



FACULDADE ESTACIO  
CURSO TECNÓLOGO EM DESENVOLVIMENTO FULL STACK

**RAFAEL VALVERDE FONSECA**

**MISSÃO PRÁTICA – NÍVEL 1 – MUNDO 3**  
**Iniciando o caminho pelo Java**

Serrinha - BA  
2024

**RAFAEL VALVERDE FONSECA**

**MISSÃO PRÁTICA – NÍVEL 1 – MUNDO 3**

**Iniciando o caminho pelo Java**

Trabalho apresentado à disciplina Iniciando o caminho pelo Java do Curso Tecnólogo em Desenvolvimento Full Stack, período 2024.4 Flex, como requisito parcial do relatório de acompanhamento.

Tutoria: Maria Manso

Serrinha – BA

2024

## OBJETIVOS

Este relatório apresenta a composição do trabalho proposto para o Nível 1: Iniciando o cainho pelo Java o qual está contido no semestre letivo no período 2024, o qual apresenta todos os códigos solicitados, resultados da execução desses códigos e descrição de avaliação sobre o tema abordado, respondendo perguntas propostas pelo tutor.

**Palavras-chave:** Java, herança, interface Serializable, classes, objetos.

## SUMÁRIO

<b>1</b>	<b>CÓDIGOS .....</b>	<b>9</b>
<b>2</b>	<b>RESULTADOS DE EXECUÇÃO DE CÓDIGOS.....</b>	<b>15</b>
<b>3</b>	<b>ANÁLISE .....</b>	<b>16</b>

## 1 CÓDIGOS

```
// CadastroPOO
package cadastrapoo;

import model.*;
import java.io.IOException;

public class CadastroPOO {
    public static void main(String[] args) {
        try {
            // Repositório de pessoas físicas (repo1)
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

            // Adicionar duas pessoas físicas
            repo1.inserir(new PessoaFisica(1, "Joao Silva", "123.456.789-00", 25));
            repo1.inserir(new PessoaFisica(2, "Maria Oliveira", "987.654.321-00", 30));

            // Persistir dados no arquivo fixo
            String arquivoPessoasFisicas = "pessoasFisicas.dat";
            repo1.persistir(arquivoPessoasFisicas);

            // Outro repositório de pessoas físicas (repo2)
            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

            // Recuperar dados do arquivo
            repo2.recuperar(arquivoPessoasFisicas);

            // Exibir dados recuperados de pessoas físicas
            System.out.println("Pessoas Fisicas Armazenados.");
            for (PessoaFisica pf : repo2.obterTodos()) {
                pf.exibir();
            }

            // Repositório de pessoas jurídicas (repo3)
            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

            // Adicionar duas pessoas jurídicas
            repo3.inserir(new PessoaJuridica(1, "Empresa ABC", "12.345.678/0001-00"));
            repo3.inserir(new PessoaJuridica(2, "Corporacao XYZ", "98.765.432/0001-99"));

            // Persistir dados no arquivo fixo
            String arquivoPessoasJuridicas = "pessoasJuridicas.dat";
            repo3.persistir(arquivoPessoasJuridicas);

            // Outro repositório de pessoas jurídicas (repo4)
            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

            // Recuperar dados do arquivo
            repo4.recuperar(arquivoPessoasJuridicas);

            // Exibir dados recuperados de pessoas jurídicas
            System.out.println("\nDados de Pessoas Juridicas Armazenados:");
            for (PessoaJuridica pj : repo4.obterTodos()) {
                pj.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
// Pessoa
package model;

import java.io.Serializable;

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

    // Construtor padrão
    public Pessoa() {}

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

    // Método exibir
    public void exibir() {
        System.out.println("ID: " + id + ", Nome: " + nome);
    }

    // Getters e Setters
    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;
    }
}
```

```
// PessoaFisica
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private String cpf;
    private int idade;

    // Construtor padrão
    public PessoaFisica() {}

    // Construtor completo
    public PessoaFisica(int id, String nome, String cpf, int idade) {
        super(id, nome);
        this.cpf = cpf;
        this.idade = idade;
    }

    // Método exibir (polimórfico)
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + cpf + ", Idade: " + idade);
    }

    // Getters e Setters
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }
}
```

```
// PessoaJuridica
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {}

    // Construtor completo
    public PessoaJuridica(int id, String nome, String cnpj) {
        super(id, nome);
        this.cnpj = cnpj;
    }

    // Método exibir (polimórfico)
    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }

    // Getters e Setters
    public String getCnpj() {
        return cnpj;
    }

    public void setCnpj(String cnpj) {
        this.cnpj = cnpj;
    }
}
```



```

// PessoaFisicaRepo
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {
    private List<PessoaFisica> lista = new ArrayList<>();

    // Método para inserir uma nova PessoaFisica
    public void inserir(PessoaFisica pessoaFisica) {
        lista.add(pessoaFisica);
    }

    // Método para alterar uma PessoaFisica existente
    public void alterar(PessoaFisica pessoaFisica) {
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i).getId() == pessoaFisica.getId()) {
                lista.set(i, pessoaFisica);
                return;
            }
        }
    }

    // Método para excluir uma PessoaFisica pelo ID
    public void excluir(int id) {
        lista.removeIf(pessoa -> pessoa.getId() == id);
    }

    // Método para obter uma PessoaFisica pelo ID
    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoa : lista) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null;
    }

    // Método para obter todas as PessoasFisicas
    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(lista);
    }

    // Método para persistir os dados em arquivo
    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(nomeArquivo))) {
            oos.writeObject(lista);
        }
    }

    // Método para recuperar os dados de um arquivo
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            lista = (List<PessoaFisica>) ois.readObject();
        }
    }
}

