



Estácio

CAMPUS POLO AUSTIN - NOVA IGUAÇU – RJ

DESENVOLVIMENTO FULL STACK

Nível 1: Iniciando o Caminho Pelo Java

RPG0014 9001

2024/2

SALOMÃO ISAAC CARVALHO GARCIA

**Desenvolvimento de Sistema de Cadastro em Java com Persistência em
Arquivos**

Nova Iguaçu

15/07/2024

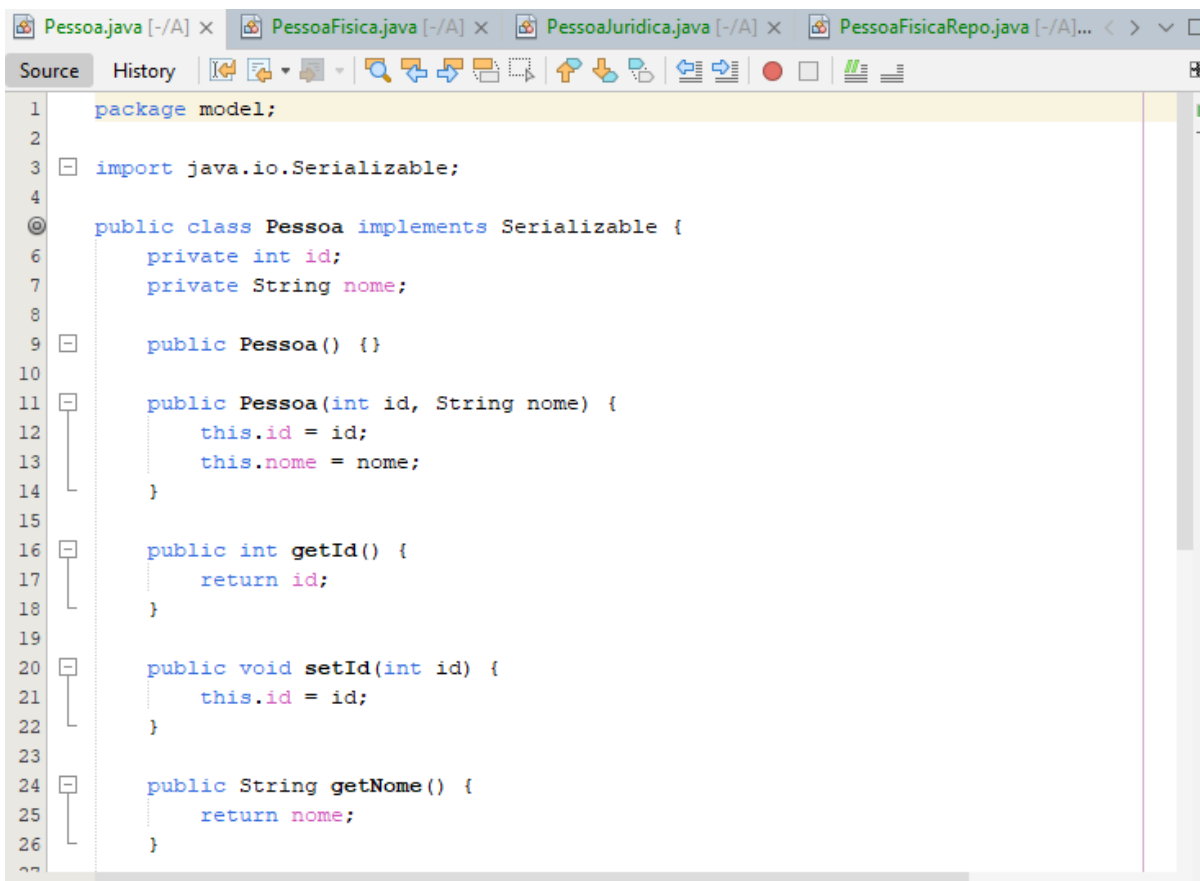
Desenvolvimento de Sistema de Cadastro em Java com Persistência em Arquivos

Objetivo da Prática: Implementar um sistema de cadastro de clientes em Java utilizando herança, polimorfismo, e persistência em arquivos binários.

