



Missão Prática | Mundo 3 - Nível 1

Polo Centro - Palhoça – SC

Curso: Desenvolvimento Full Stack

Disciplina: RPG0014 - Iniciando o caminho pelo Java

Turma: 9001

Semestre Letivo: 3

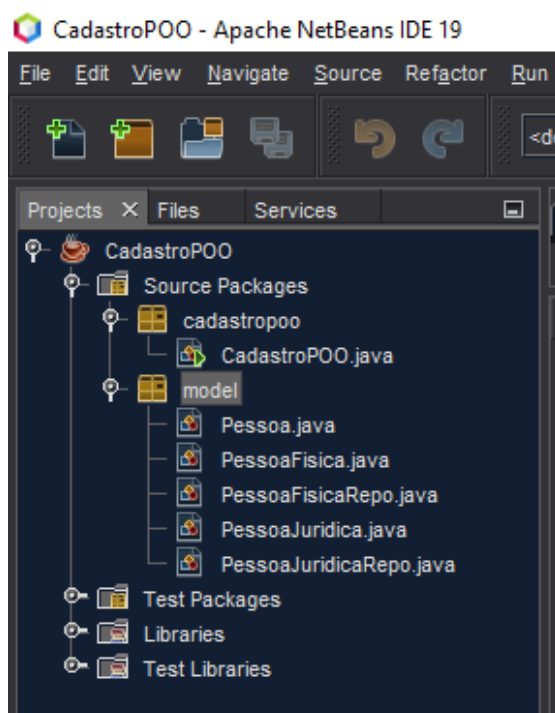
Integrantes da Prática: Andrey Haertel Aires

Repositório GIT

https://github.com/AndreyHaires/MissaoPraticaMundo3_N1

1 - 1º Procedimento | Criação das Entidades e Sistema de Persistência

PASTAS CRIADAS:



2 - 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.

Todos os códigos solicitados neste roteiro de aula

CRIANDO CLASSES:

***Criando a classe pessoa.java

- A classe **Pessoa** é uma representação básica de uma pessoa com atributos como id e nome.
- Implementa a interface **Serializable**, permitindo que objetos dessa classe sejam serializados/desserializados.
- Possui um método **exibir()** que imprime no console as informações da pessoa.

```
package model;

import java.io.Serializable;

// Classe que representa uma Pessoa e implementa a interface Serializable para suportar serialização.

public class Pessoa implements Serializable {

    private int id;    // Identificador único da Pessoa.

    private String nome; // Nome da Pessoa.

    // Construtor que inicializa os atributos id e nome da Pessoa.

    public Pessoa(int id, String nome) {

        this.id = id;

        this.nome = nome;

    }

    // Construtor padrão vazio necessário para serialização.

    public Pessoa() {

    }

    // Método getter para obter o ID da Pessoa.

    public int getId() {

        return id;

    }

    // Método getter para obter o nome da Pessoa.

    public String getNome() {

        return nome;

    }

    // Método setter para definir o ID da Pessoa.

    public void setId(int id) {

        this.id = id;

    }

    // Método setter para definir o nome da Pessoa.
```

```
public void setNome(String nome) {

    this.nome = nome;

}

// Método para exibir as informações da Pessoa no console.

public void exibir() {

    System.out.println("ID: " + id);

    System.out.println("Nome: " + nome);

}

}
```

***Criando a classe PessoaFisica.java

- A classe PessoaFisica estende a classe Pessoa, adicionando atributos específicos como CPF e idade.
- Implementa Serializable.
- Sobrescreve o método exibir() para incluir as informações específicas da pessoa física.

```
package model;

import java.io.Serializable;

// Classe que representa uma Pessoa Física, estendendo a classe Pessoa e implementando Serializable para suportar serialização.

public class PessoaFisica extends Pessoa implements Serializable {

    private String cpf;    // Número do CPF da Pessoa Física.

    private int idade;     // Idade da Pessoa Física.

    // Construtor que inicializa os atributos id, nome, cpf e idade da Pessoa Física.

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

        super(id, nome);

        this.cpf = cpf;

        this.idade = idade;

    }

    // Construtor padrão vazio necessário para serialização.

    public PessoaFisica() {

    }

    // Método getter para obter o CPF da Pessoa Física.

    public String getCPF() {

        return cpf;

    }

    // Método getter para obter a idade da Pessoa Física.

    public int getIdade() {

        return idade;

    }

    // Método setter para definir o CPF da Pessoa Física.

    public void setCPF(String cpf) {

        this.cpf = cpf;

    }

    // Método setter para definir a idade da Pessoa Física.

    public void setIdade(int idade) {
```

```

        this.idade = idade;
    }

    // Método override para exibir as informações da Pessoa Física no console, chamando o método exibir da classe Pessoa.
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da classe Pessoa

        System.out.println("CPF: " + cpf);

        System.out.println("Idade: " + idade);
    }
}

