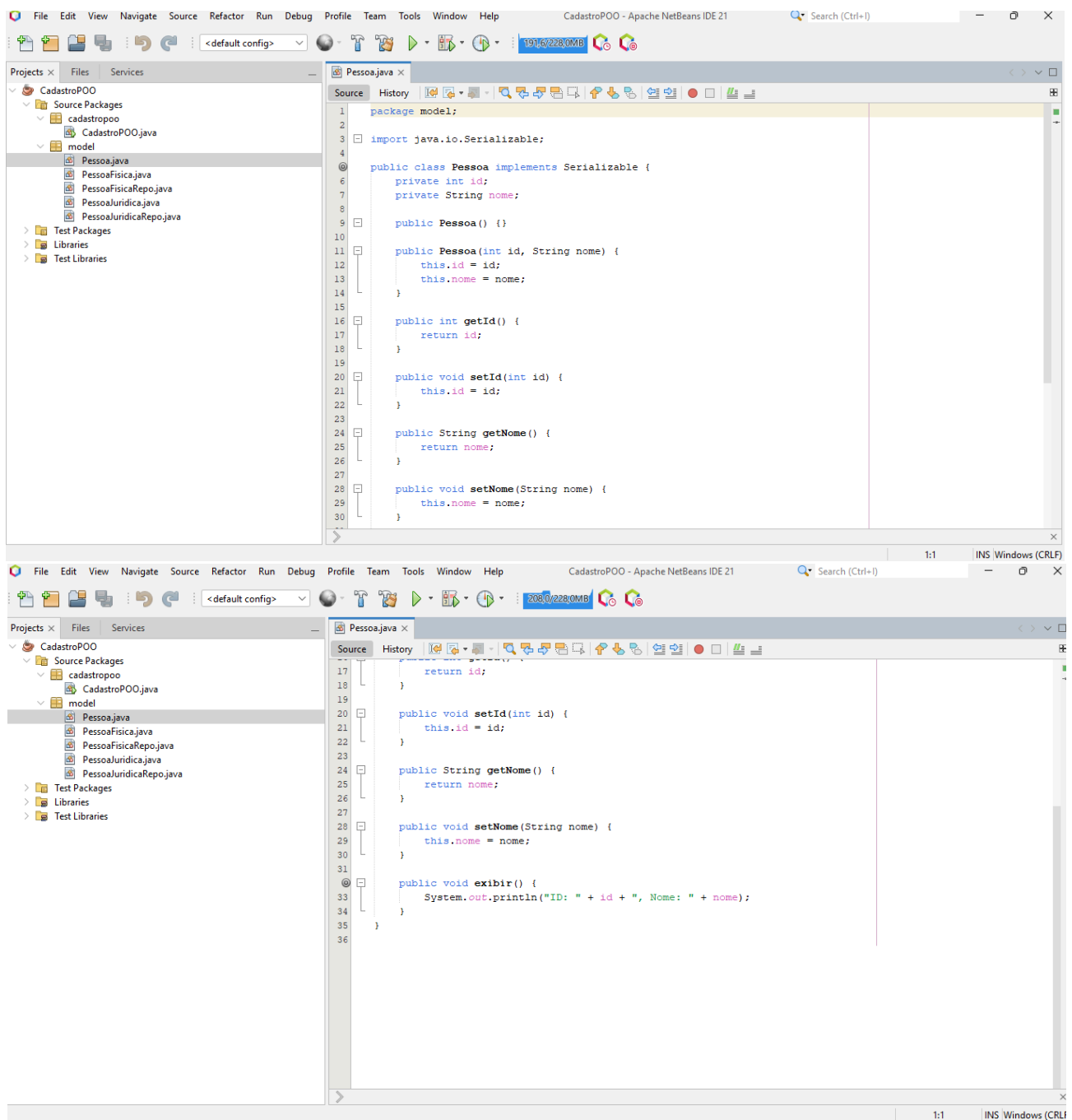
Source History
85      int id = scanner.nextInt();
86      scanner.nextLine(); // Limpar o buffer
87      System.out.println("Nome:");
88      String nome = scanner.nextLine();
89      System.out.println("CNPJ:");
90      String cnpj = scanner.nextLine();
91
92      pessoaJuridicaRepo.inserir(new PessoaJuridica(id, nome, cnpj));
93      System.out.println("Pessoa Juridica incluída com sucesso.");
94  } else {
95      System.out.println("Tipo inválido.");
96  }
97  }
98
99  private static void alterar(Scanner scanner, PessoaFisicaRepo pessoaFisicaRepo, PessoaJuridicaRepo pess
100  System.out.println("Tipo de pessoa (1 - Fisica, 2 - Juridica):");
101  int tipo = scanner.nextInt();
102  scanner.nextLine(); // Limpar o buffer
103
104  if (tipo == 1) {
105      System.out.println("ID da pessoa a ser alterada:");
106      int id = scanner.nextInt();
107      scanner.nextLine(); // Limpar o buffer
