



# Estácio

Campus: Polo Centro - Jequié (BA)  
Curso: Desenvolvimento fullstack

Aluno: Israel dos Santos Hamdan  
D'Araujo

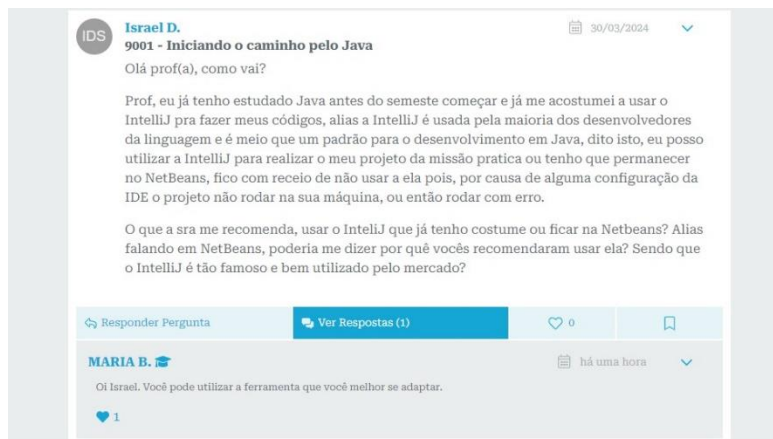
Disciplina: Nível 1 Iniciando caminho pelo Java  
turma: 2023.1  
Semestre: 2024.1

## Título da pratica: Sistema de cadastro

### Objetivos da prática:

1. Utilizar herança e polimorfismo na definição de entidades.
2. Utilizar persistência de objetos em arquivos binários.
3. Implementar uma interface cadastral em modo texto.
4. Utilizar o controle de exceções da plataforma Java.
5. No final do projeto, o aluno terá implementado um sistema cadastral em Java, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários

Neste projeto, foi usado a IDE JetBrains IntelliJ e não o Apache NetBeans como o projeto mencionava, mas a tutora da disciplina me autorizou a usar o IntelliJ



Eu usei o IntelliJ pois penso que ele é melhor de utilizar do que o NetBeans

Para tirar as prints eu usei o Visual Studio Code, pois o mesmo apresenta uma interface melhor para printar os códigos

Link do repositório: <https://github.com/IsraelHamdan/sistemaDeCadastro-.git>

## Estrutura de pastas do projeto:



## Classe Pessoa:

```
package model.entities;

import java.io.Serializable;

public class Pessoa implements Serializable {
    private int id;
    private String nome;

    public Pessoa(int id, String nome) {
        this.id = id;
        this.nome = nome;
    }

    public Pessoa() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String exibir() {
        return "Pessoa{" + "id=" + id + ", nome=" + nome + '}';
    }
}
```

A classe pessoa é responsável por definir as características comuns entre as pessoas físicas e jurídicas, ela é a classe mãe

