



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 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.

SUMARIO

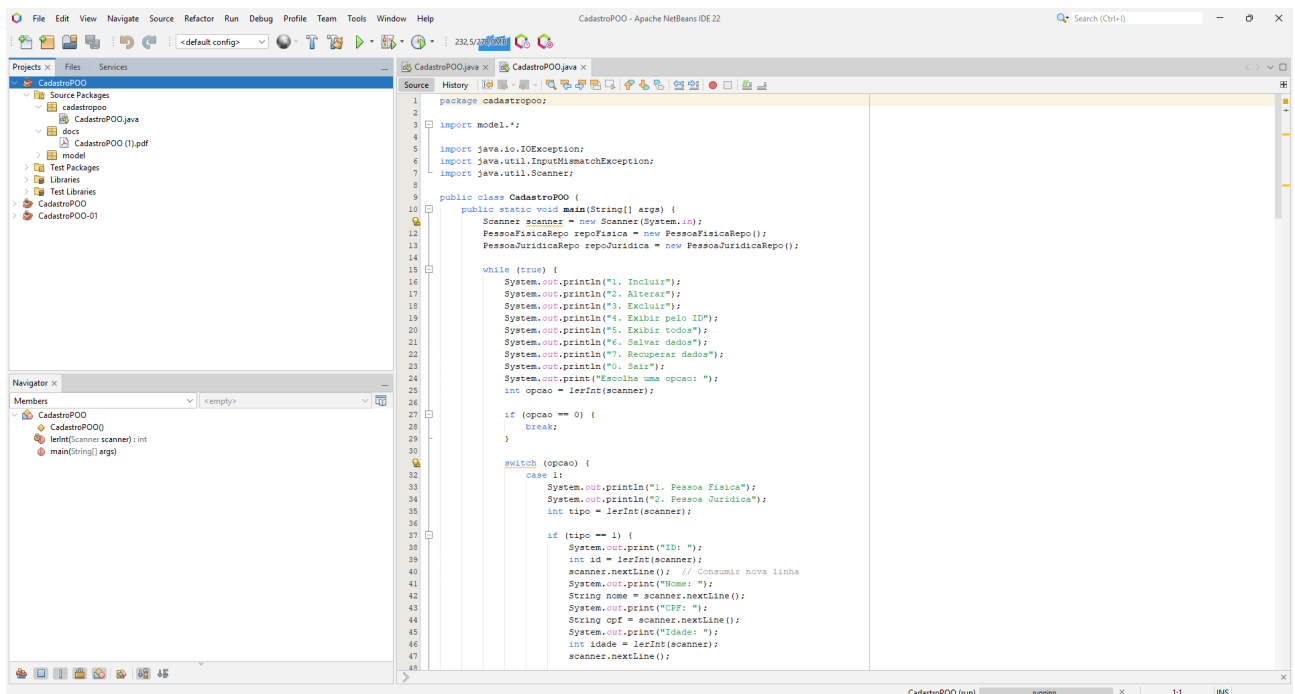
1 INTRODUÇÃO.....	5
2 CADASTRO POO.....	6
2.1 CÓDIGO CADASTRO POO.....	6
3 MODEL.....	9
3.1 PESSOA.JAVA.....	9
3.2 PESSOA FISICA.JAVA.....	10
3.3 PESSOA FISICA REPO.JAVA.....	11
3.4 PESSOA JURIDICA.JAVA.....	12
3.5 PESSOA JURIDICA REPO.JAVA.....	13
4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS.....	14
4.1 INCLUIR.....	14
4.2 ALTERAR.....	15
4.3 EXCLUIR.....	16
4.4 EXIBIR PELO ID.....	17
4.5 EXIBIR TODOS.....	18
4.6 SALVAR DADOS.....	19
4.7 RECUPERAR DADOS.....	20
5 ANALISE.....	21
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?.....	22
6 CONCLUSÃO.....	23
7 REFERÊNCIAS.....	24

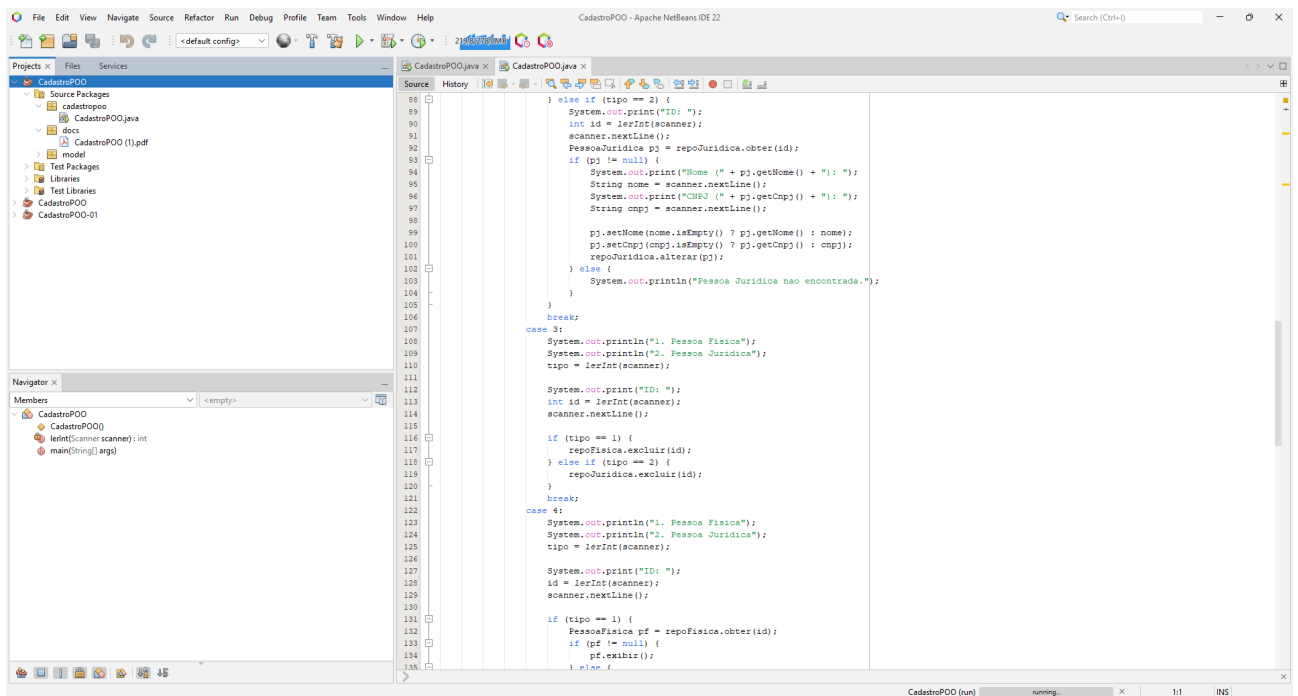
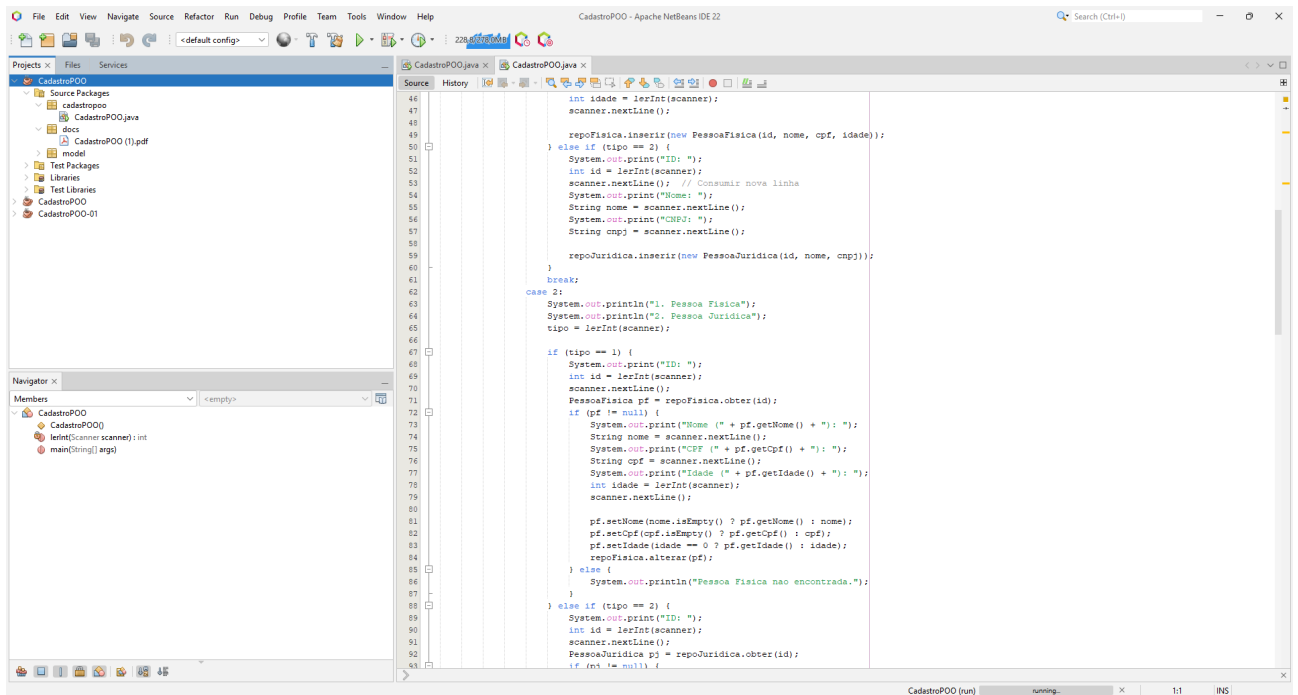
1 INTRODUÇÃO