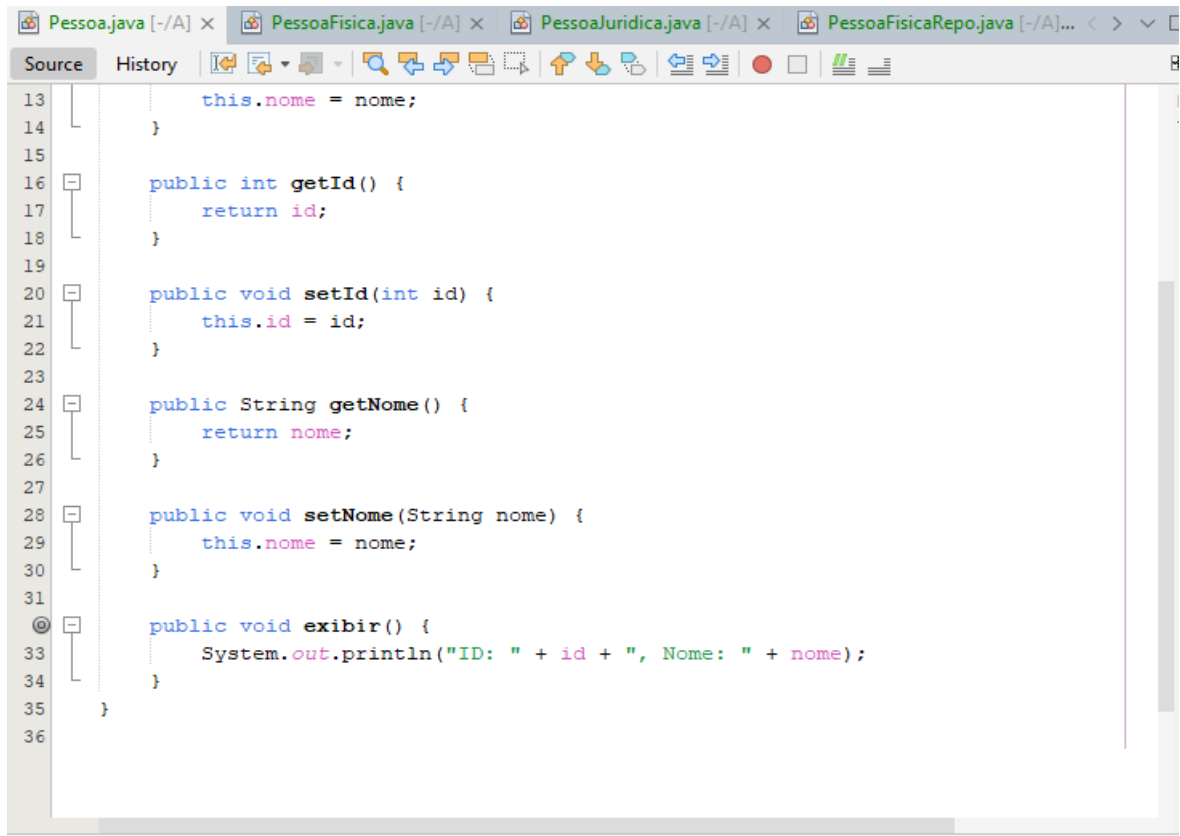
<https://github.com/SaloGarcia/Nivel01.git>