108
109      PessoaFisica pf = pessoaFisicaRepo.obter(id);
110      if (pf == null) {
111          System.out.println("Pessoa Fisica não encontrada.");
112          return;
113      }
114
115      System.out.println("Dados atuais:");
```



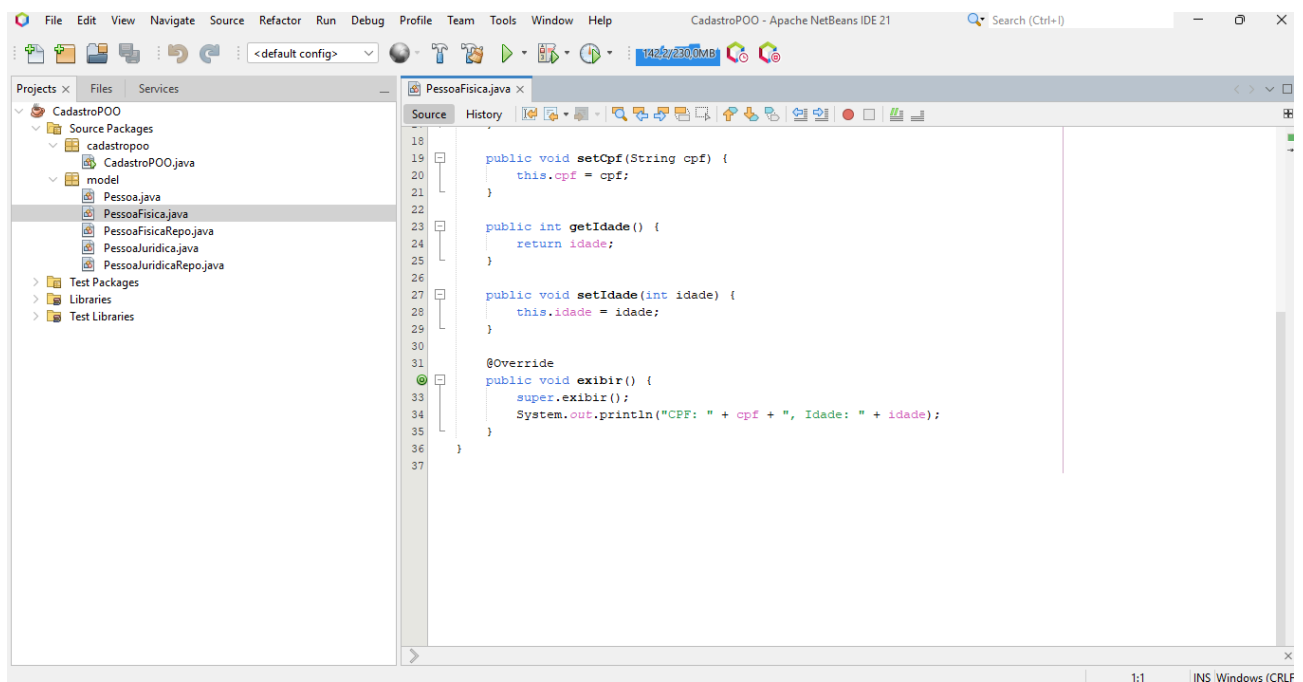
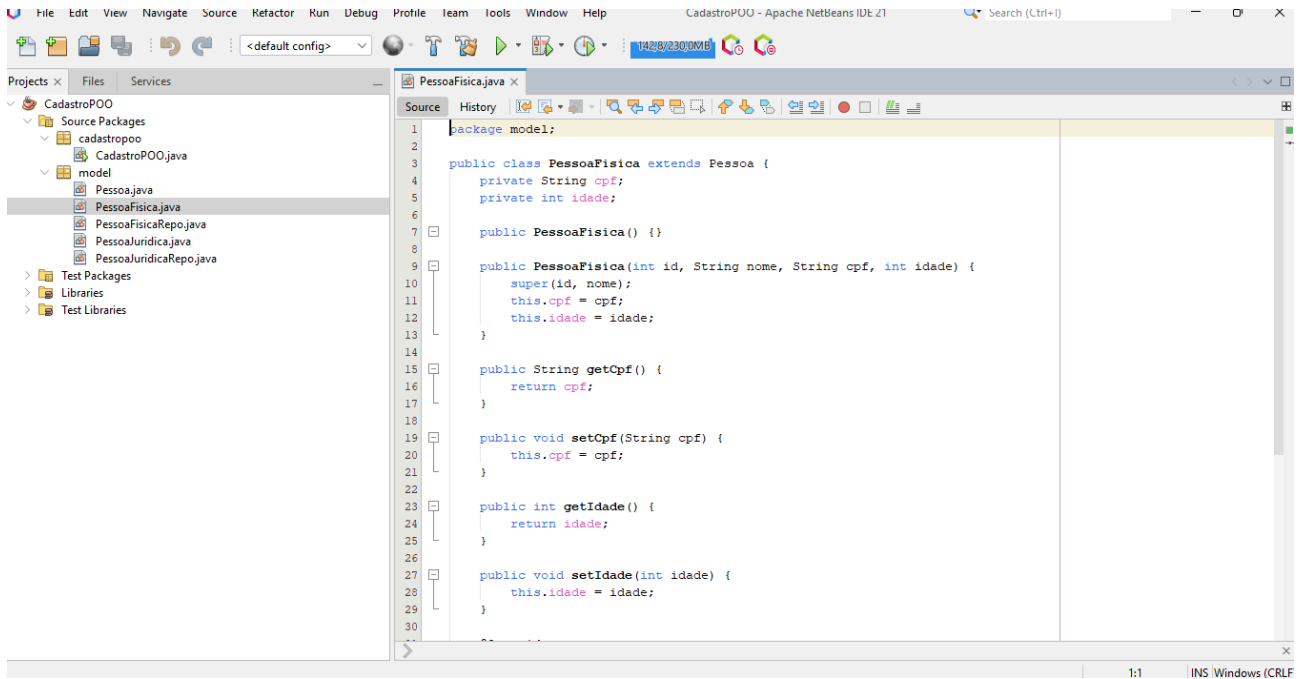


### 3 MODEL

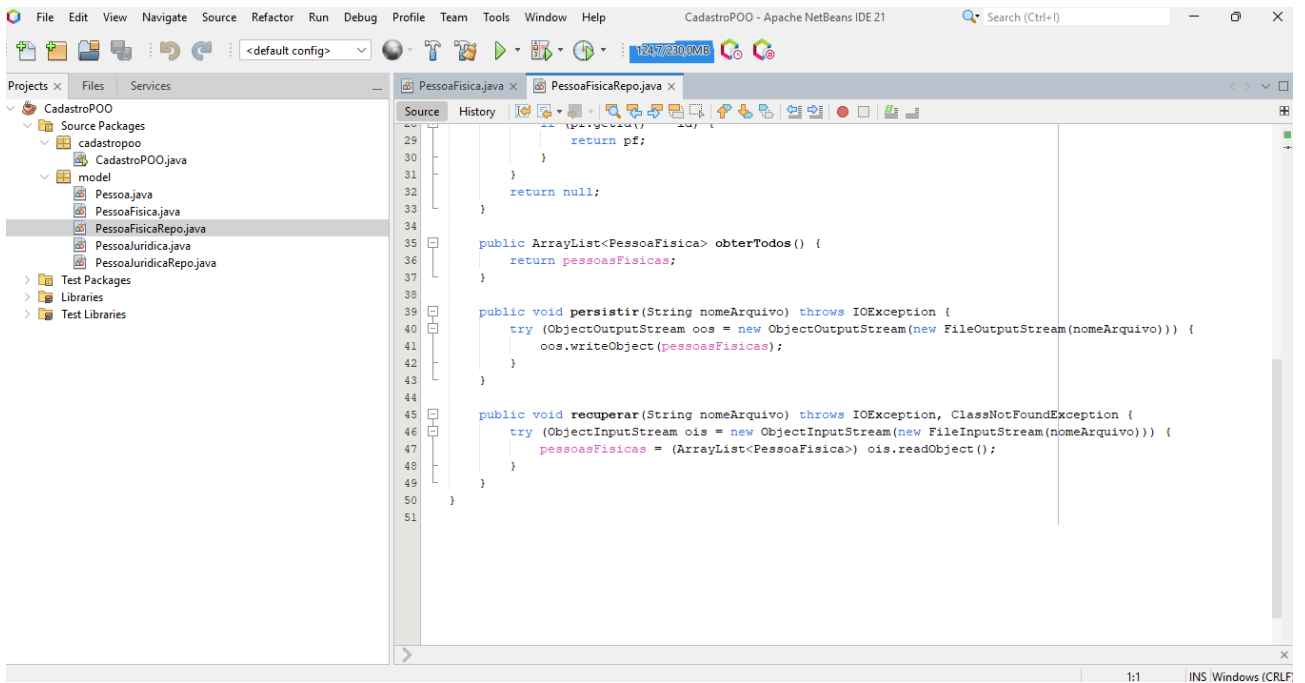
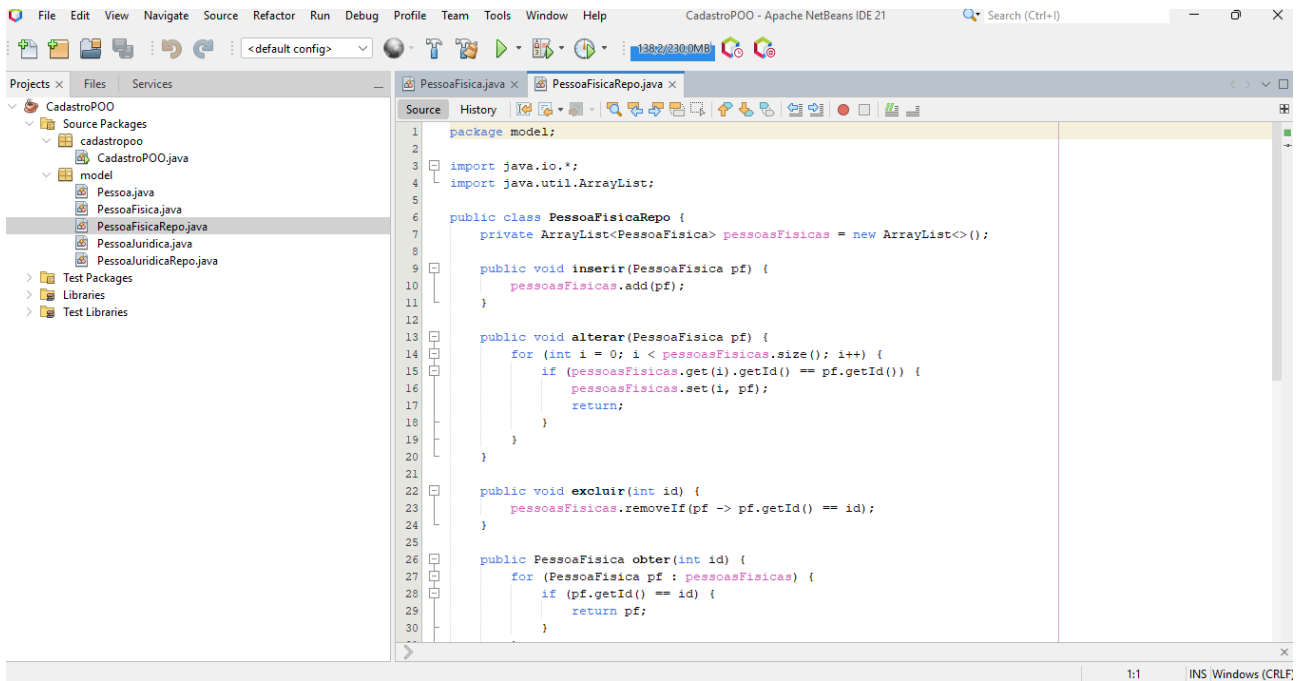
#### 3.1 PESSOA.JAVA



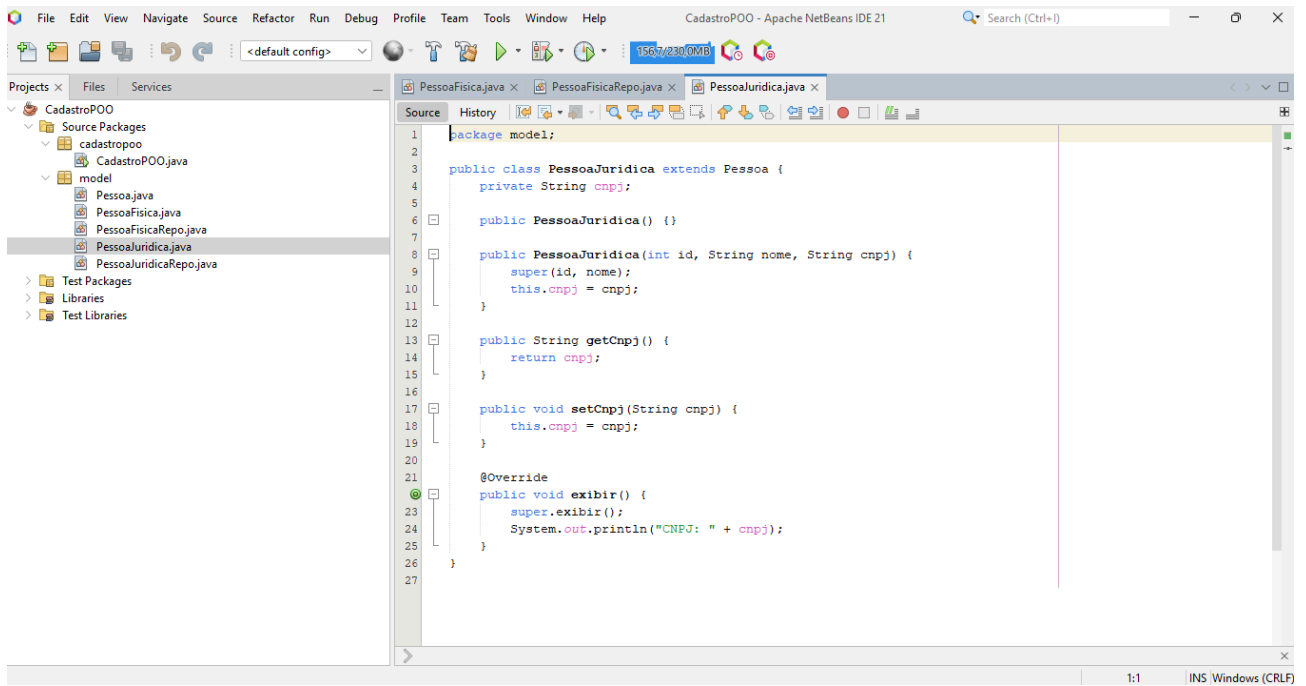
## 3.2 PESSOA FISICA.JAVA



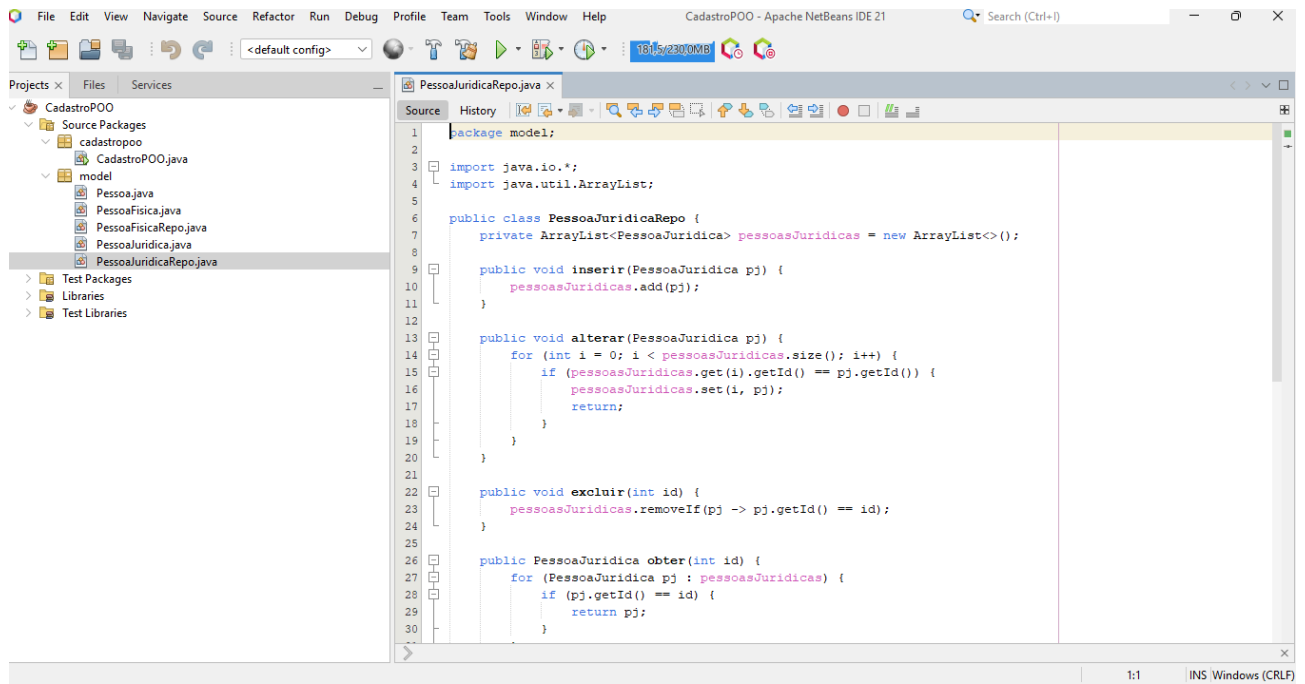
### 3.3 PESSOA FISICA REPO.JAVA



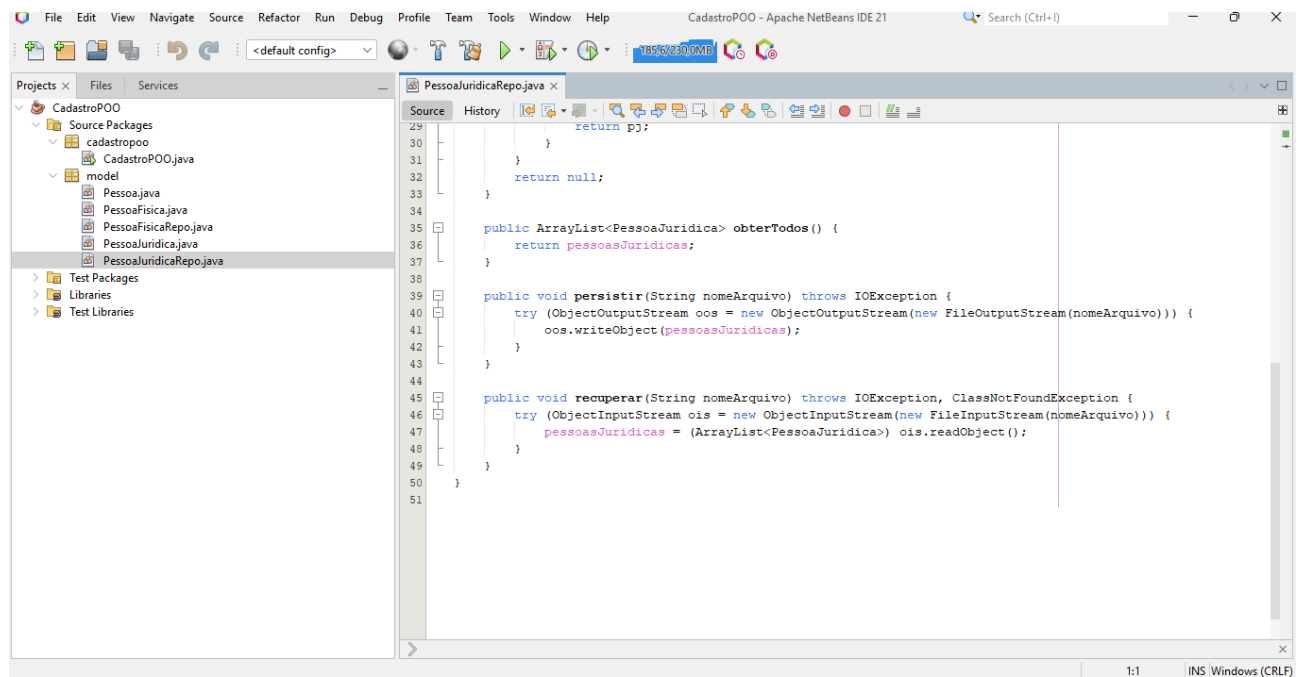
### 3.4 PESSOA JURIDICA.JAVA



### 3.5 PESSOA JURIDICA REPO.JAVA



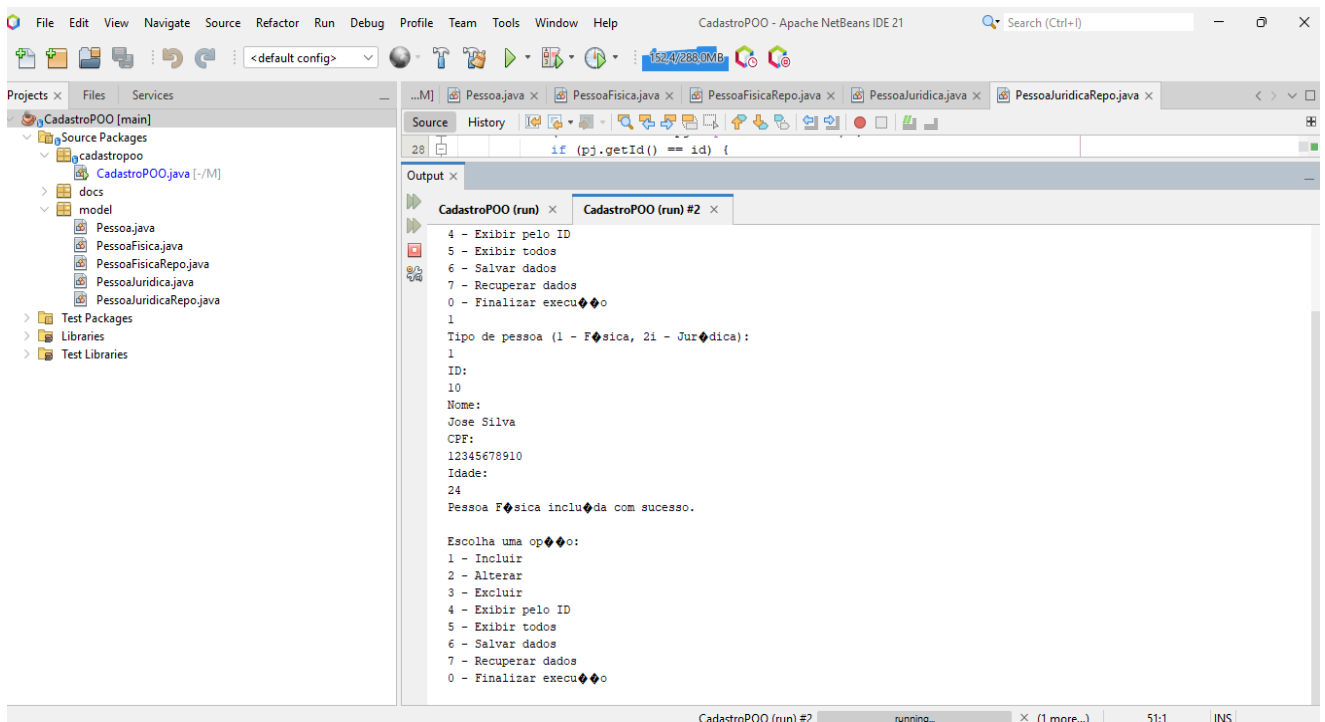
```
1 package model;
2
3 import java.io.*;
4 import java.util.ArrayList;
5
6 public class PessoaJuridicaRepo {
7     private ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
8