Este projeto descreve a criação de um sistema para cadastro de pessoas físicas e jurídicas usando a linguagem de programação Java. O sistema foi desenhado para ser eficiente e claro, permitindo operações como adicionar, alterar, excluir, exibir e manter dados. A implementação utiliza conceitos essenciais de programação orientada a objetos (POO), incluindo encapsulamento, modularização e reutilização de código, facilitando a manutenção e a expansão do sistema. O ponto de entrada da aplicação é o método `main`, marcado como estático, permitindo que a JVM (Java Virtual Machine) o chame diretamente sem necessidade de instanciar a classe `CadastroPOO`. A interação com o usuário é gerenciada pela classe `Scanner`, que captura as entradas do usuário e facilita a navegação pelas várias 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 negócios e a interface do usuário. Esta estrutura não apenas melhora a legibilidade do código, mas também segue o princípio da responsabilidade única, tornando cada classe responsável por uma parte específica 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 CADASTRO POO

2.1 CÓDIGO CADASTRO POO



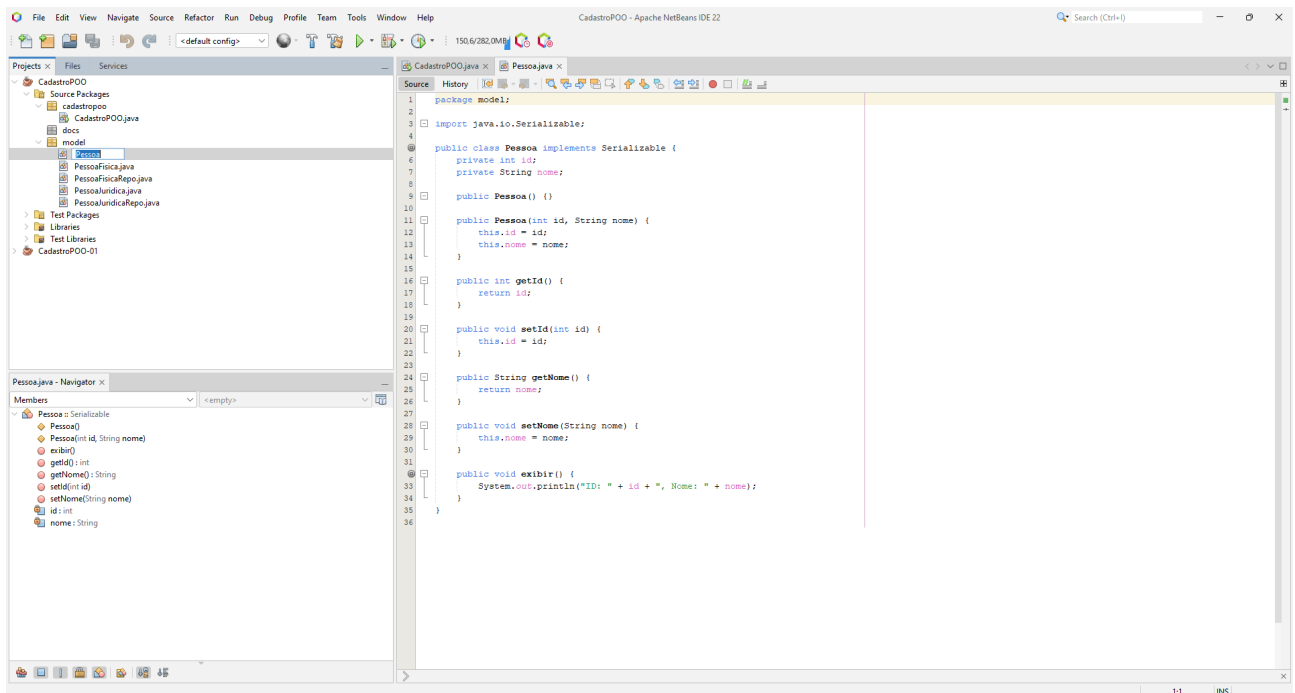


```
133         if (pf != null) {
134             pf.exibir();
135         } else {
136             System.out.println("Pessoa Fisica nao encontrada.");
137         }
138     } else if (tipo == 2) {
139         PessoaJuridica p = repoJuridica.obter(id);
140         if (p != null) {
141             p.exibir();
142         } else {
143             System.out.println("Pessoa Juridica nao encontrada.");
144         }
145     }
146     break;
147 case 5:
148     System.out.println("1. Pessoa Fisica");
149     System.out.println("2. Pessoa Juridica");
150     tipo = lerInt(scanner);
151
152     if (tipo == 1) {
153         for (PessoaFisica pf : repoFisica.obterTodos()) {
154             pf.exibir();
155         }
156     } else if (tipo == 2) {
157         for (PessoaJuridica pj : repoJuridica.obterTodos()) {
158             pj.exibir();
159         }
160     }
161     break;
162 case 6:
163     System.out.print("Prefixo do arquivo: ");
164     String prefixo = scanner.nextLine();
165
166     try {
167         repoFisica.persistir(prefixo + ".fisica.bin");
168         repoJuridica.persistir(prefixo + ".juridica.bin");
169     } catch (IOException e) {
170         System.out.println("Erro ao salvar os dados: " + e.getMessage());
171     }
172     break;
173 case 7:
174     System.out.print("Prefixo do arquivo: ");
175     prefixo = scanner.nextLine();
176
177     try {
178         repoFisica.recuperar(prefixo + ".fisica.bin");
179         repoJuridica.recuperar(prefixo + ".juridica.bin");
180     } catch (IOException | ClassNotFoundException e) {
181         System.out.println("Erro ao recuperar os dados: " + e.getMessage());
182     }
183     break;
184 default:
185     System.out.println("Opção invalida.");
186     break;
187 }
188 scanner.close();
189
190 private static int lerInt(Scanner scanner) {
191     while (true) {
192         try {
193             return scanner.nextInt();
194         } catch (InputMismatchException e) {
195             System.out.print("Entrada invalida. Digite um numero inteiro: ");
196             scanner.next(); // Consumir a entrada invalida
197         }
198     }
199 }
```