CÓDIGOS

Pessoa.java

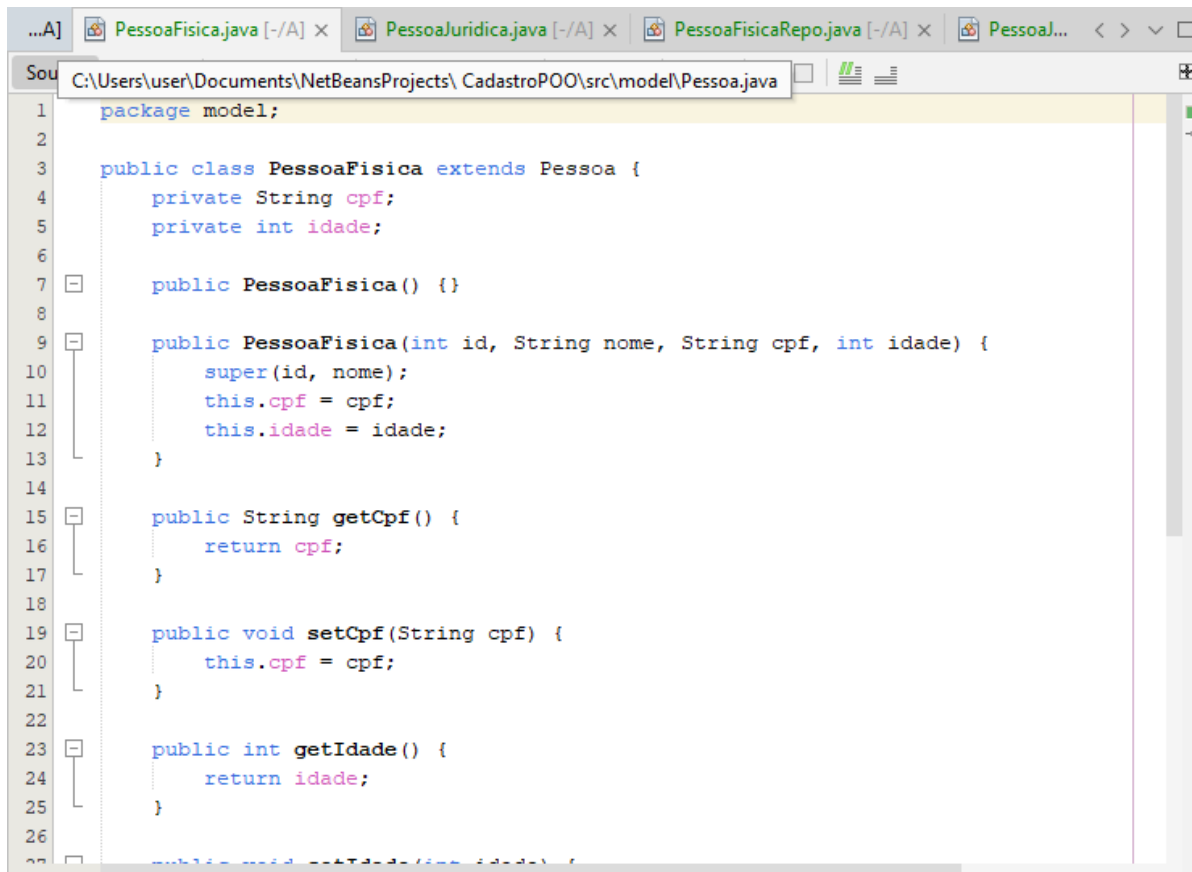
A screenshot of an IDE window showing the code for Pessoa.java. The window has multiple tabs at the top: Pessoa.java, PessoaFisica.java, PessoaJuridica.java, and PessoaFisicaRepo.java. The 'Source' tab is active, showing the code for Pessoa.java. The code is as follows:

```
1 package model;
2
3 import java.io.Serializable;
4
5 public class Pessoa implements Serializable {
6     private int id;
7     private String nome;
8
9     public Pessoa() {}
10
11     public Pessoa(int id, String nome) {
12         this.id = id;
13         this.nome = nome;
14     }
15
16     public int getId() {
17         return id;
18     }
19
20     public void setId(int id) {
21         this.id = id;
22     }
23
24     public String getNome() {
25         return nome;
26     }
27 }
```

