



Estácio

Nome do Campus: BANCÁRIOS.

Nome do Curso: Desenvolvimento Full Stack.

Nome da Disciplina: Nível 1: Iniciando o Caminho Pelo Java.

Número da Turma: 202403781506.

Semestre letivo: 3º período - semestre 2025.1.

Nome dos integrantes da Prática: Marcelo Picado Schulze.

Repositório no GIT: <https://github.com/darklaw53/AtividadeJava.git>

Missão Prática | Nível 1 | Mundo 3

<https://github.com/darklaw53/AtividadeJava.git>

Objetivos da prática:

- Utilizar herança e polimorfismo na definição de entidades.
- Utilizar persistência de objetos em arquivos binários.
- Implementar uma interface cadastral em modo texto.
- Utilizar o controle de exceções da plataforma Java.
- 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.

Códigos usados:

```
package cadastrapoo;

import java.io.IOException;
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;

public class CadastroPOO
```

```

{
    public static void main(String[] args)
    {
        try
        {
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
            repo1.inserir(new PessoaFisica(1, "Ana", "1111111111", 25));
            repo1.inserir(new PessoaFisica(2, "Carlos", "2222222222", 52));

            String arquivoPF = "pessoas_fisicas.dat";
            repo1.persistir(arquivoPF);
            System.out.println("Dados de Pessoa Física Armazenados:");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar(arquivoPF);
            System.out.println("Dados de Pessoa Física Recuperados:");

            for (PessoaFisica pf : repo2.obterTodos())
            {
                System.out.println("Id: " + pf.getId());
                System.out.println("Nome: " + pf.getNome());
                System.out.println("CPF: " + pf.getCpf());
                System.out.println("Idade: " + pf.getIdade());
            }

            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
            repo3.inserir(new PessoaJuridica(3, "XPTO Sales", "3333333333333333"));
            repo3.inserir(new PessoaJuridica(4, "XPTO Solutions", "4444444444444444"));

            String arquivoPJ = "pessoas_juridicas.dat";
            repo3.persistir(arquivoPJ);
            System.out.println("Dados de Pessoa Jurídica Armazenados:");

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar(arquivoPJ);
            System.out.println("Dados de Pessoa Jurídica Recuperados:");

            for (PessoaJuridica pj : repo4.obterTodos())
            {
                System.out.println("Id: " + pj.getId());
                System.out.println("Nome: " + pj.getNome());
                System.out.println("CNPJ: " + pj.getCnpj());
            }
        }
        catch (IOException | ClassNotFoundException e)
        {
            System.err.println("Erro ao manipular arquivos: " + e.getMessage());
        }
    }
}

```

Package model;

```
import java.io.Serializable;

public class Pessoa implements Serializable
{
    private static final long serialVersionUID = 1L;

    private int id;
    private String nome;

    public Pessoa() {}

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

    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;
    }
}
```

Package model;

```
public class PessoaFisica extends Pessoa
{
    private static final long serialVersionUID = 1L;

    private String cpf;
    private int idade;

    public PessoaFisica() {}

    public PessoaFisica(int id, String nome, String cpf, int idade)
    {
```

```

    super(id, nome);
    this.cpf = cpf;
    this.idade = idade;
}

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;
}
}

```

Package model;

```

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

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

    public void inserir(PessoaFisica pessoa)
    {
        pessoas.add(pessoa);
    }

    public void alterar(PessoaFisica pessoa)
    {
        for(int i = 0; i < pessoas.size(); i++)
        {
            if (pessoas.get(i).getId() == pessoa.getId())
            {
                pessoas.set(i, pessoa);
                return;
            }
        }
    }
}

```

```

public void excluir (int id)
{
    pessoas.removeIf(p → p.getId() == id);
}

public PessoaFisica obter (int id)
{
    return pessoas.stream().filter(p → p.getId() == id).findFirst().orElse(null);
}

public List<PessoaFisica> obterTodos()
{
    return new ArrayList<>(pessoas);
}

public void persistir(String arquivo) throws IOException
{
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(arquivo)))
    {
        oos.writeObject(pessoas);
    }
}

@SuppressWarnings("unchecked")
public void recuperar(String arquivo) throws IOException, ClassNotFoundException
{
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(arquivo)))
    {
        pessoas = (List<PessoaFisica>) ois.readObject();
    }
}
}

```

Package model;

```

public class PessoaJuridica extends Pessoa
{
    private static final long serialVersionUID = 1L;

    private String cnpj;

    public PessoaJuridica() {}

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

    public String getCnpj ()
    {

```

```

        return cnpj;
    }

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

```

Package model;

```

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

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

    public void inserir( PessoaJuridica empresa)
    {
        empresas.add(empresa);
    }

    public void alterar(PessoaJuridica empresa)
    {
        for(int i = 0; i < empresas.size(); i++)
        {
            if (empresas.get(i).getId() == empresa.getId())
            {
                empresas.set(i, empresa);
                return;
            }
        }
    }

    public void excluir (int id)
    {
        empresas.removeIf(p -> p.getId() == id);
    }

    public PessoaJuridica obter (int id)
    {
        return empresas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
    }

    public List<PessoaJuridica> obterTodos()
    {
        return new ArrayList<>(empresas);
    }

    public void persistir(String arquivo) throws IOException