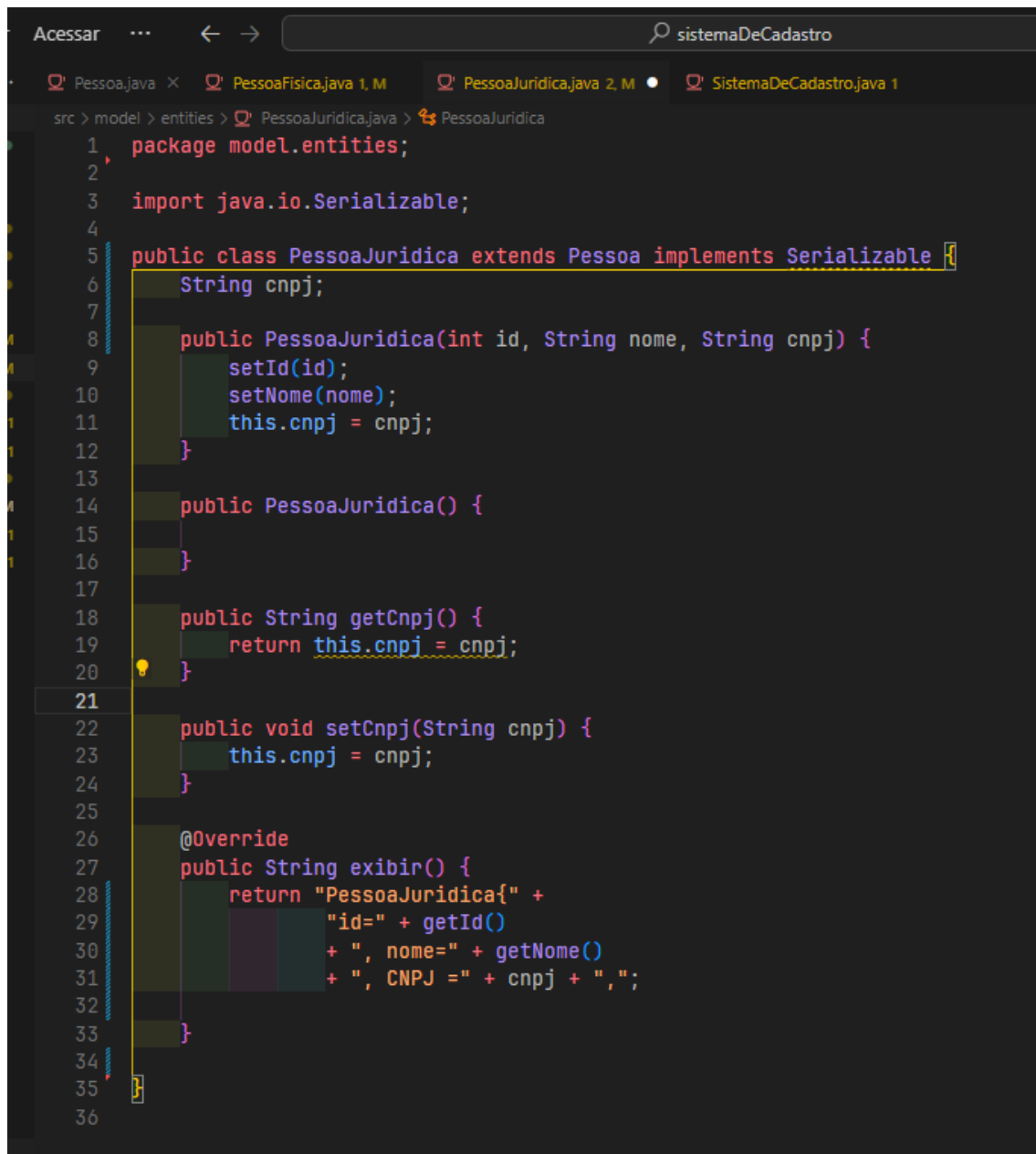
## Classe PessoaFísica

```
src > model > entities > PessoaFisica.java > PessoaFisica > PessoaFisica(int, String, String, int)
1 package model.entities;
2
3 import java.io.Serializable;
4
5 public class PessoaFisica extends Pessoa implements Serializable {
6
7     private String cpf;
8     private int idade;
9
10    public PessoaFisica(int id, String nome, String cpf, int idade) {
11        setId(id);
12        setNome(nome);
13        this.cpf = cpf;
14        this.idade = idade;
15    }
16
17    public String getCpf() {
18        return cpf;
19    }
20
21    public void setCpf(String cpf) {
22        this.cpf = cpf;
23    }
24
25    public int getIdade() {
26        return idade;
27    }
28
29    public void setIdade(int idade) {
30        this.idade = idade;
31    }
32 }
```

```
@Override
public String exibir() {
    return "PessoaFisica{"
        + "id=" + getId()
        + ", nome=" + getNome()
        + ", cpf=" + cpf + ", "
        + "idade=" + idade + '}';
}
}
```

A classe PessoaFísica herda as características da classe mãe Pessoa através da palavra-chave “extends” e chamando a classe Pessoa, e implementa por polimorfismo as características particulares da pessoa física que é CPF e idade