9     public void inserir(PessoaJuridica pj) {
10         pessoasJuridicas.add(pj);
11     }
12
13     public void alterar(PessoaJuridica pj) {
14         for (int i = 0; i < pessoasJuridicas.size(); i++) {
15             if (pessoasJuridicas.get(i).getId() == pj.getId()) {
16                 pessoasJuridicas.set(i, pj);
17                 return;
18             }
19         }
20     }
21
22     public void excluir(int id) {
23         pessoasJuridicas.removeIf(pj -> pj.getId() == id);
24     }
25
26     public PessoaJuridica obter(int id) {
27         for (PessoaJuridica pj : pessoasJuridicas) {
28             if (pj.getId() == id) {
29                 return pj;
30             }
31         }
32     }
33 }
```



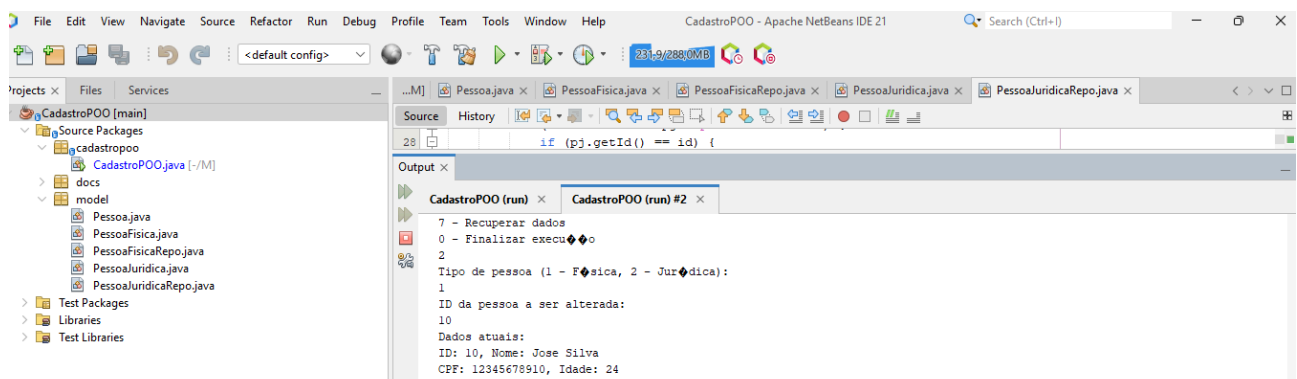
```
29     }
30     }
31     }
32     return null;
33 }
34
35 public ArrayList<PessoaJuridica> obterTodos() {
36     return pessoasJuridicas;
37 }
38