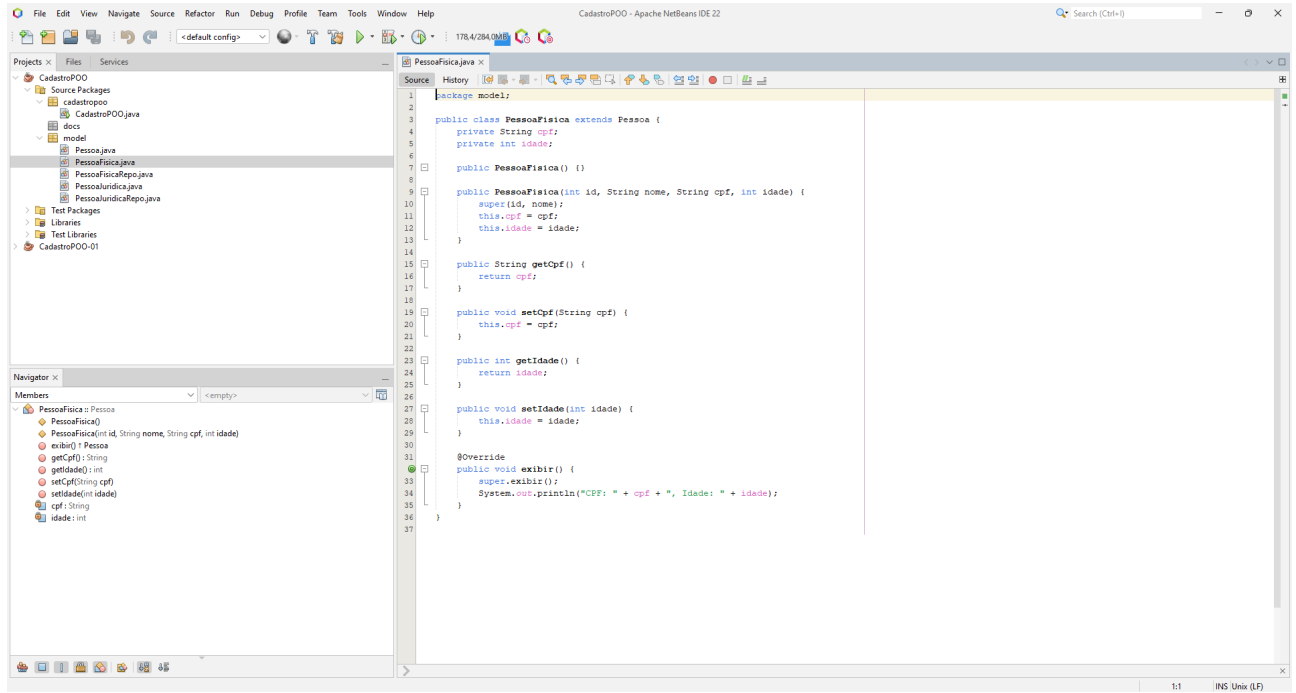
```
172         break;
173 case 7:
174     System.out.print("Prefixo do arquivo: ");
175     prefixo = scanner.nextLine();
176
177     try {
178         repoFisica.recuperar(prefixo + ".fisica.bin");
179         repoJuridica.recuperar(prefixo + ".juridica.bin");
180     } catch (IOException | ClassNotFoundException e) {
181         System.out.println("Erro ao recuperar os dados: " + e.getMessage());
182     }
183     break;
184 default:
185     System.out.println("Opção invalida.");
186     break;
187 }
188 scanner.close();
189
190 private static int lerInt(Scanner scanner) {
191     while (true) {
192         try {
193             return scanner.nextInt();
194         } catch (InputMismatchException e) {
195             System.out.print("Entrada invalida. Digite um numero inteiro: ");
196             scanner.next(); // Consumir a entrada invalida
197         }
198     }
199 }
```