## Classe PessoaJuridica

The image is a screenshot of an IDE window titled 'sistemaDeCadastro'. It shows the source code for the 'PessoaJuridica' class. The code is written in Java and includes package declarations, imports, and class definitions. The class 'PessoaJuridica' extends 'Pessoa' and implements 'Serializable'. It has a 'cnpj' attribute and methods for setting and getting it, as well as an 'exibir()' method for displaying object details. The IDE interface includes a top bar with 'Acessar' and navigation icons, a tab bar with open files, and a breadcrumb trail 'src > model > entities > PessoaJuridica.java > PessoaJuridica'. The code is displayed with syntax highlighting and line numbers from 1 to 36.

```
1 package model.entities;
2
3 import java.io.Serializable;
4
5 public class PessoaJuridica extends Pessoa implements Serializable {
6     String cnpj;
7
8     public PessoaJuridica(int id, String nome, String cnpj) {
9         setId(id);
10        setNome(nome);
11        this.cnpj = cnpj;
12    }
13
14    public PessoaJuridica() {
15    }
16
17
18    public String getCnpj() {
19        return this.cnpj;
20    }
21
22    public void setCnpj(String cnpj) {
23        this.cnpj = cnpj;
24    }
25
26    @Override
27    public String exibir() {
28        return "PessoaJuridica{" +
29            "id=" + getId()
30            + ", nome=" + getNome()
31            + ", CNPJ =" + cnpj + ",";
32    }
33 }
34
35
36
```

A classe PessoaJuridica herda as características da classe mãe e através do polimorfismo implementa as características únicas da PJ que neste caso é o CNPJ

## Classe gerenciadora PessoaFisicaRepo

```
package model.managers;

import model.entities.PessoaFisica;
import java.util.*;
import java.io.*;

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoasFisicas;

    public PessoaFisicaRepo() {
        this.pessoasFisicas = new ArrayList<>();
    }

    public void inserir(PessoaFisica pessoaFisica) {
        this.pessoasFisicas.add(pessoaFisica);
    }

    public void alterar(PessoaFisica pessoaFisica) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            PessoaFisica pessoaFisicaAtual = pessoasFisicas.get(i);
            if (pessoaFisicaAtual.getId() == pessoaFisica.getId()) {
                pessoasFisicas.set(i, pessoaFisica);
            }
        }
    }

    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
            }
        }
        return null;
    }
}
```

```
    public ArrayList<PessoaFisica> obterTodos() {
        return pessoasFisicas;
    }

    public void excluir(int id) {
        Iterator<PessoaFisica> iterator = pessoasFisicas.iterator();
        while (iterator.hasNext()) {
            PessoaFisica pessoa = iterator.next();
            if (pessoa.getId() == id) {
                iterator.remove();
                return;
            }
        }
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(
            new FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(pessoasFisicas);
        }
    }

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(
            new FileInputStream(nomeArquivo))) {
            pessoasFisicas = (ArrayList<PessoaFisica>) inputStream.readObject();
        }
    }
}
```

A classe PessoaFisicaRepo herda as características da classe PessoaFisica e faz todo o gerenciamento de criar e armazenar os dados da pessoa física

## Classe gerenciadora PessoaJuridicaRepo

```
1 package model.managers;
2
3 import model.entities.PessoaJuridica;
4 import java.util.*;
5 import java.io.*;
6
7 public class PessoaJuridicaRepo {
8
9     private ArrayList<PessoaJuridica> pessoasJuridicas;
10
11     public PessoaJuridicaRepo() {
12         this.pessoasJuridicas = new ArrayList<>();
13     }
14
15     public void inserir(PessoaJuridica pessoaJuridica) {
16         this.pessoasJuridicas.add(pessoaJuridica);
17     }
18
19     public void alterar(PessoaJuridica pessoaJuridica) {
20         for (int p = 0; p < pessoasJuridicas.size(); p++) {
21             PessoaJuridica pessoaJuridicaAtual = pessoasJuridicas.get(p);
22             if (pessoaJuridicaAtual.getId() == pessoaJuridica.getId()) {
23                 pessoasJuridicas.set(p, pessoaJuridica);
24             }
25         }
26     }
27
28     public PessoaJuridica obter(int id) {
29         for (int p = 0; p < pessoasJuridicas.size(); p++) {
30             PessoaJuridica pessoaJuridica = pessoasJuridicas.get(p);
31             if (pessoaJuridica.getId() == id) {
32                 return pessoaJuridica;
33             }
34         }
35         return null;
36     }
37 }
```

```
    public void excluir(int id) {
        Iterator<PessoaJuridica> iterator = pessoasJuridicas.iterator();
        while (iterator.hasNext()) {
            PessoaJuridica pessoa = iterator.next();
            if (pessoa.getId() == id) {
                iterator.remove();
                return;
            }
        }
    }

    public ArrayList<PessoaJuridica> obterTodos() {
        return pessoasJuridicas;
    }

    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            outputStream.writeObject(pessoasJuridicas);
        }
    }

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoasJuridicas = (ArrayList<PessoaJuridica>) inputStream.readObject();
        }
    }
}
```