39 public void persistir(String nomeArquivo) throws IOException {
40     try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
41         oos.writeObject(pessoasJuridicas);
42     }
43 }
44
45 public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
46     try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
47         pessoasJuridicas = (ArrayList<PessoaJuridica>) ois.readObject();
48     }
49 }
50 }
51 }
```

## 4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS

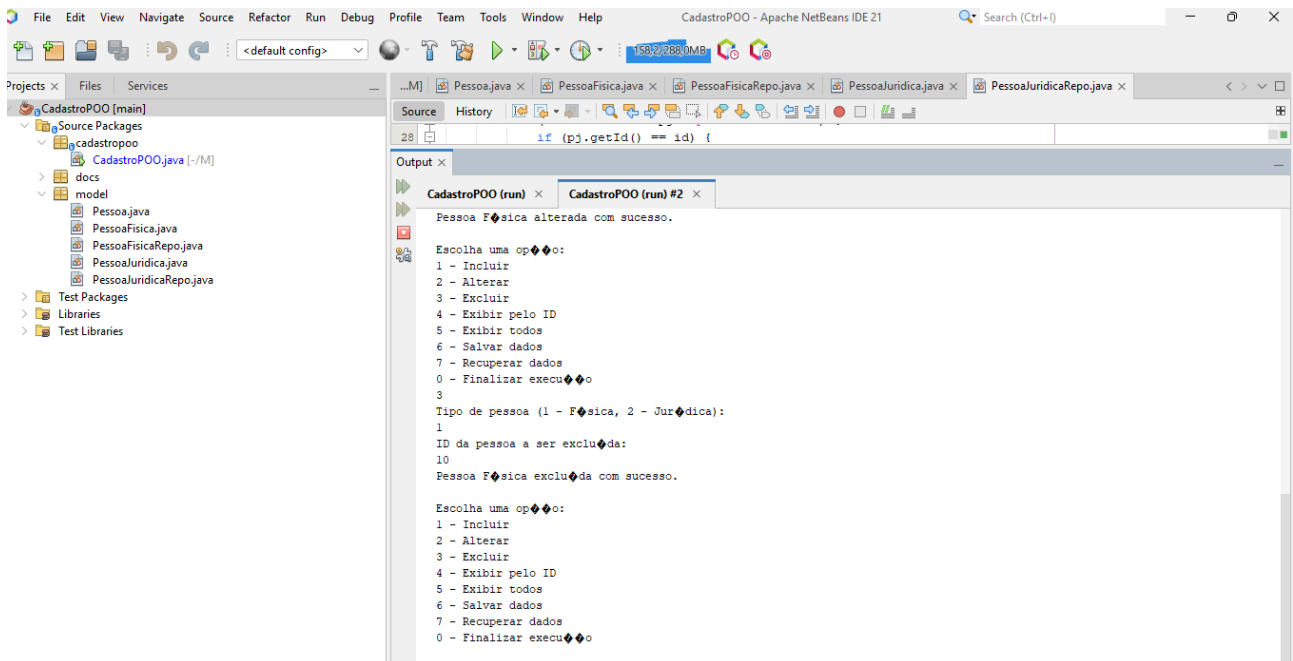
### 4.1 INCLUIR



### 4.2 Alterar

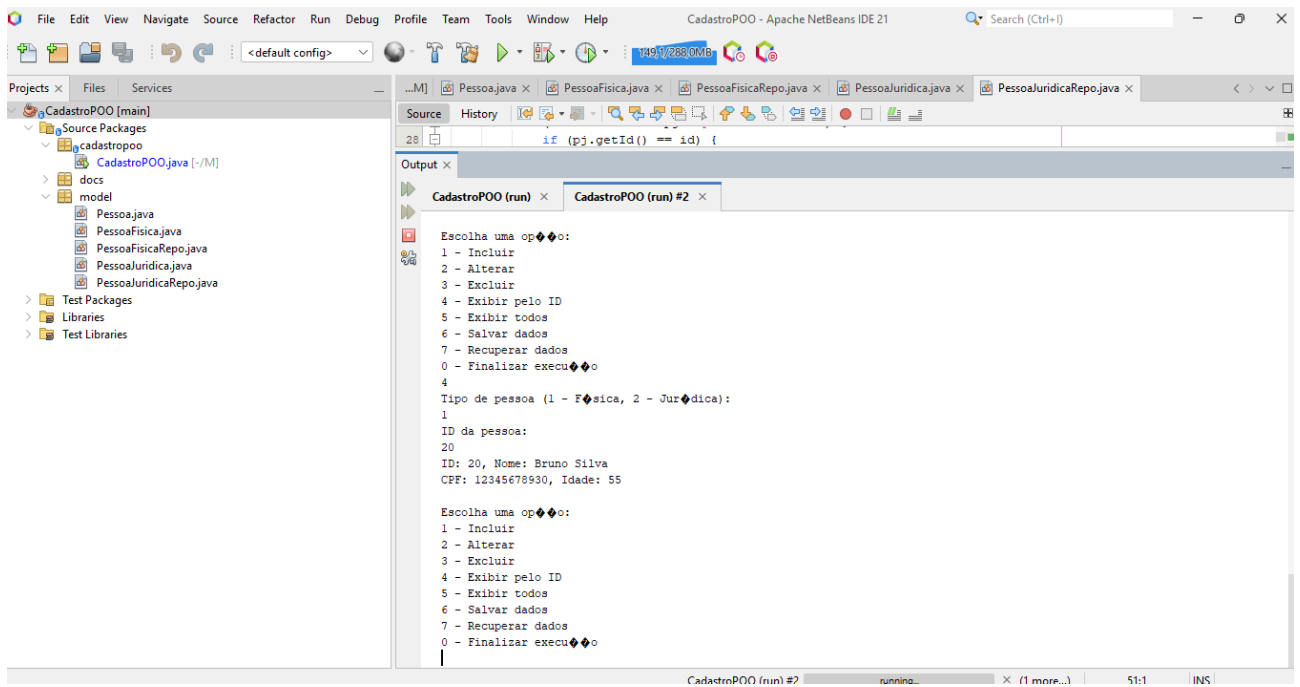


### 4.3 Excluir

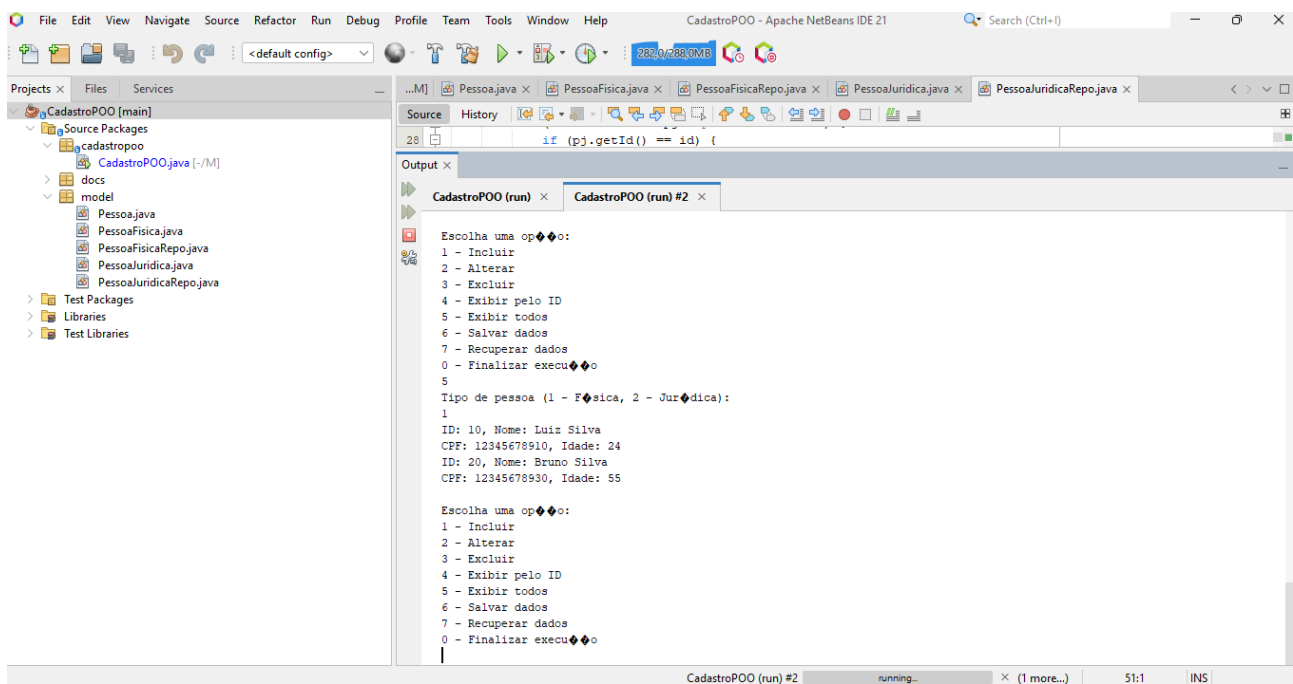


### 4.4 Exibir pelo ID

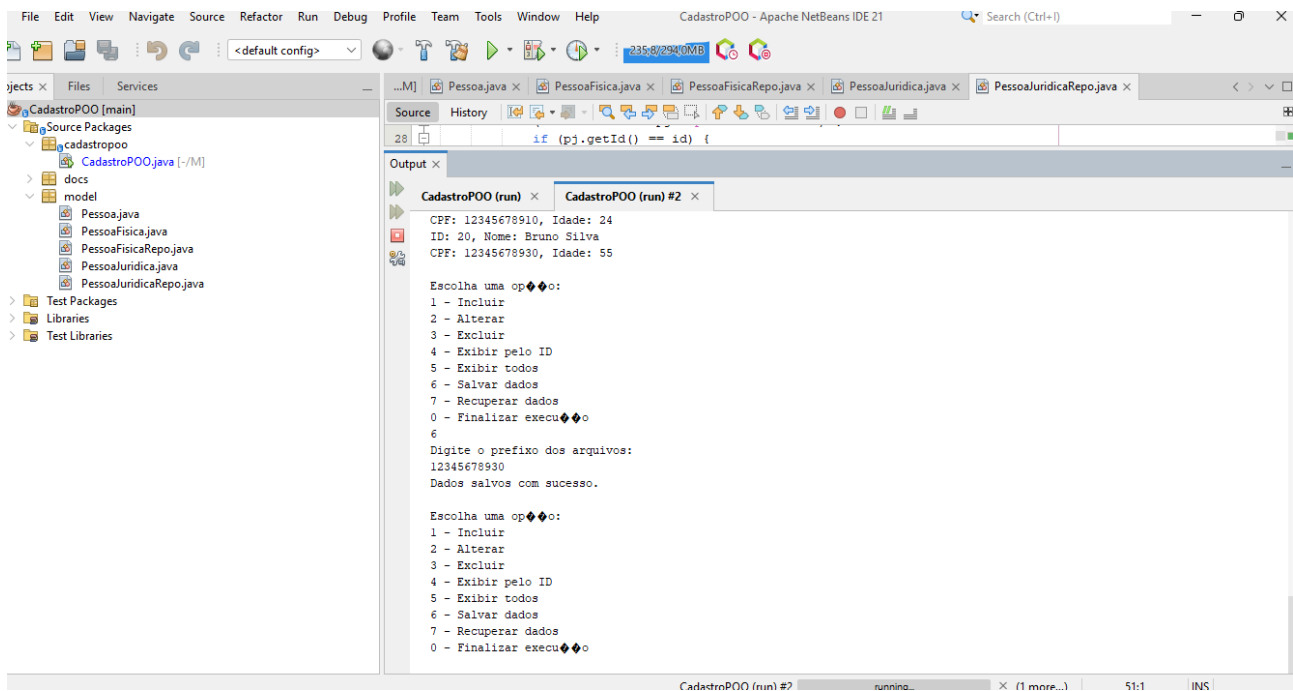




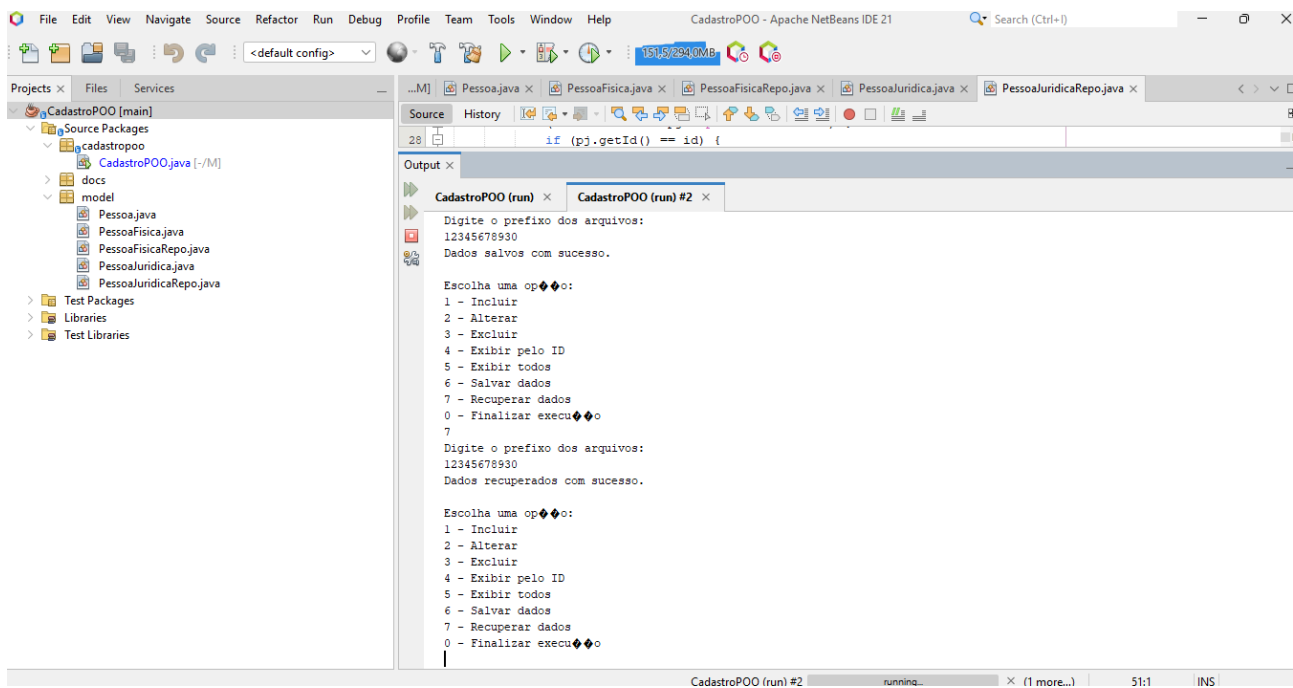
## 4.5 Exibir todos



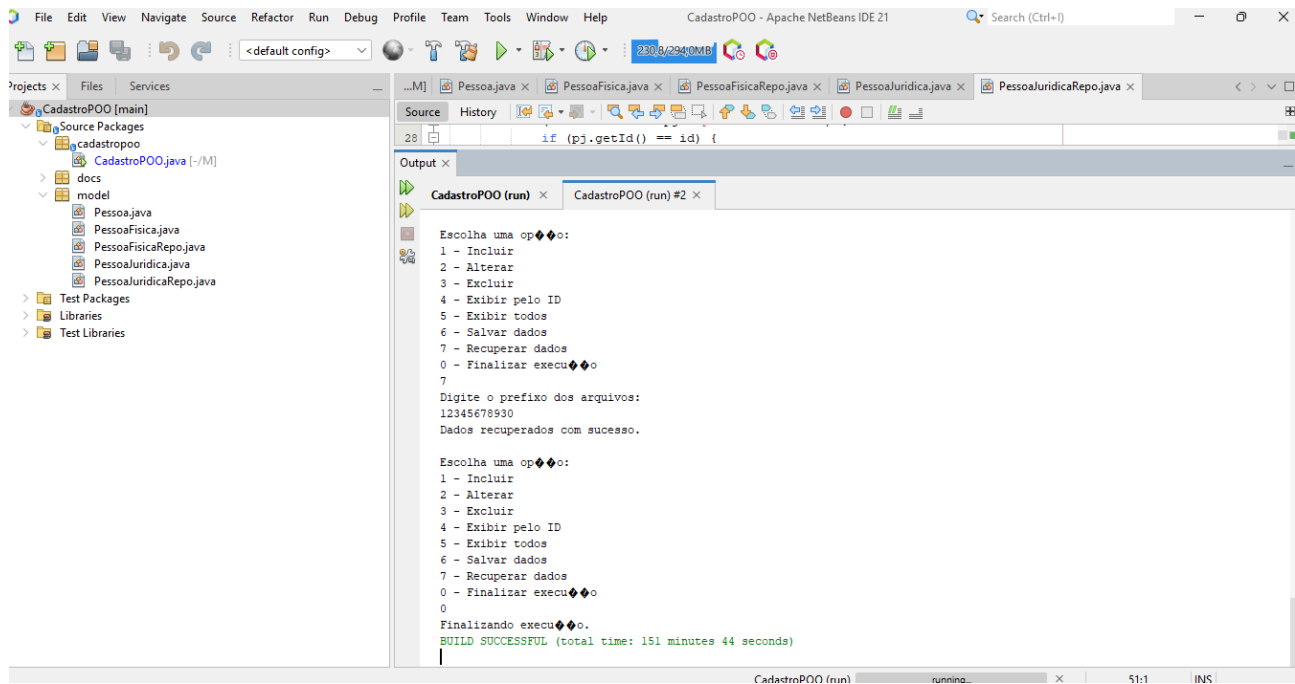