```

```

// PessoaJuridicaRepo
package model;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaRepo {
    private List<PessoaJuridica> lista = new ArrayList<>();

    // Método para inserir uma nova PessoaJuridica
    public void inserir(PessoaJuridica pessoaJuridica) {
        lista.add(pessoaJuridica);
    }

    // Método para alterar uma PessoaJuridica existente
    public void alterar(PessoaJuridica pessoaJuridica) {
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i).getId() == pessoaJuridica.getId()) {
                lista.set(i, pessoaJuridica);
                return;
            }
        }
    }

    // Método para excluir uma PessoaJuridica pelo ID
    public void excluir(int id) {
        lista.removeIf(pessoa -> pessoa.getId() == id);
    }

    // Método para obter uma PessoaJuridica pelo ID
    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : lista) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null;
    }

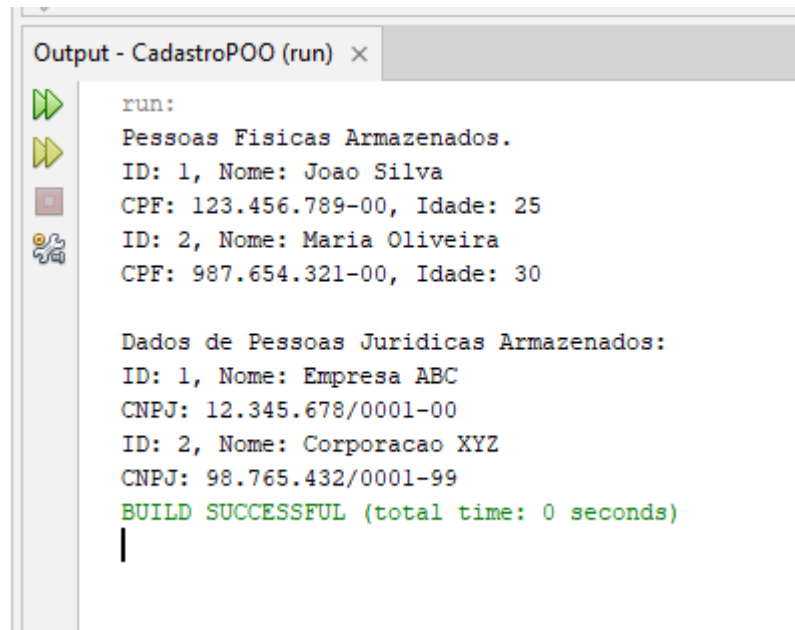
    // Método para obter todas as PessoasJuridicas
    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(lista);
    }

    // Método para persistir os dados em arquivo
    public void persistir(String nomeArquivo) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream(nomeArquivo))) {
            oos.writeObject(lista);
        }
    }

    // Método para recuperar os dados de um arquivo
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            lista = (List<PessoaJuridica>) ois.readObject();
        }
    }
}

```

## 2 RESULTADOS DE EXECUÇÃO DE CÓDIGOS



```
run:
Pessoas Fisicas Armazenados.
ID: 1, Nome: Joao Silva
CPF: 123.456.789-00, Idade: 25
ID: 2, Nome: Maria Oliveira
CPF: 987.654.321-00, Idade: 30

Dados de Pessoas Juridicas Armazenados:
ID: 1, Nome: Empresa ABC
CNPJ: 12.345.678/0001-00
ID: 2, Nome: Corporacao XYZ
CNPJ: 98.765.432/0001-99
BUILD SUCCESSFUL (total time: 0 seconds)
|
```

### 3 ANÁLISE

#### **Quais as vantagens e desvantagens do uso de herança?**

Como vantagens do uso de herança, podemos citar o reuso do código. Herança ajuda a organizar o código em uma estrutura lógica de hierarquia, facilitando a compreensão do relacionamento entre as classes. Facilidade de manutenção e alterações, também são vantagens do uso de herança, uma vez que precisamos modificar apenas um arquivo afetando todas as subclasses.

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

É necessária para persistência em arquivos binários porque ela instrui a máquina virtual Java a permitir que objetos de uma classe sejam convertidos em uma sequência de bytes, controlando o acesso aos campos de uma classe, garantindo que os dados sensíveis só sejam serializados se explicitamente programados para isso.

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

Esse paradigma é muito utilizado pela API de Streams do Java para trabalhar com coleções de dados de maneira mais declarativa, concisa e eficiente. Ele permite escrever operações de processamento de dados como sequências de transformações. A API de Streams não altera os dados originais. Em vez disso, ela cria novos fluxos com os resultados das operações.