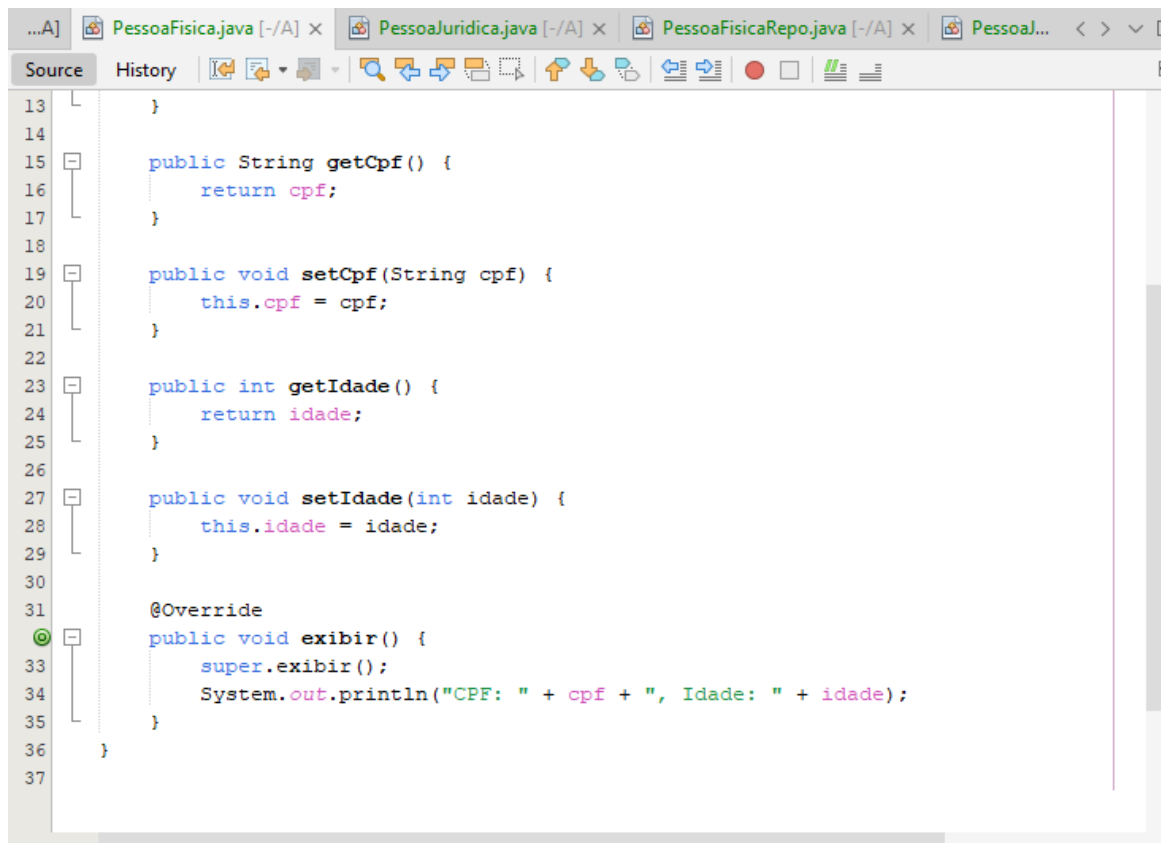


```
13         this.nome = nome;
14     }
15
16     public int getId() {
17         return id;
18     }
19
20     public void setId(int id) {
21         this.id = id;
22     }
23
24     public String getNome() {
25         return nome;
26     }
27
28     public void setNome(String nome) {
29         this.nome = nome;
30     }
31
32     public void exibir() {
33         System.out.println("ID: " + id + ", Nome: " + nome);
34     }
35 }
36
```