3 MODEL

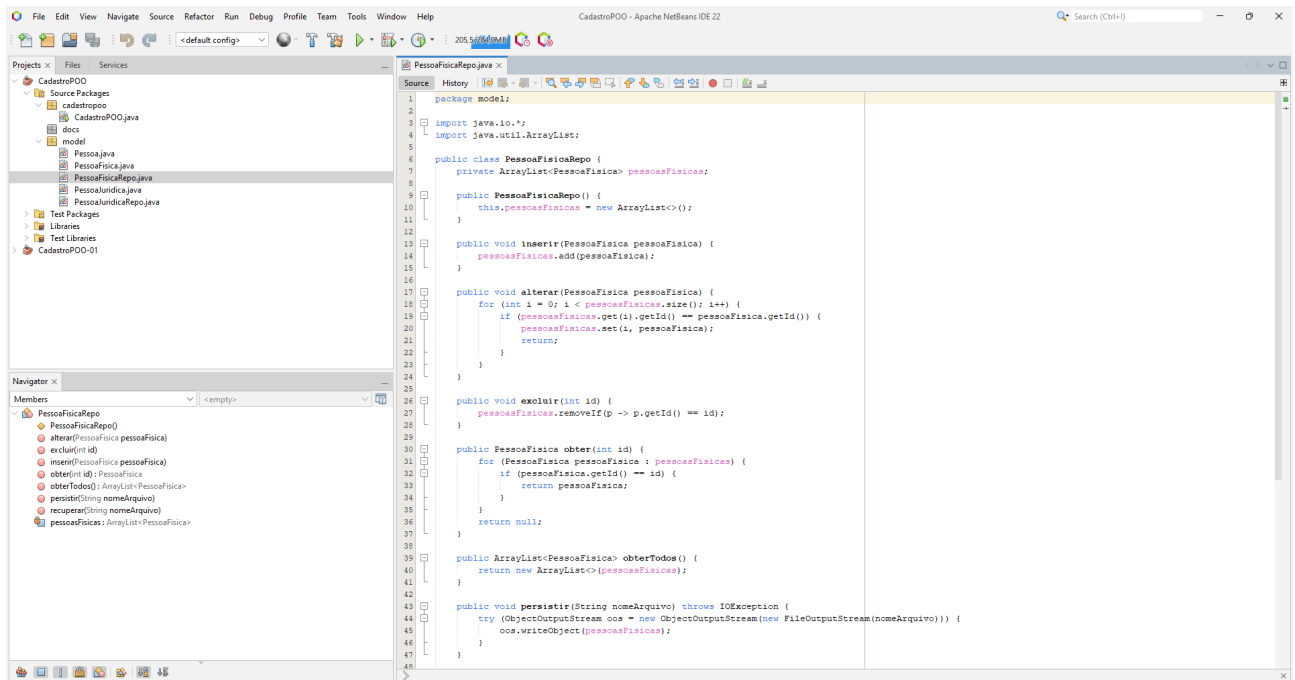
3.1 PESSOA.JAVA



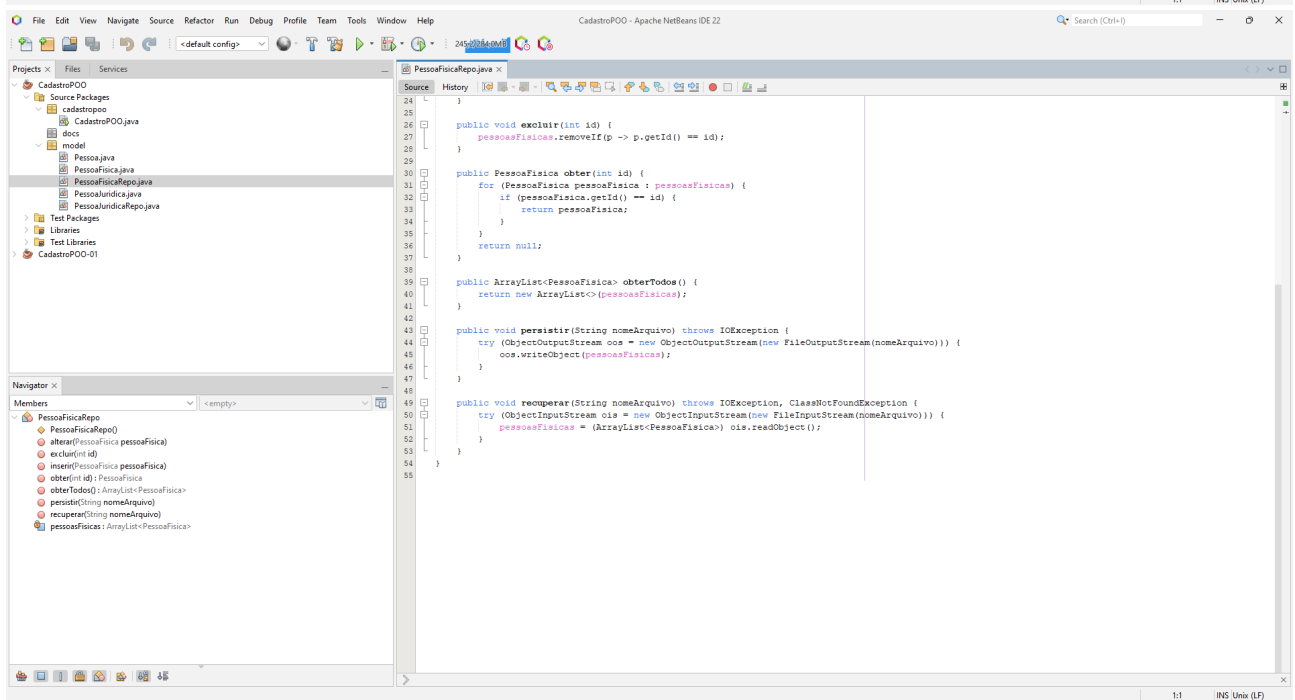
3.2 PESSOA FISICA.JAVA



3.3 PESSOA FISICA REPO.JAVA

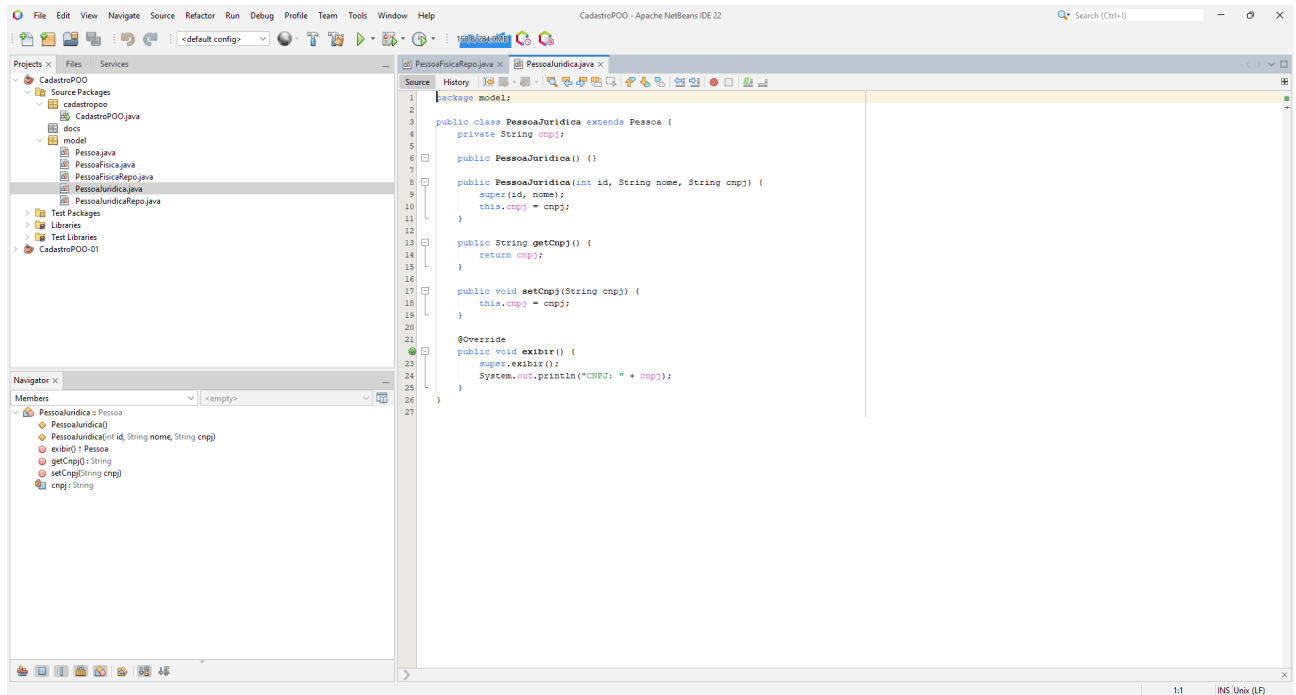


```
1 package model;
2
3 import java.io.*;
4 import java.util.ArrayList;
5
6 public class PessoaFisicaRepo {
7     private ArrayList<PessoaFisica> pessoasFisicas;
8
9     public PessoaFisicaRepo() {
10         this.pessoasFisicas = new ArrayList<>();
11     }
12
13     public void inserir(PessoaFisica pessoaFisica) {
14         pessoasFisicas.add(pessoaFisica);
15     }
16
17     public void alterar(PessoaFisica pessoaFisica) {
18         for (int i = 0; i < pessoasFisicas.size(); i++) {
19             if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {
20                 pessoasFisicas.set(i, pessoaFisica);
21                 return;
22             }
23         }
24     }
25
26     public void excluir(int id) {
27         pessoasFisicas.removeIf(p -> p.getId() == id);
28     }
29
30     public PessoaFisica obter(int id) {
31         for (PessoaFisica pessoaFisica : pessoasFisicas) {
32             if (pessoaFisica.getId() == id) {
33                 return pessoaFisica;
34             }
35         }
36         return null;
37     }
38
39     public ArrayList<PessoaFisica> obterTodos() {
40         return new ArrayList<>(pessoasFisicas);
41     }
42
43     public void persistir(String nomeArquivo) throws IOException {
44         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
45             oos.writeObject(pessoasFisicas);
46         }
47     }
48 }
```