A classe PessoaJuridicaRepo herda as características da classe PessoaJuridica e faz todo o gerenciamento de criar e armazenar os dados da pessoa física



## Classe main

```
Systemadecadastro > SistemaDeCadastro.java > SistemaDeCadastro
package sistemadecadastro;

public class SistemaDeCadastro {
    private static String nameFile;
    Run | Debug
    public static void main(String[] args) {
        // Cadastrando uma PessoaFisica
        CadastroPessoaFisica cadastroPf = new CadastroPessoaFisica();
        cadastroPf.createPf();
        cadastroPf.createfilePf();
        cadastroPf.recoverFilePf();
        cadastroPf.showPf();

        // Cadastrando uma PessoaJuridica
        CadastroPessoaJuridica cadastroPj = new CadastroPessoaJuridica();
        cadastroPj.createPj();
        cadastroPj.createFilePj();
        cadastroPj.recoverFilePj();
        cadastroPj.showPj();
    }
}
```

A classe main está chamando os métodos create, createFile, recoverFile e showFile das pessoas físicas e jurídicas, o motivo de ter sido feito dessa maneira é melhorar a organização do código, pois na documentação do Sway fazia com que houvesse muita repetição de código e a classe main iria ficar muito grande e bem bagunçada, já que toda criação das pessoas físicas e jurídicas ficava nela, vou deixar as prints da classe CadastroPessoaJuridica e da classe CadastroPessoaFisica.

## Classe CadastroPessoaJuridica

```
package sistemadecadastro;

import model.entities.PessoaJuridica;
import model.managers.PessoaJuridicaRepo;

import java.io.IOException;
import java.util.ArrayList;

public class CadastroPessoaJuridica extends PessoaJuridicaRepo {
    PessoaJuridicaRepo pjRepo1 = new PessoaJuridicaRepo();
    PessoaJuridicaRepo pjRepo2 = new PessoaJuridicaRepo();

    protected void createPj() {
        PessoaJuridica pj1 = new PessoaJuridica(id: 1, nome: "XPTO", cnpj: "12.345.678/0001-00");
        PessoaJuridica pj2 = new PessoaJuridica(id: 2, nome: "Lpto", cnpj: "13.355.668/0702-01");

        pjRepo1.inserir(pj1);
        pjRepo1.inserir(pj2);
    }

    String nameFilePj = "pessoas_juridicas_data.dat";

    protected void createFilePj() {
        try {
            pjRepo1.persistir(nameFilePj);
            System.out.println("Dados inseridos com sucesso no arquivo: " + nameFilePj);
        } catch (IOException e) {
            System.out.println("Erro ao persistir os dados no arquivo: " + nameFilePj);
            e.printStackTrace();
        }
    }
}
```

```
    protected void recoverFilePj() {
        try {
            pjRepo2.recuperar(nameFilePj);
            System.out.println("Dados obtidos com sucesso do arquivo: " + nameFilePj);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar os dados do arquivo: " + nameFilePj);
            e.printStackTrace();
        }
    }

    public void showPj() {
        ArrayList<PessoaJuridica> pessoasRecuperada = pjRepo2.obterTodos();
        for (PessoaJuridica pessoa : pessoasRecuperada) {
            System.out.println("Id: " + pessoa.getId() + "\n"
                + "Nome: " + pessoa.getNome() + "\n"
                + "CNPJ: " + pessoa.getCnpj() + "\n");
        }
    }
}
```