PessoaFisica.java

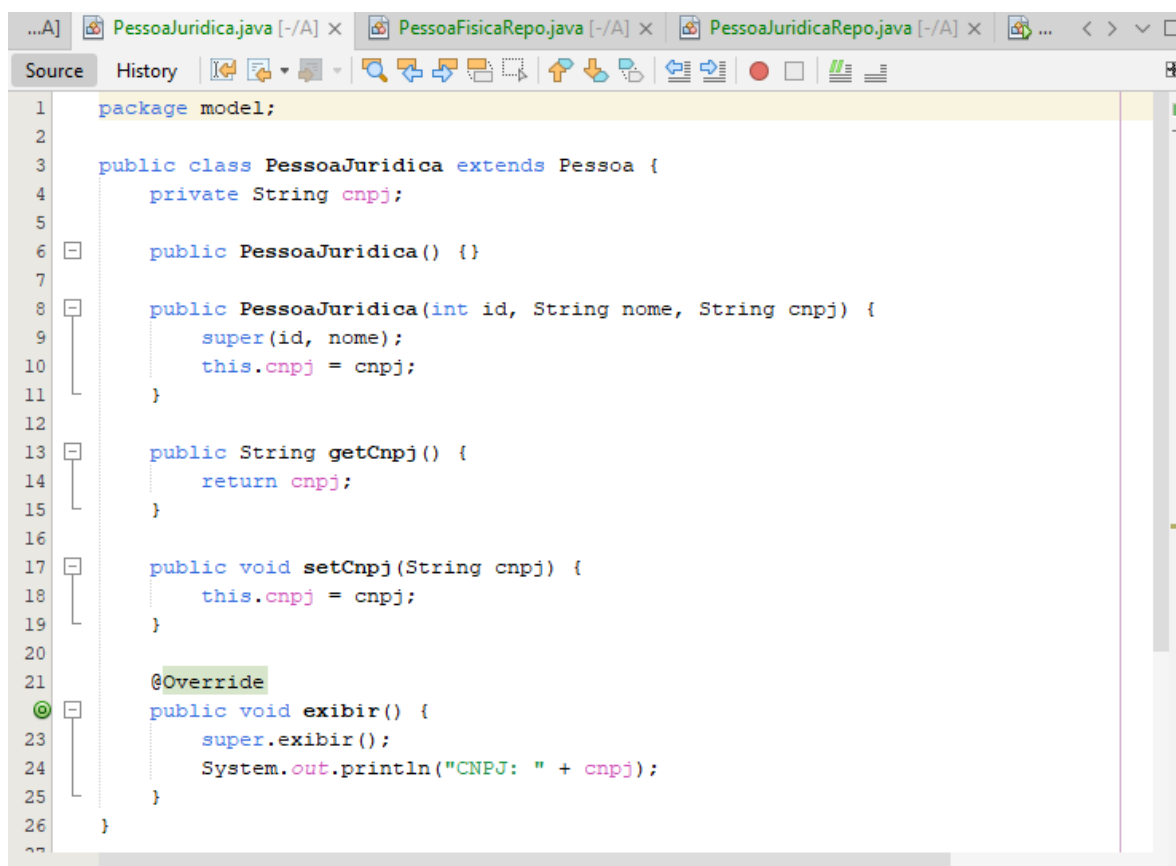


```
1 package model;
2
3 public class PessoaJuridica extends Pessoa {
4     private String cpf;
5     private int idade;
6
7     public PessoaJuridica() {}
8
9     public PessoaJuridica(int id, String nome, String cpf, int idade) {
10         super(id, nome);
11         this.cpf = cpf;
12         this.idade = idade;
13     }
14
15     public String getCpf() {
16         return cpf;
17     }
18
19     public void setCpf(String cpf) {
20         this.cpf = cpf;
21     }
22
23     public int getIdade() {
24         return idade;
25     }
26
27     public void setIdade(int idade) {
28         this.idade = idade;
29     }
30 }
31
```



```
13     }
14
15     public String getCPF() {
16         return cpf;
17     }
18
19     public void setCPF(String cpf) {
20         this.cpf = cpf;
21     }
22
23     public int getIdade() {
24         return idade;
25     }
26
27     public void setIdade(int idade) {
28         this.idade = idade;
29     }
30
31     @Override
32     public void exibir() {
33         super.exibir();
34         System.out.println("CPF: " + cpf + ", Idade: " + idade);
35     }
36 }
37
```