## 4.6 Salvar dados



## 4.7 Recuperar dados



## 4.8 Finalizar execução



## 5 ANALISE

### 5.1 O QUE SÃO ELEMENTOS ESTÁTICOS E QUAL O MOTIVO PARA O MÉTODO MAIN ADOTAR ESSE MODIFICADOR?

Elementos estáticos, como o método **main**, são associados à classe e não às suas instâncias, permitindo que a JVM (Java Virtual Machine) invoque **main** diretamente para iniciar a execução do programa. A utilização do modificador **static** é essencial para que o método **main** possa ser executado sem a necessidade de instanciar a classe **CadastroPOO**. Isso é crucial, pois o método **main** serve como ponto de entrada da aplicação, e a JVM precisa ser capaz de chamá-lo diretamente sem criar um objeto da classe.

### 5.2 PARA QUE SERVE A CLASSE **SCANNER**?

A classe **Scanner** desempenha um papel crucial na interação com o usuário, permitindo a leitura de diversos tipos de dados (como **int**, **double**, **String**) a partir da entrada padrão, geralmente o teclado. No contexto do sistema, **Scanner** é utilizado para capturar as escolhas do usuário no menu e para coletar os dados necessários nas operações de inclusão, alteração e exclusão de cadastros. Isso facilita a entrada e manipulação de dados, tornando a interface do usuário mais amigável e eficiente.

### 5.3 COMO O USO DE CLASSES DE REPOSITÓRIO IMPACTOU NA ORGANIZAÇÃO DO CÓDIGO?

O uso de classes de repositório (**PessoaFisicaRepo** e **PessoaJuridicaRepo**) é uma abordagem significativa que impacta positivamente a organização do código. Essas classes são responsáveis pela gestão dos dados, oferecendo métodos para inserir, alterar, excluir e recuperar registros. Ao encapsular a lógica de manipulação de dados, os repositórios promovem o princípio da responsabilidade única, isolando a lógica de persistência da lógica de interface com o usuário. Isso resulta em um código mais modular, legível e de fácil manutenção.