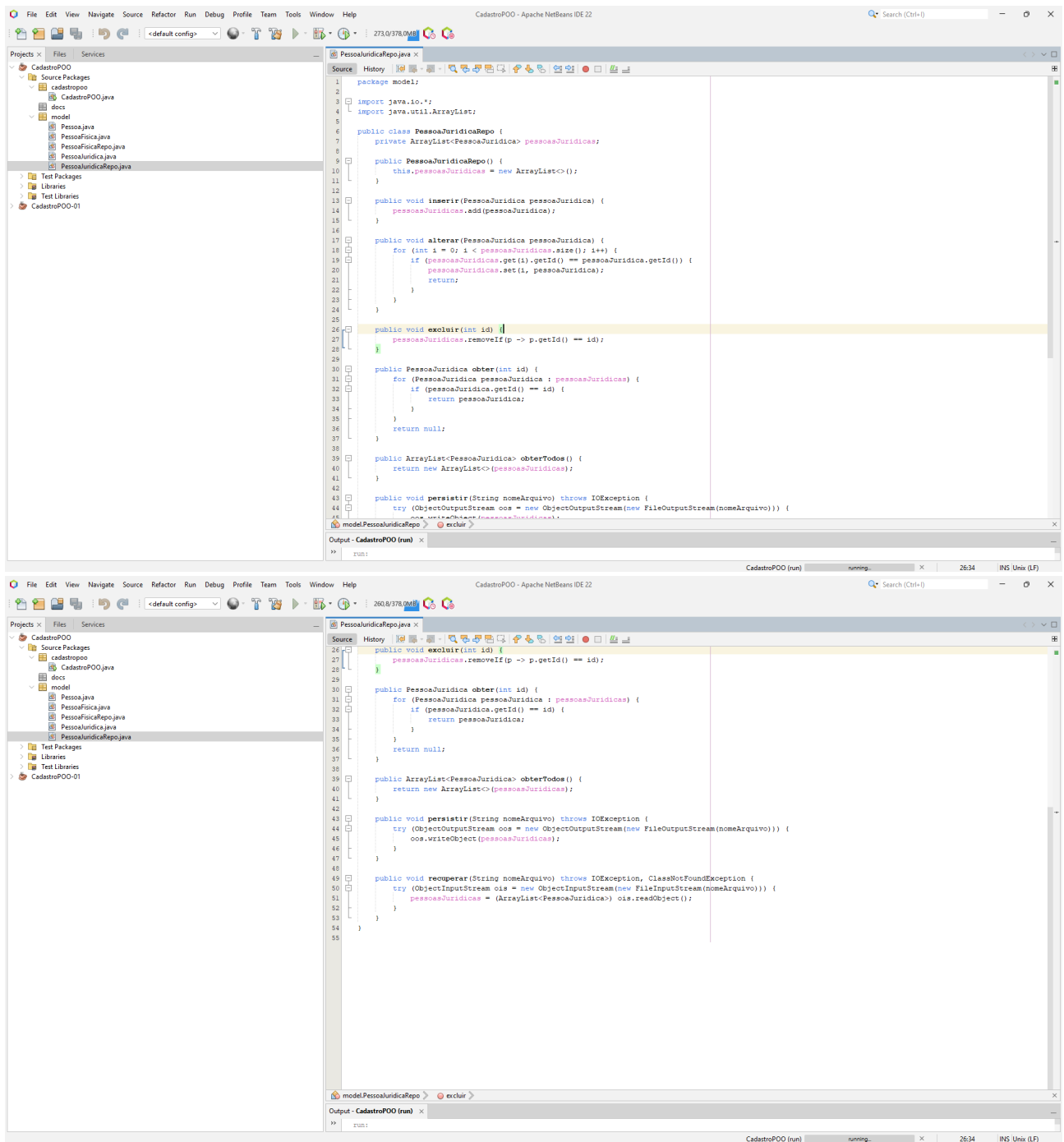


```
24 }
25
26 public void excluir(int id) {
27     pessoasFisicas.removeIf(p -> p.getId() == id);
28 }
29
30 public PessoaFisica obter(int id) {
31     for (PessoaFisica pessoaFisica : pessoasFisicas) {
32         if (pessoaFisica.getId() == id) {
33             return pessoaFisica;
34         }
35     }
36     return null;
37 }
38
39 public ArrayList<PessoaFisica> obterTodos() {
40     return new ArrayList<>(pessoasFisicas);
41 }
42
43 public void persistir(String nomeArquivo) throws IOException {
44     try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
45         oos.writeObject(pessoasFisicas);
46     }
47 }
48
49 public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
50     try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
51         pessoasFisicas = (ArrayList<PessoaFisica>) ois.readObject();
52     }
53 }
54
55 }
```