```

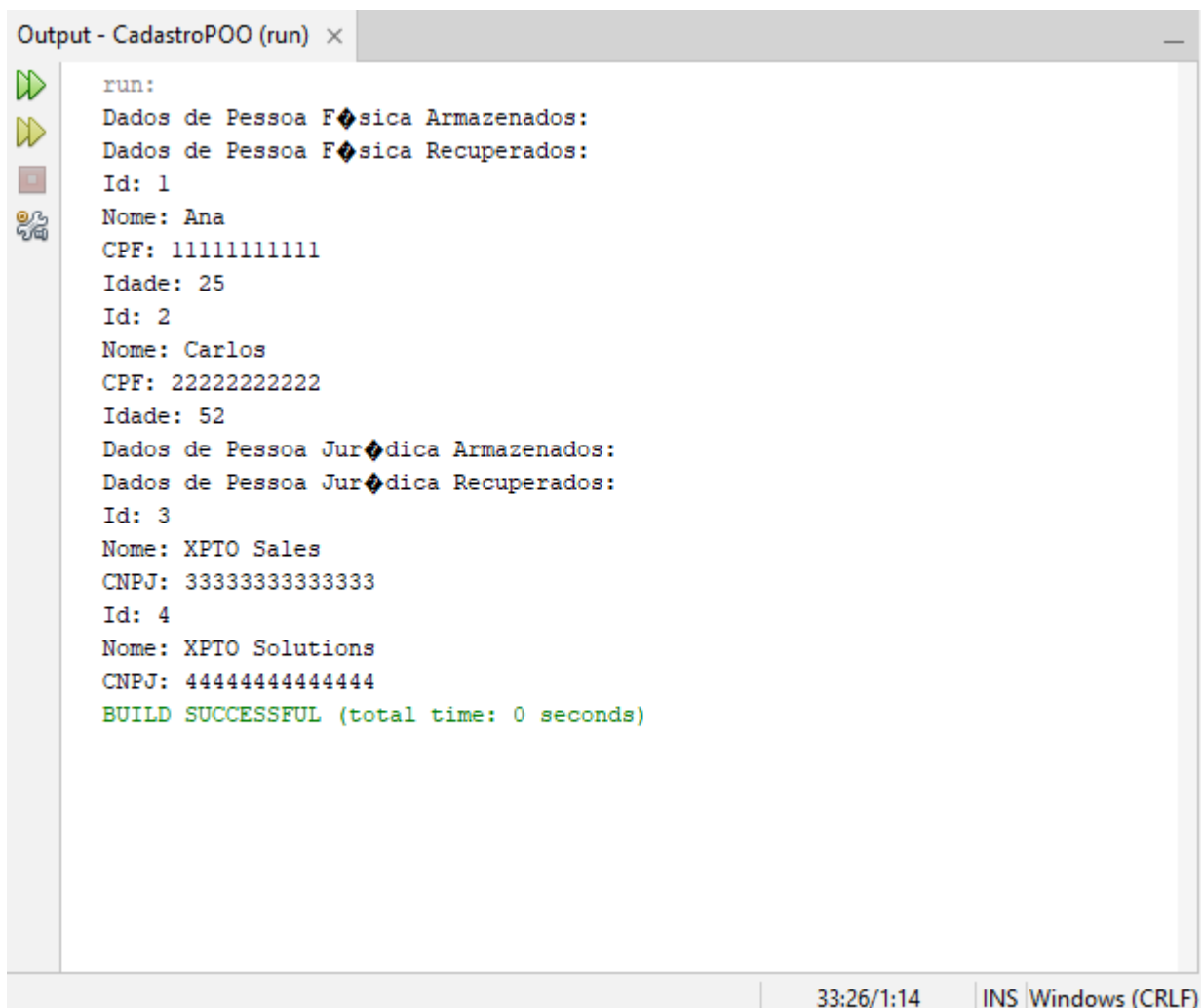
```

{
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(arquivo)))
    {
        oos.writeObject(empresas);
    }
}

@SuppressWarnings("unchecked")
public void recuperar(String arquivo) throws IOException, ClassNotFoundException
{
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(arquivo)))
    {
        empresas = (List<PessoaJuridica>) ois.readObject();
    }
}
}

```

Output da execução:



```

run:
Dados de Pessoa Física Armazenados:
Dados de Pessoa Física Recuperados:
Id: 1
Nome: Ana
CPF: 111111111111
Idade: 25
Id: 2
Nome: Carlos
CPF: 222222222222
Idade: 52
Dados de Pessoa Jurídica Armazenados:
Dados de Pessoa Jurídica Recuperados:
Id: 3
Nome: XPTO Sales
CNPJ: 33333333333333
Id: 4
Nome: XPTO Solutions
CNPJ: 4444444444444444
BUILD SUCCESSFUL (total time: 0 seconds)

```

Análise e Conclusão:

Quais as vantagens e desvantagens do uso de herança?

Vantagens:

- Evita repetição de código.
- Permite a criação de códigos mais genéricos usando o polimorfismo.

Desvantagens:

- Se uma superclasse for alterada, todas as subclasses podem ser afetadas.
- Requer planejamento para evitar ter que constantemente revisar classes quando alguma das classes tiver requerimentos diferentes.

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

- A interface Serializable é usada para sinalizar para o Java que aquele arquivo está preparado para ser convertido para uma sequência de bytes (serialização) para ser salvo no sistema.

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

- A API stream no Java utiliza o paradigma funcional para manipular coleções utilizando funções como filter(), map() e reduce() para transformar e combinar dados, sem modificar a coleção original.

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

- DAO (Data Access Object).