Além disso, a separação das operações de cadastro em métodos específicos na classe `CadastroPOO` (como `incluir`, `alterar`, `excluir`, `exibirPeloid`, `exibirTodos`, `salvarDados`, `recuperarDados`) contribui para uma melhor organização e clareza do código. Cada método tem uma responsabilidade bem definida, tornando o fluxo do programa mais intuitivo e facilitando futuras modificações ou expansões do sistema.

A modularização do código, juntamente com a utilização de repositórios, também facilita a implementação de funcionalidades adicionais, como validação de dados e tratamento de exceções. Por exemplo, ao persistir dados em arquivos binários, os métodos `persistir` e `recuperar` nas classes de repositório garantem a integridade dos dados armazenados, permitindo a recuperação e utilização dos dados em execuções subsequentes do programa.

## 6 CONCLUSÃO

O sistema de cadastro de pessoas físicas e jurídicas desenvolvido em Java demonstra a eficácia da aplicação de conceitos de programação orientada a objetos, como modularização e encapsulamento. A utilização de elementos estáticos, como o método `main`, facilita a inicialização do programa, enquanto a classe `Scanner` simplifica a captura de entradas do usuário, tornando a interação mais intuitiva. As classes de repositório, responsáveis pela gestão dos dados, promovem uma separação clara entre a lógica de negócio e a interface com o usuário, resultando em um código mais organizado e de fácil manutenção.

A modularização do código não só melhora a legibilidade, mas também facilita futuras expansões e adaptações do sistema. A abordagem adotada permite uma manutenção eficiente e a implementação de novas funcionalidades de maneira estruturada. Em suma, a estrutura do sistema é robusta, escalável e mantém uma clara separação de responsabilidades, garantindo a integridade e a eficiência da aplicação.

## 7 REFERÊNCIAS

**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).