3.4 PESSOA JURIDICA.JAVA

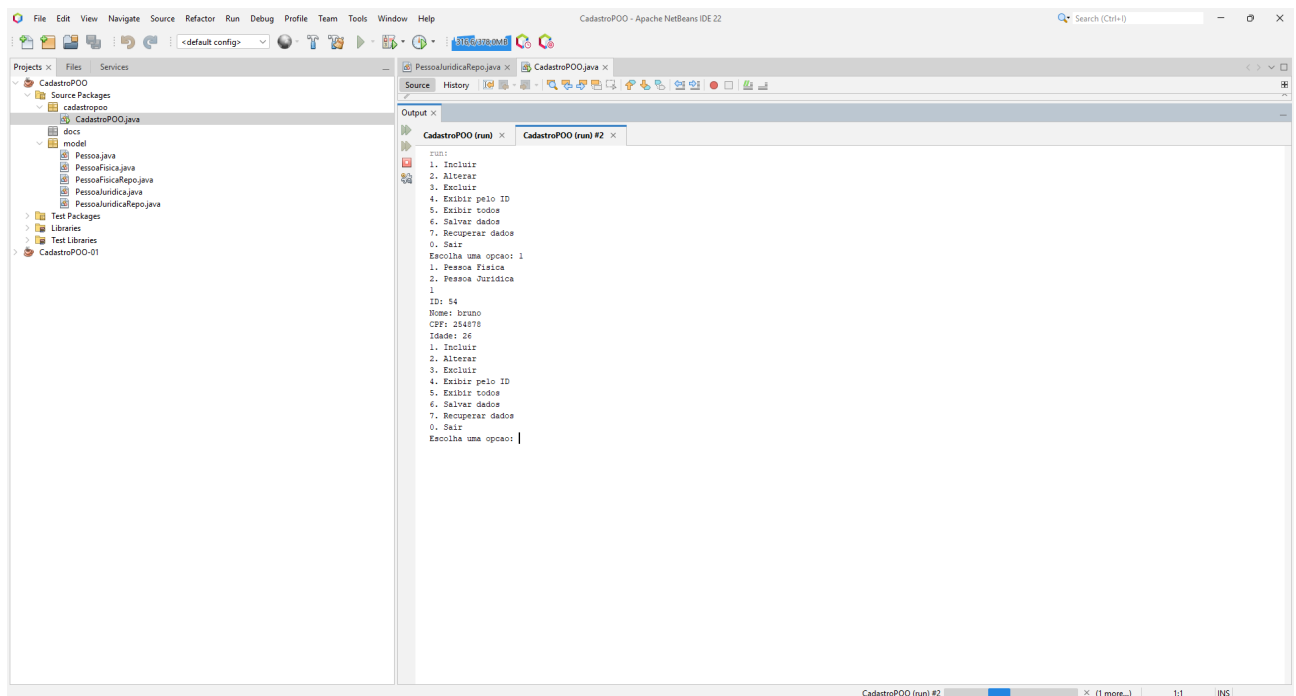


3.5 PESSOA JURIDICA REPO.JAVA

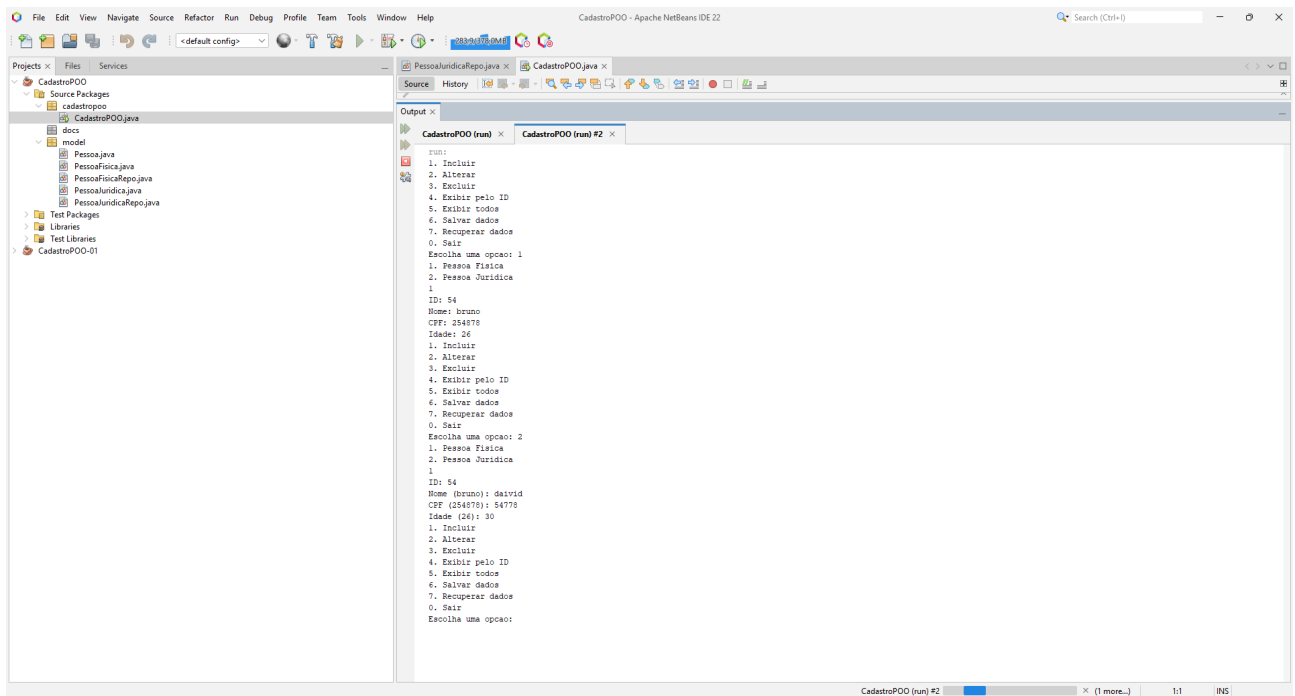


4 RESULTADOS DA EXECUÇÃO DOS CÓDIGOS

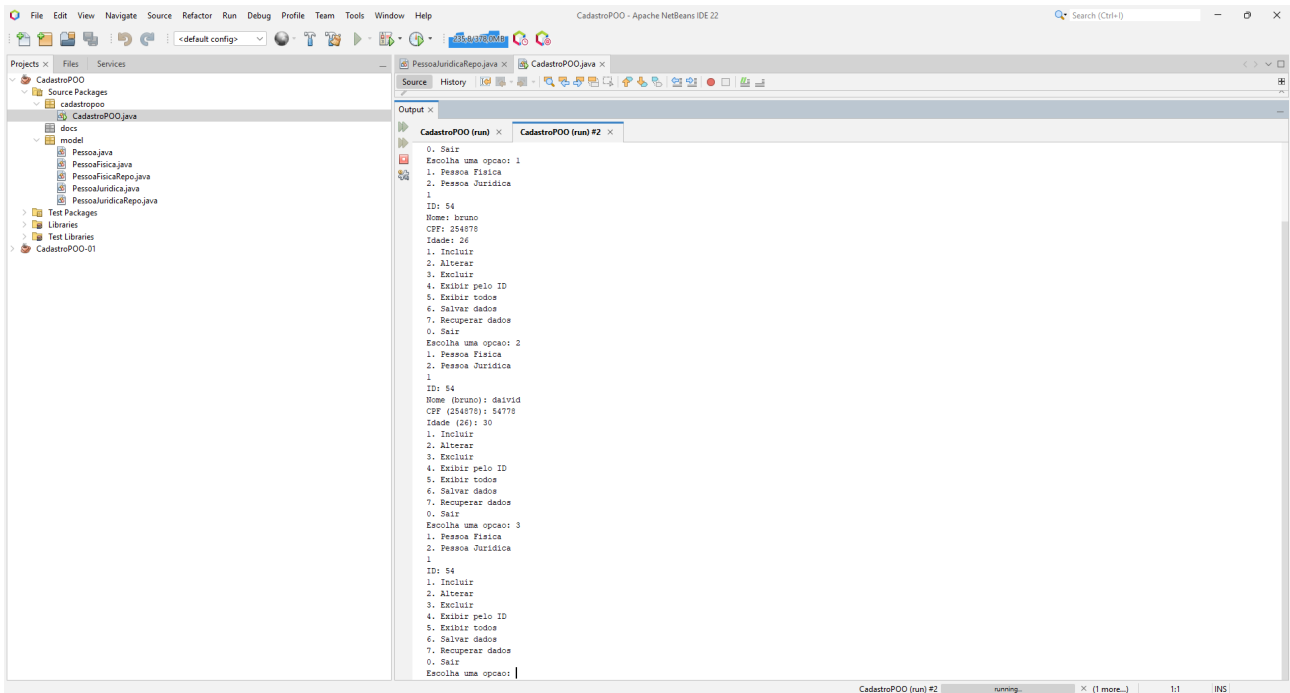
4.1 INCLUIR



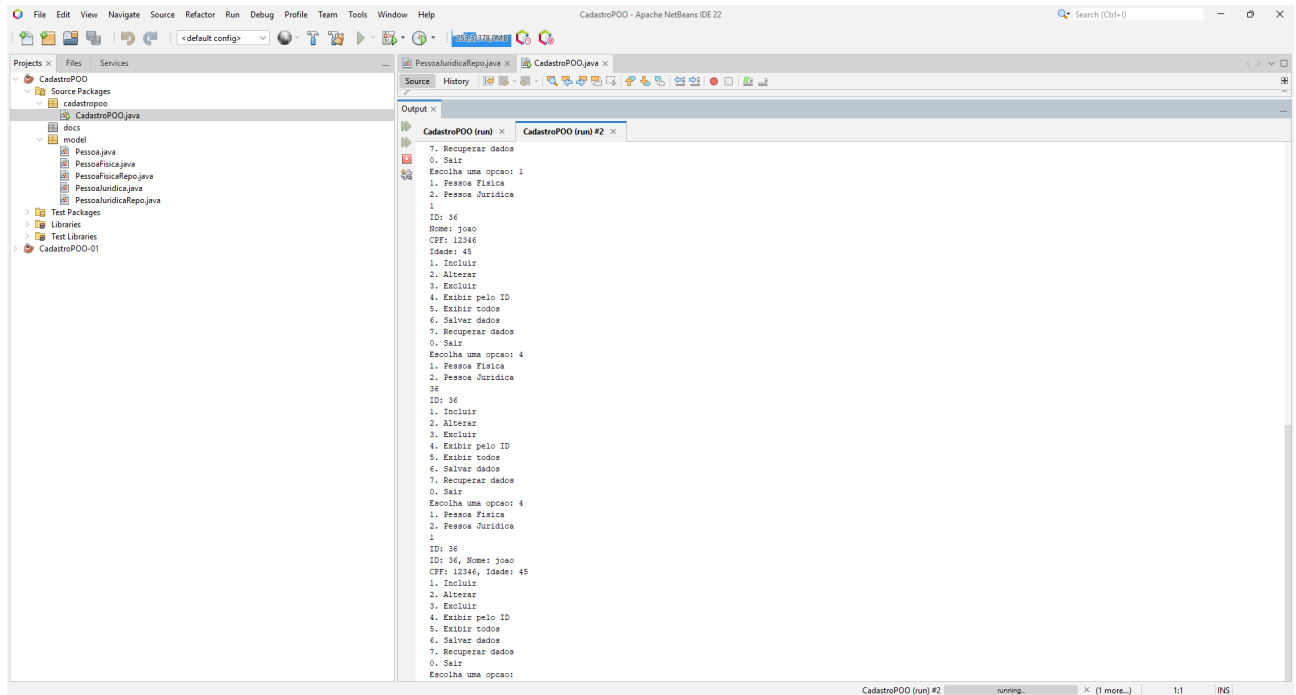
4.2 ALTERAR



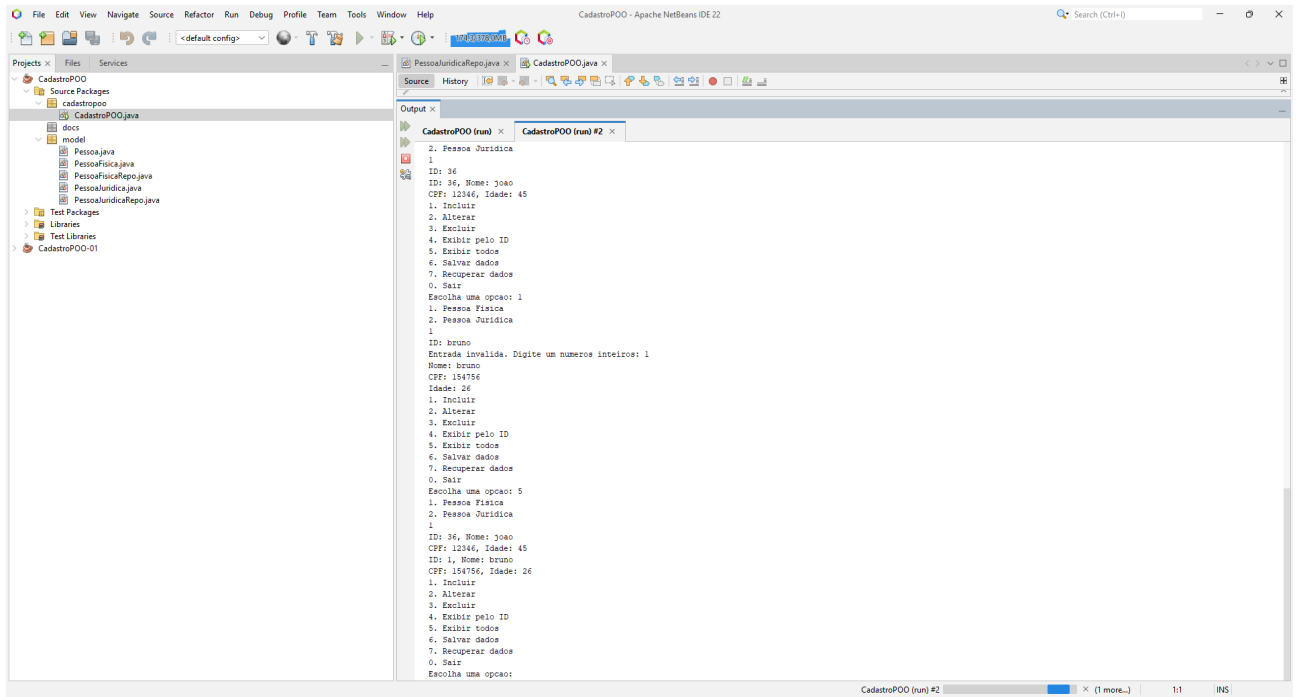
4.3 EXCLUIR



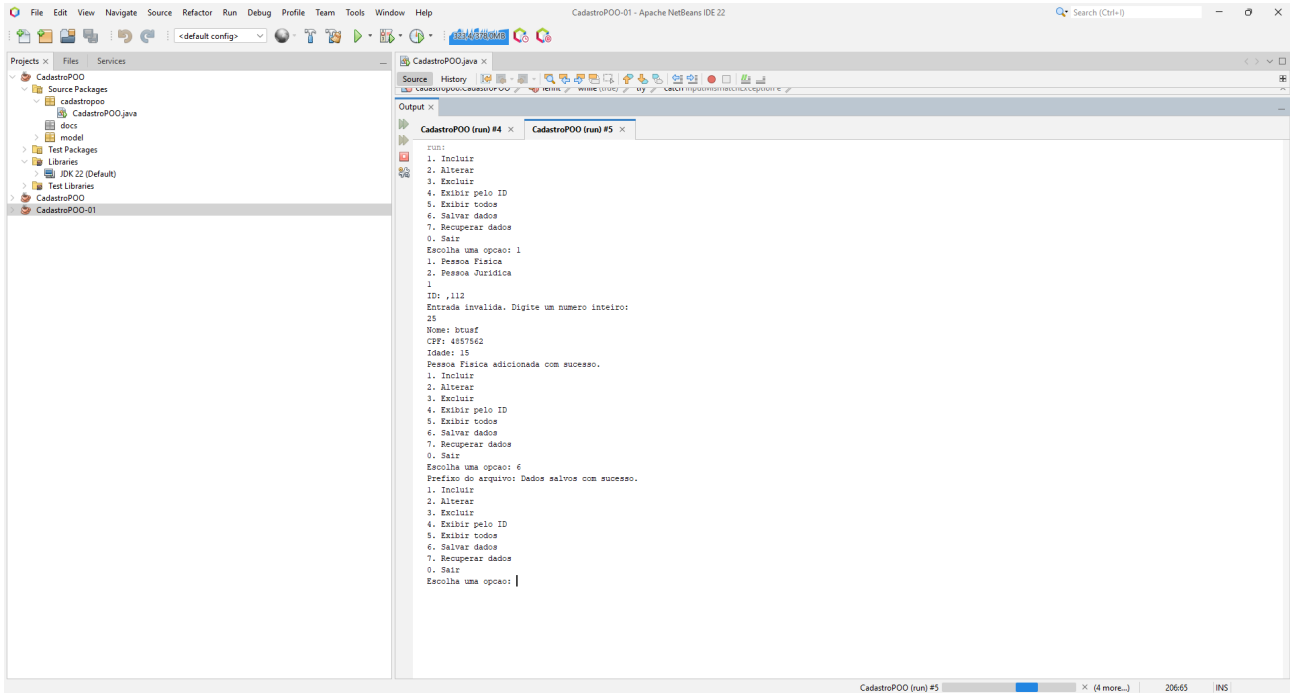
4.4 EXIBIR PELO ID



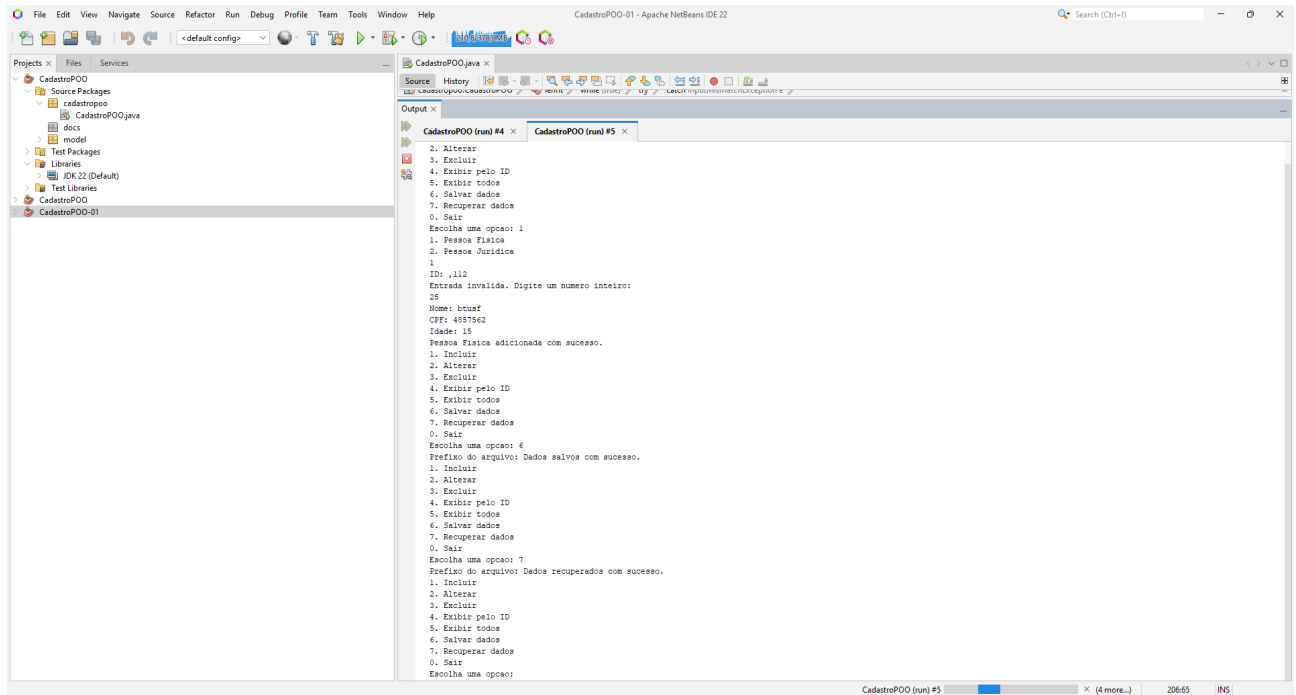
4.5 EXIBIR TODOS



4.6 SALVAR DADOS



4.7 RECUPERAR DADOS



5 ANALISE

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

Elementos estáticos, como o método main, estão associados à classe em vez de suas instâncias, permitindo que a JVM (Java Virtual Machine) invoque o main diretamente para iniciar a execução do programa. O uso do modificador static é essencial para que o método main possa ser executado sem a necessidade de instanciar a classe CadastroPOO. Isso é crucial porque 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 tem um papel fundamental na interação com o usuário, possibilitando a leitura de diferentes tipos de dados (como inteiros, números decimais e strings) a partir da entrada padrão, geralmente o teclado. No contexto do sistema, o Scanner é utilizado para capturar as escolhas do usuário no menu e para