Nome do Campus: Polo Cruzeiro-Sp

Nome do Curso: Desenvolvimento FullStack

Nome da Disciplina: Iniciando o caminho pelo Java

Número da Turma: 2025.1

Semestre Letivo: Primeiro Semestre

Nome do Aluno: Moniza de Oliveira Silva Santos Pelegrini

Matrícula: 202401190829

Título da Prática

Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java.

Objetivo da Prática

Utilizar herança e polimorfismo na definição de entidades:

Criar uma hierarquia de classes (Pessoa, PessoaFisica, PessoaJuridica) para demonstrar herança e polimorfismo.

Utilizar persistência de objetos em arquivos binários:

Implementar métodos para salvar e recuperar dados em arquivos binários usando serialização.

Implementar uma interface cadastral em modo texto:

Desenvolver um menu interativo para interação com o usuário via linha de comando.

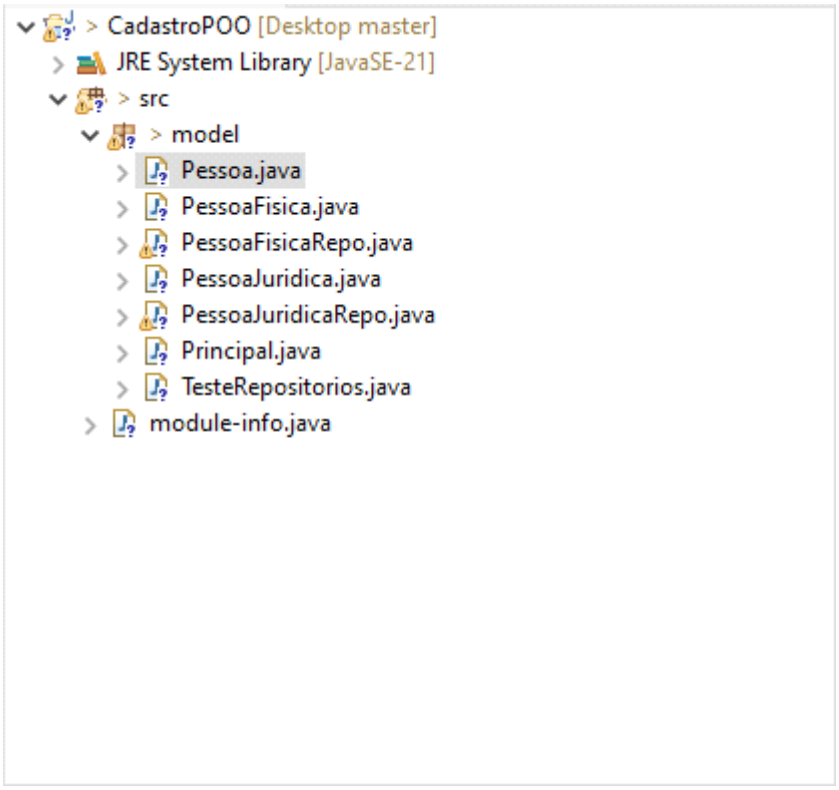
Utilizar o controle de exceções da plataforma Java:

Tratar possíveis erros durante operações como persistência e recuperação de dados.

Primeiro Procedimento:

Criação das entidades e sistema de Persistência

Códigos utlizados neste roteiro:



Classe Pessoa:

Pessoa.java

```java
package model;

import java.io.Serializable;

public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L; // Identificador para serialização

    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);
        System.out.println("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;
    }
}

```

Classe Pessoa Fisica:

PessoaFisica.java

```java
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L; // Identificador para serialização

    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); // Chama o construtor da superclasse
        this.cpf = cpf;
        this.idade = idade;
    }

    // Sobrescrita do método exibir
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da superclasse
        System.out.println("CPF: " + cpf);
        System.out.println("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;
    }
```