PessoaJuridica.java



```
1 package model;
2
3 public class PessoaJuridica extends Pessoa {
4     private String cnpj;
5
6     public PessoaJuridica() {}
7
8     public PessoaJuridica(int id, String nome, String cnpj) {
9         super(id, nome);
10        this.cnpj = cnpj;
11    }
12
13    public String getCnpj() {
14        return cnpj;
15    }
16
17    public void setCnpj(String cnpj) {
18        this.cnpj = cnpj;
19    }
20
21    @Override
22    public void exibir() {
23        super.exibir();
24        System.out.println("CNPJ: " + cnpj);
25    }
26 }
27
```

PessoaFisicaRepo.java

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
1 package model;
2
3 import java.io.*;
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class PessoaFisicaRepo {
8     private List<PessoaFisica> pessoasFisicas;
9
10    public PessoaFisicaRepo() {
11        pessoasFisicas = new ArrayList<>();
12    }
13
14    public void inserir(PessoaFisica pessoaFisica) {
15        pessoasFisicas.add(pessoaFisica);
16    }
17
18    public void alterar(PessoaFisica pessoaFisica) {
19        for (int i = 0; i < pessoasFisicas.size(); i++) {
20            if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {
21                pessoasFisicas.set(i, pessoaFisica);
22                return;
23            }
24        }
25    }
26
27    public void excluir(int id) {
28        pessoasFisicas.removeIf(p -> p.getId() == id);
29    }
30
31    public PessoaFisica obter(int id) {
32        return pessoasFisicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
33    }
34
35    public List<PessoaFisica> obterTodos() {
36        return new ArrayList<>(pessoasFisicas);
37    }
38
39    public void persistir(String nomeArquivo) throws IOException {
40        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
41            oos.writeObject(pessoasFisicas);
42        }
43    }
44
45    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
46        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
47            pessoasFisicas = (List<PessoaFisica>) ois.readObject();
48        }
49    }
50 }
51
```

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
25 }
26
27 public void excluir(int id) {
28     pessoasFisicas.removeIf(p -> p.getId() == id);
29 }
30
31 public PessoaFisica obter(int id) {
32     return pessoasFisicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
33 }
34
35 public List<PessoaFisica> obterTodos() {
36     return new ArrayList<>(pessoasFisicas);
37 }
38
39 public void persistir(String nomeArquivo) throws IOException {
40     try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
41         oos.writeObject(pessoasFisicas);
42     }
43 }
44
45 public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
46     try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
47         pessoasFisicas = (List<PessoaFisica>) ois.readObject();
48     }
49 }
50 }
51
```

```
...[A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
25
26
27 1uir(int id) {
28 2as.removeIf(p -> p.getId() == id);
29
30
31 3ica obter(int id) {
32 4asFisicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
33
34
35 5oaFisica> obterTodos() {
36 6rrayList<>(pessoasFisicas);
37
38
39 7istir(String nomeArquivo) throws IOException {
40 8utputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo)) {
41 9eObject(pessoasFisicas);
42
43
44
45 10perar(String nomeArquivo) throws IOException, ClassNotFoundException {
46 11nputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo)) {
47 12fisicas = (List<PessoaFisica>) ois.readObject();
48
49
50
51
```

PessoaJuridicaRepo.java

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

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
25 }
26
27 public void excluir(int id) {
28     pessoasJuridicas.removeIf(p -> p.getId() == id);
29 }
30
31 public PessoaJuridica obter(int id) {
32     return pessoasJuridicas.stream().filter(p -> p.getId() == id).findFirst().get();
33 }
34
35 public List<PessoaJuridica> obterTodos() {
36     return new ArrayList<>(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 = (List<PessoaJuridica>) ois.readObject();
48     }
49 }
50 }
```

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
25 }
26
27 public void excluir(int id) {
28     pessoasJuridicas.removeIf(p -> p.getId() == id);
29 }
30
31 public PessoaJuridica obter(int id) {
32     return pessoasJuridicas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
33 }
34
35 public List<PessoaJuridica> obterTodos() {
36     return new ArrayList<>(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 = (List<PessoaJuridica>) ois.readObject();
48     }
49 }
50 }
```