coletar os dados necessários durante as operações de inclusão, alteração e exclusão de cadastros. Isso simplifica a entrada e manipulação de dados, contribuindo para uma interface do usuário mais amigável e eficiente.

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

O uso das classes de repositório, como `PessoaFisicaRepo` e `PessoaJuridicaRepo`, é uma abordagem essencial que melhora significativamente a organização do código. Essas classes são responsáveis pela administração dos dados, oferecendo métodos para adicionar, atualizar, remover e recuperar registros. Ao encapsular a lógica de manipulação de dados, os repositórios seguem o princípio da responsabilidade única, separando claramente a lógica de persistência da interação com o usuário. Isso resulta em um código mais modular, legível e fácil de manter.

Além disso, a divisão das operações de cadastro em métodos específicos na classe `CadastroPOO` (como `adicionar`, `modificar`, `excluir`, `exibirPorId`, `exibirTodos`, `salvarDados`, `recuperarDados`) contribui para uma estrutura mais organizada e compreensível do código. Cada método possui uma função bem definida, o que torna o fluxo do programa mais intuitivo e simplifica modificações ou expansões futuras do sistema.

A modularização do código, junto com o uso 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 de persistência e recuperação nas classes de repositório garantem a integridade dos dados armazenados, possibilitando a recuperação e utilização desses dados em execuções posteriores do programa.

6 CONCLUSÃO

O sistema desenvolvido em Java para cadastro de pessoas físicas e jurídicas exemplifica a aplicação eficaz de conceitos de programação orientada a objetos, como modularização e encapsulamento. O uso de elementos estáticos, como o método main, simplifica a inicialização do programa, enquanto a classe Scanner facilita a captura de entradas do usuário, tornando a interação mais intuitiva. As classes de repositório, responsáveis pela administração dos dados, estabelecem uma separação clara entre a lógica de negócios e a interface com o usuário, resultando em um código mais organizado e fácil de manter.

A modularização não apenas melhora a legibilidade, mas também facilita futuras expansões e adaptações do sistema. Essa abordagem permite uma manutenção eficiente e a implementação estruturada de novas funcionalidades. Em resumo, a estrutura do sistema é robusta, escalável e mantém uma clara divisão de responsabilidades, assegurando a integridade e eficiência da aplicação.

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/>.