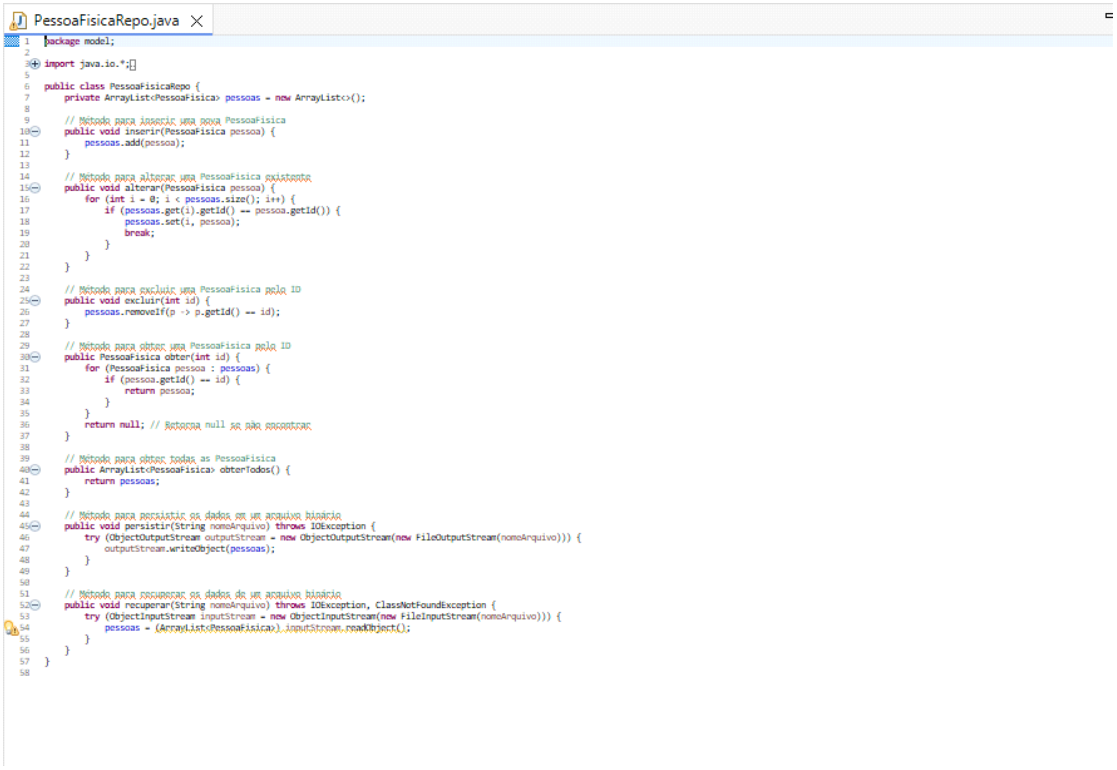
Classe Pessoa Juridica:

PessoaJuridica.java

```java
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L; // Identificador para serialização

    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
    }

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

    // Sobrescrita do método exibir
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da superclasse
        System.out.println("CNPJ: " + cnpj);
    }

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

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

Classe Pessoa Fisica Repo:

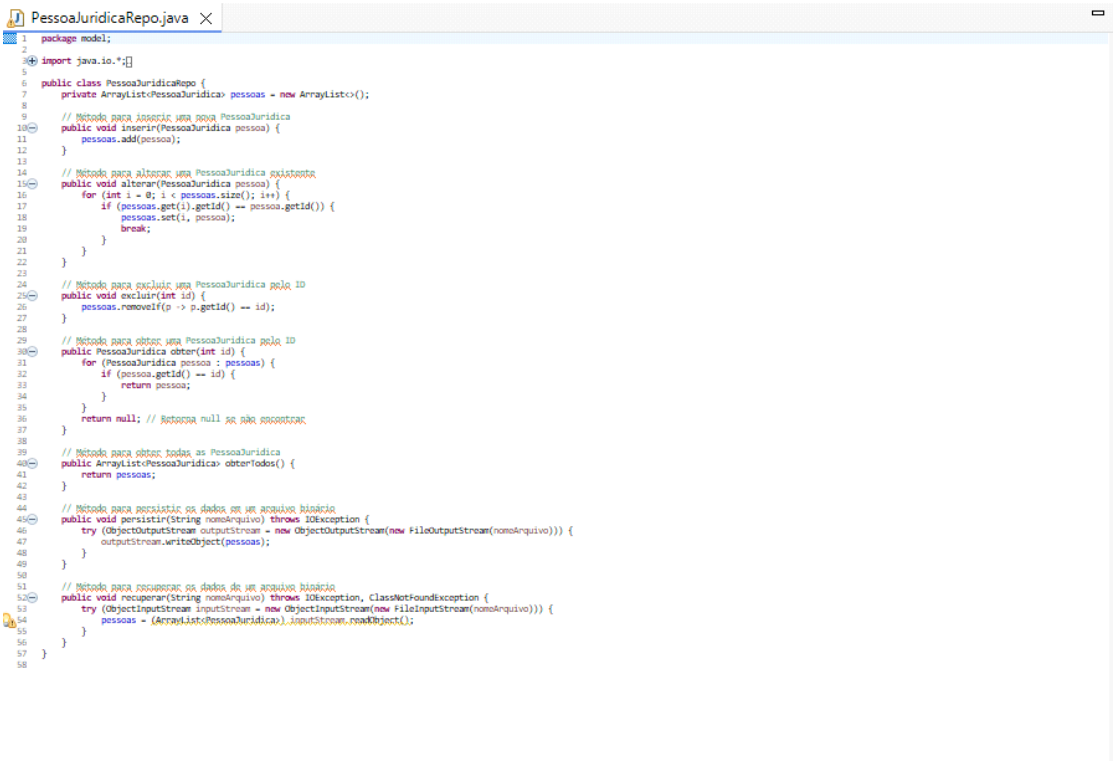
PessoaFisicaRepo.java



```java
package model;

import java.io.*;

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

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

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

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

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

    // Método para obter todas as PessoaFisica
    public ArrayList<PessoaFisica> obterTodos() {
        return pessoas;
    }

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

    // Método para recuperar os dados de um arquivo binário
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoas = (ArrayList<PessoaFisica>) inputStream.readObject();
        }
    }
}

```

Classe Pessoa Juridica Repo:

PessoaJuridicaRepo.java



```java
package model;

import java.io.*;

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

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

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

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

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

    // Método para obter todas as PessoaJuridica
    public ArrayList<PessoaJuridica> obterTodos() {
        return pessoas;
    }

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

    // Método para recuperar os dados de um arquivo binário
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoas = (ArrayList<PessoaJuridica>) inputStream.readObject();
        }
    }
}

```

Classe Principal:

Principal.java

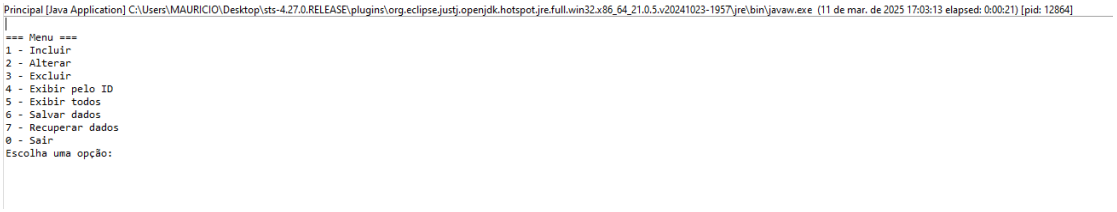
```java
package model;

import java.util.Scanner;

public class Principal {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
        PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();

        int opcao;
        do {
            // Exibir o menu
            System.out.println("\n=== Menu ===");
            System.out.println("1 - Incluir");
            System.out.println("2 - Alterar");
            System.out.println("3 - Excluir");
            System.out.println("4 - Exibir pelo ID");
            System.out.println("5 - Exibir todos");
            System.out.println("6 - Salvar dados");
            System.out.println("7 - Recuperar dados");
            System.out.println("0 - Sair");
            System.out.print("Escolha uma opção: ");
            opcao = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer

            try {
                switch (opcao) {
                    case 1: // Incluir
                        incluir(scanner, repoFisica, repoJuridica);
                        break;
                    case 2: // Alterar
                        alterar(scanner, repoFisica, repoJuridica);
                        break;
                    case 3: // Excluir
                        excluir(scanner, repoFisica, repoJuridica);
                        break;
                    case 4: // Exibir pelo ID
                        exibirPorId(scanner, repoFisica, repoJuridica);
                        break;
                    case 5: // Exibir todos
                        exibirTodos(scanner, repoFisica, repoJuridica);
                        break;
                    case 6: // Salvar dados
                        salvarDados(scanner, repoFisica, repoJuridica);
                        break;
                    case 7: // Recuperar dados
                        recuperarDados(scanner, repoFisica, repoJuridica);
                        break;
                    case 0: // Sair
                        System.out.println("Finalizando o sistema...");
                        break;
                    default:
                        System.out.println("Opção inválida. Tente novamente.");
                }
            } catch (Exception e) {
                System.out.println("Erro: " + e.getMessage());
            }

        } while (opcao != 0);

        scanner.close();
    }
```

Principal.java

```java
    }

    // Método para incluir uma nova pessoa
    private static void incluir(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        if (tipo == 1) {
            System.out.print("ID: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Nome: ");
            String nome = scanner.nextLine();
            System.out.print("CPF: ");
            String cpf = scanner.nextLine();
            System.out.print("Idade: ");
            int idade = scanner.nextInt();
            repoFisica.inserir(new PessoaFisica(id, nome, cpf, idade));
        } else if (tipo == 2) {
            System.out.print("ID: ");
            int id = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Nome: ");
            String nome = scanner.nextLine();
            System.out.print("CNPJ: ");
            String cnpj = scanner.nextLine();
            repoJuridica.inserir(new PessoaJuridica(id, nome, cnpj));
        } else {
            System.out.println("Tipo inválido.");
        }
    }
```

```java
    // Método para alterar uma pessoa existente
    private static void alterar(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        System.out.print("ID: ");
        int id = scanner.nextInt();
        scanner.nextLine();

        if (tipo == 1) {
            PessoaFisica pessoa = repoFisica.obter(id);
            if (pessoa != null) {
                System.out.println("Dados atuais:");
                pessoa.exibir();
                System.out.print("Novo nome: ");
                String nome = scanner.nextLine();
                System.out.print("Novo CPF: ");
                String cpf = scanner.nextLine();
                System.out.print("Nova idade: ");
                int idade = scanner.nextInt();
                pessoa.setNome(nome);
                pessoa.setCpf(cpf);
                pessoa.setIdade(idade);
                repoFisica.alterar(pessoa);
                System.out.println("Pessoa física alterada com sucesso.");
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        } else if (tipo == 2) {
            PessoaJuridica pessoa = repoJuridica.obter(id);
            if (pessoa != null) {
                System.out.println("Dados atuais:");
                pessoa.exibir();
                System.out.print("Novo nome: ");
                String nome = scanner.nextLine();
                System.out.print("Novo CNPJ: ");
                String cnpj = scanner.nextLine();
                pessoa.setNome(nome);
                pessoa.setCnpj(cnpj);
                repoJuridica.alterar(pessoa);
                System.out.println("Pessoa jurídica alterada com sucesso.");
            } else {
                System.out.println("Pessoa jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }


    // Método para excluir uma pessoa
    private static void excluir(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        System.out.print("ID: ");
        int id = scanner.nextInt();

        if (tipo == 1) {
            repoFisica.excluir(id);
            System.out.println("Pessoa física excluída com sucesso.");
        } else if (tipo == 2) {
            repoJuridica.excluir(id);
            System.out.println("Pessoa jurídica excluída com sucesso.");
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    // Método para exibir uma pessoa pelo ID
    private static void exibirPorId(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        System.out.print("ID: ");
        int id = scanner.nextInt();

        if (tipo == 1) {
            PessoaFisica pessoa = repoFisica.obter(id);
            if (pessoa != null) {
                pessoa.exibir();
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        } else if (tipo == 2) {
            PessoaJuridica pessoa = repoJuridica.obter(id);
            if (pessoa != null) {
                pessoa.exibir();
            } else {
                System.out.println("Pessoa jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }


    // Método para exibir todas as pessoas
    private static void exibirTodos(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) {
        System.out.println("Escolha o tipo (1 - Física, 2 - Jurídica): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        if (tipo == 1) {
            for (PessoaFisica pessoa : repoFisica.obterTodos()) {
                pessoa.exibir();
                System.out.println("----------------------------");
            }
        } else if (tipo == 2) {
            for (PessoaJuridica pessoa : repoJuridica.obterTodos()) {
                pessoa.exibir();
                System.out.println("----------------------------");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }

    // Método para salvar os dados em arquivos
    private static void salvarDados(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) throws IOException {
        System.out.print("Informe o prefixo dos arquivos: ");
        String prefixo = scanner.nextLine();

        repoFisica.persistir(prefixo + ".fisica.bin");
        repoJuridica.persistir(prefixo + ".juridica.bin");

        System.out.println("Dados salvos com sucesso.");
    }

    // Método para recuperar os dados de arquivos
    private static void recuperarDados(Scanner scanner, PessoaFisicaRepo repoFisica, PessoaJuridicaRepo repoJuridica) throws IOException, ClassNotFoundException {
        System.out.print("Informe o prefixo dos arquivos: ");
        String prefixo = scanner.nextLine();

        repoFisica.recuperar(prefixo + ".fisica.bin");
        repoJuridica.recuperar(prefixo + ".juridica.bin");

        System.out.println("Dados recuperados com sucesso.");
    }
}
```

Resultado da execução:



Análise e Conclusão

1. **Quais as vantagens e desvantagens do uso de herança?**

   Vantagens:

   - **Reutilização de código**: Classes derivadas herdam atributos e comportamentos da superclasse, evitando duplicação.
   - **Organização**: Facilita a estruturação do código em hierarquias lógicas.
   - **Polimorfismo**: Permite tratar objetos de diferentes tipos de forma uniforme.
   - **Extensibilidade**: Novas funcionalidades podem ser adicionadas sem modificar o código existente.

   Desvantagens:

   - **Complexidade**: Hierarquias profundas podem dificultar a manutenção e compreensão do código.
   - **Rigidez**: Alterações na superclasse podem impactar todas as subclasses, aumentando o risco de erros.
   - **Acoplamento**: A forte dependência entre classes pode dificultar a refatoração e a reutilização em novos contextos.

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

   A interface `Serializable` atua como um marcador, indicando que uma classe pode ser convertida em uma sequência de bytes para ser armazenada ou transmitida. Sem essa interface, o Java não sabe como serializar os objetos, resultando em erros durante a persistência.

   Além disso, classes que implementam `Serializable` podem ser manipuladas com `ObjectOutputStream` e `ObjectInputStream`. Em casos que exigem mais controle sobre a serialização, a interface `Externalizable` pode ser utilizada.

3. **Como o paradigma funcional é utilizado pela API Stream no Java?**

A API Stream permite processar coleções de forma declarativa e eficiente, utilizando conceitos da programação funcional, como:

- **Funções lambda**: Para definir operações concisas e reutilizáveis.
- **Operações intermediárias e terminais**: Exemplo: `filter`, `map`, `collect`, permitindo manipular dados de maneira fluida.
- **Imutabilidade**: Os dados originais não são alterados, garantindo maior segurança e previsibilidade no processamento.
- **Paralelismo**: A API permite a execução paralela com `parallelStream()`, otimizando o desempenho em grandes volumes de dados.

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

   O padrão mais comum é o **DAO (Data Access Object)**, que separa a lógica de acesso aos dados da lógica de negócios, facilitando a manutenção e reutilização do código.

   No projeto, esse padrão foi implementado nas classes `PessoaFisicaRepo` e `PessoaJuridicaRepo`. Além disso, frameworks ORM como **Hibernate** são frequentemente utilizados para facilitar a manipulação de bancos de dados.