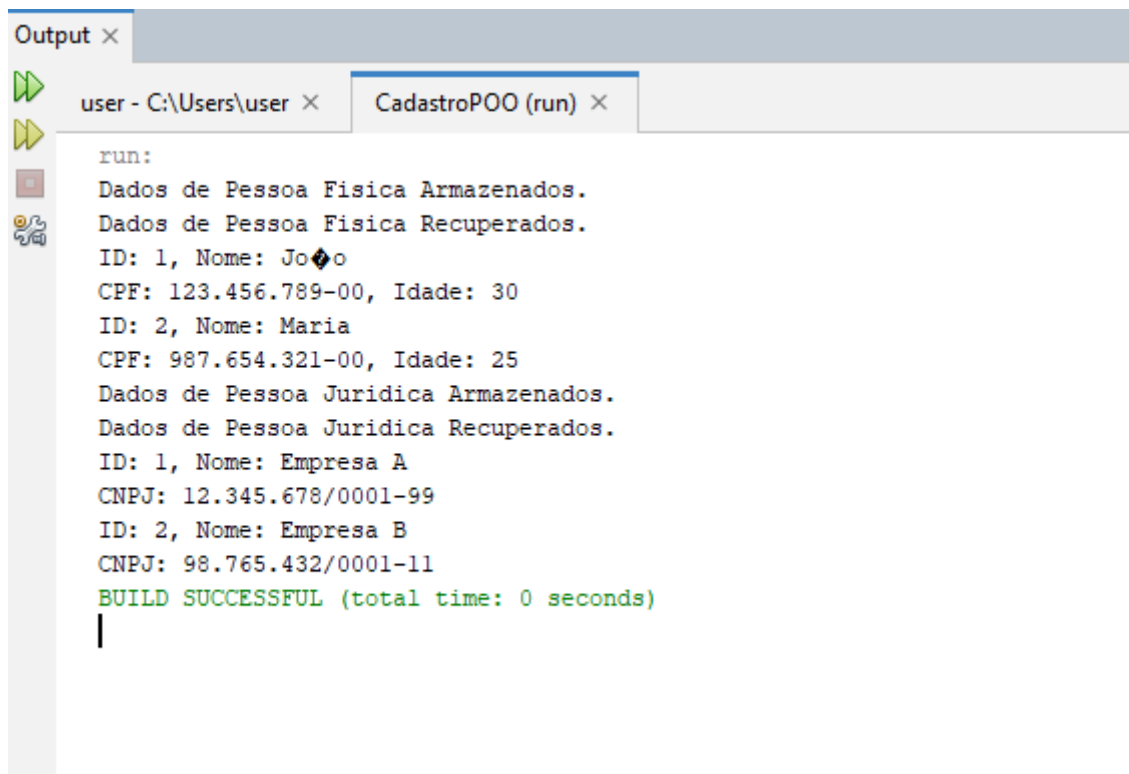
Main.java

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
1 package cadastroPoo;
2
3 import model.*;
4
5 import java.io.IOException;
6 import java.util.List;
7
8 public class Main {
9     public static void main(String[] args) {
10         try {
11             // Testando repositório de PessoaFisica
12             PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
13             repo1.inserir(new PessoaFisica(1, "João", "123.456.789-00", 30));
14             repo1.inserir(new PessoaFisica(2, "Maria", "987.654.321-00", 25));
15
16             System.out.println("Dados de Pessoa Fisica Armazenados.");
17
18             repo1.persistir("pessoasFisicas.dat");
19
20             System.out.println("Dados de Pessoa Fisica Recuperados.");
21
22             PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
23             repo2.recuperar("pessoasFisicas.dat");
24             List<PessoaFisica> pessoasFisicas = repo2.obterTodos();
25             for (PessoaFisica pf : pessoasFisicas) {
26                 pf.exibir();
27             }
28         }
29     }
30 }
```