Para diferenciar o nome das funções das classes CadastroPessoaFisica e CadastroPessoaJuridica eu coloquei no final de cada nome de função Pj para representar que é da pessoa jurídica e Pf que é para a pessoa física

## Classe CadastroPessoaFisica

```
package sistemadecadastro;

import model.entities.PessoaFisica;
import model.managers.PessoaFisicaRepo;

import java.io.IOException;
import java.util.ArrayList;

public class CadastroPessoaFisica extends PessoaFisicaRepo {
    PessoaFisicaRepo pfRpo1 = new PessoaFisicaRepo();
    PessoaFisicaRepo pfRpo2 = new PessoaFisicaRepo();
    static String nameFile = "pessoas_fisicas_data.dat";

    protected void createPf() {
        PessoaFisica p1 = new PessoaFisica(id:1, nome:"Fulano", cpf:"048.542.455-82", idade:20);
        PessoaFisica p2 = new PessoaFisica(id:2, nome:"Beltrano", cpf:"186.856.458-58", idade:22);

        pfRpo1.inserir(p1);
        pfRpo1.inserir(p2);
    }

    protected void createfilePf() {
        try {
            pfRpo1.persistir(nameFile);
            System.out.println("Dados inseridos com sucesso no arquivo: " + nameFile);
        } catch (IOException e) {
            System.out.println("Erro ao persistir os dados no arquivo: " + nameFile);
            e.printStackTrace();
        }
    }

    protected void recoverFilePf() {
        try {
            pfRpo2.recuperar(nameFile);
            System.out.println("Dados obtidos com sucesso do arquivo:" + nameFile);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar os dados do arquivo: "
                               + nameFile);
            e.printStackTrace();
        }
    }

    protected void showPf() {
        ArrayList<PessoaFisica> pessoasRecuperada = pfRpo2.obterTodos();
        for (PessoaFisica pessoa : pessoasRecuperada) {
            System.out.println("Id:" + pessoa.getId() + "\n"
                               + "Nome:" + pessoa.getNome() + "\n"
                               + "CPF:" + pessoa.getCpf() + "\n"
                               + "Idade:" + pessoa.getIdade() + "\n");
        }
    }
}
```

## Resultados da pratica

```
Dados inseridos com sucesso no arquivo: pessoas_fisicas_data.dat
Dados obtidos com sucesso do arquivo: pessoas_fisicas_data.dat
Id:1
Nome:Fulano
CPF:048.542.455-82
Idade:20

Id:2
Nome:Beltrano
CPF:186.856.458-58
Idade:22

Dados inseridos com sucesso no arquivo: pessoas_juridicas_data.dat
Dados obtidos com sucesso do arquivo: pessoas_juridicas_data.dat
Id:1
Nome:XPT0
CNPJ:12.345.678/0001-00

Id:2
Nome:Lpto
CNPJ:13.355.668/0702-01
```

## Análise e conclusão:

- a. O uso de herança facilita muito a criação de uma nova classe pois possibilita que atributos comuns a outras classes fiquem dentro da classe mãe além de que reduz a quantidade de código escrito. A desvantagem fica por conta de uma limitação de uma classe poder ser herdeira de somente uma classe.
- b. Sem implementar a interface `Serializable` seria impossível pra JVM (Java Virtual Machine) reconhecer que o objeto deveria converter os arquivos em bytes.
- c. Usando expressões lambda e interfaces funcionais para manipular as coleções de elementos de forma eficiente, como mostra o exemplo abaixo retirado de uma pesquisa no ChatGpt

```

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Integer> numeros = new ArrayList<>();
        numeros.add(1);
        numeros.add(2);
        numeros.add(3);
        numeros.add(4);
        numeros.add(5);

        // Utilizando uma expressão lambda para filtrar apenas os números pares
        numeros.stream()
            .filter(n -> n % 2 == 0)
            .forEach(System.out::println);
    }
}

```

- 1.
- d. No Java usamos o padrão DAO (Data Access Object) onde ele encapsula as operações dentro do arquivo e fornece métodos para manipular os dados armazenados. Por exemplo a classe PessoaFisicaRepo implementa as manipulações do DAO para criação da pessoa física, o mesmo também acontece na classe PessoaJuridicaRepo