```
...A] PessoaFisicaRepo.java [-/A] x PessoaJuridicaRepo.java [-/A] x Main.java [-/A] x
Source History
25         for (PessoaFisica pf : pessoasFisicas) {
26             pf.exibir();
27         }
28
29         // Testando repositório de PessoaJuridica
30         PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
31         repo3.inserir(new PessoaJuridica(1, "Empresa A", "12.345.678/0001-99"));
32         repo3.inserir(new PessoaJuridica(2, "Empresa B", "98.765.432/0001-11"));
33
34         System.out.println("Dados de Pessoa Juridica Armazenados.");
35
36         repo3.persistir("pessoasJuridicas.dat");
37
38         System.out.println("Dados de Pessoa Juridica Recuperados.");
39
40         PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
41         repo4.recuperar("pessoasJuridicas.dat");
42         List<PessoaJuridica> pessoasJuridicas = repo4.obterTodos();
43         for (PessoaJuridica pj : pessoasJuridicas) {
44             pj.exibir();
45         }
46     } catch (IOException | ClassNotFoundException e) {
47         e.printStackTrace();
48     }
49 }
50 }
```

Output x

Resultados da Execução



```
run:
Dados de Pessoa Fisica Armazenados.
Dados de Pessoa Fisica Recuperados.
ID: 1, Nome: João
CPF: 123.456.789-00, Idade: 30
ID: 2, Nome: Maria
CPF: 987.654.321-00, Idade: 25
Dados de Pessoa Juridica Armazenados.
Dados de Pessoa Juridica Recuperados.
ID: 1, Nome: Empresa A
CNPJ: 12.345.678/0001-99
ID: 2, Nome: Empresa B
CNPJ: 98.765.432/0001-11
BUILD SUCCESSFUL (total time: 0 seconds)
```

Quais as vantagens e desvantagens do uso de herança?

Vantagens:

- **Reutilização de código:** Classes filhas podem herdar atributos e métodos da classe pai, evitando duplicação de código.
- **Polimorfismo:** Permite tratar objetos de classes derivadas como objetos da classe base, facilitando a programação genérica e flexível.
- **Estrutura Hierárquica:** Permite modelar relações hierárquicas entre classes, refletindo melhor a estrutura do problema.

Desvantagens:

- **Acoplamento Forte:** Pode levar a um alto acoplamento entre classes pai e filhas, tornando o sistema menos flexível às mudanças.
- **Complexidade:** Hierarquias profundas podem se tornar complexas e difíceis de gerenciar.
- **Herança Múltipla Limitada:** Java não suporta herança múltipla de classes, o que pode limitar a flexibilidade em certos cenários.

Por que a interface **Serializable** é necessária ao efetuar persistência em arquivos binários?

A interface **Serializable** é necessária em Java para marcar classes cujas instâncias podem ser convertidas em uma sequência de bytes. Isso é crucial para a persistência em arquivos binários porque:

- **Serialização:** Permite que objetos sejam convertidos em bytes, que podem ser armazenados ou transmitidos.
- **Persistência:** Facilita a gravação de objetos em arquivos binários de forma que possam ser recuperados posteriormente.
- **Transferência de Objetos:** Permite a transferência de objetos entre diferentes plataformas e sistemas.

Como o paradigma funcional é utilizado pela API stream no Java?

A API Stream introduzida no Java 8 permite operações de processamento de dados de forma declarativa e funcional. Ela utiliza conceitos do paradigma funcional, como:

- **Operações em Pipelines:** Streams permitem definir sequências de operações que são aplicadas em um pipeline.
- **Funções de Alta Ordem:** Métodos como `map`, `filter`, `reduce` e `forEach` permitem passar funções como parâmetros.
- **Imutabilidade:** Streams encorajam o uso de operações que não alteram o estado dos dados originais, promovendo o conceito de imutabilidade.

Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?

Ao trabalhar com Java e persistência de dados em arquivos, é comum seguir o padrão de **Serialização e Deserialização**:

- **Serialização:** Converter objetos Java em uma sequência de bytes para armazenamento.
- **Deserialização:** Converter bytes de volta em objetos Java para recuperação.
- **Utilização de Fluxos de Entrada/Saída:** Classes como `ObjectInputStream` e `ObjectOutputStream` são utilizadas para escrever e ler objetos serializados de/arquivos binários.