```

***Criando a classe PessoaJuridica.java

- A classe PessoaJuridica estende a classe Pessoa, adicionando o atributo CNPJ.
- Implementa Serializable.
- Sobrescreve o método exibir() para incluir as informações específicas da pessoa jurídica.

```

package model;

import java.io.Serializable;

// Classe que representa uma Pessoa Jurídica, estendendo a classe Pessoa e implementando Serializable para suportar serialização.

public class PessoaJuridica extends Pessoa implements Serializable {

    private String cnpj; // Número do CNPJ da Pessoa Jurídica.

    // Construtor que inicializa os atributos id, nome e cnpj da Pessoa Jurídica.

    public PessoaJuridica(int id, String cnpj, String nome) {

        super(id, nome);

        this.cnpj = cnpj;
    }

    // Construtor padrão vazio necessário para serialização.

    public PessoaJuridica() {

    }

    // Método getter para obter o CNPJ da Pessoa Jurídica.

    public String getCNPJ() {

        return cnpj;
    }

    // Método setter para definir o CNPJ da Pessoa Jurídica.

    public void setCNPJ(String cnpj) {

        this.cnpj = cnpj;
    }

    // Método override para exibir as informações da Pessoa Jurídica no console, chamando o método exibir da classe Pessoa.
    @Override
    public void exibir() {

        super.exibir(); // Chama o método exibir da classe Pessoa

        System.out.println("CNPJ: " + cnpj);
    }
}

```

***Criando a classe PessoaFisicaRepo.java

- A classe **PessoaFisicaRepo** é um repositório para armazenar objetos da classe **PessoaFisica**.
- Oferece métodos para realizar operações como inserção, alteração, exclusão, obtenção e persistência desses objetos.

```
package model;

import java.io.*;

import java.util.ArrayList;

// Classe que representa um repositório de PessoaFisica, gerenciando operações como inserção, alteração, exclusão e persistência em arquivo.

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> listPessoasFisicas = new ArrayList<>();

    // Verifica se uma pessoa com o ID fornecido existe no repositório.

    public boolean pessoaExiste(int id) {

        for (PessoaFisica pessoa : listPessoasFisicas) {

            if (pessoa.getId() == id) {

                return true;

            }

        }

        return false; // Retorna falso se a pessoa não for encontrada

    }

    // Insere uma nova PessoaFisica no repositório.

    public void inserir(PessoaFisica pessoaFisica) {

        listPessoasFisicas.add(pessoaFisica);

    }

    // Altera os dados de uma PessoaFisica com base no ID fornecido.

    public void alterar(int id, String novoNome, String novoCPF, int novaldade) {

        for (PessoaFisica pessoaFisica : listPessoasFisicas) {

            if (pessoaFisica.getId() == id) {

                // Pessoa encontrada, faça a alteração

                pessoaFisica.setNome(novoNome);

                pessoaFisica.setCPF(novoCPF);

                pessoaFisica.setIdade(novaldade);

                return; // Encerra a busca após encontrar a pessoa e realizar a alteração

            }

        }

    }

    // Exclui uma PessoaFisica com base no ID fornecido.

    public void excluir(int id) {

        for (PessoaFisica pessoaFisica : listPessoasFisicas) {

            if (pessoaFisica.getId() == id) {

                listPessoasFisicas.remove(pessoaFisica);

                return; // Encerra a busca após remover a pessoa

            }

        }

    }

}
```

```

    }

    // Obtém uma PessoaFisica com base no ID fornecido.
    public PessoaFisica obter(int id) {

        for (PessoaFisica pessoa : listPessoasFisicas) {

            if (pessoa.getId() == id) {

                return pessoa;

            }

        }

        return null; // Retorna null se a pessoa não for encontrada
    }

    // Obtém todas as PessoaFisica armazenadas no repositório.
    public ArrayList<PessoaFisica> obterTodos() {

        return listPessoasFisicas;

    }

    // Persiste as informações do repositório em um arquivo.
    public void persistir(String nomeArquivoF) throws IOException {

        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(nomeArquivoF))) {

            out.writeObject(listPessoasFisicas);

        }

    }

    // Recupera as informações do repositório a partir de um arquivo.
    public void recuperar(String nomeArquivoF) throws IOException, ClassNotFoundException {

        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivoF))) {

            listPessoasFisicas = (ArrayList<PessoaFisica>) in.readObject();

        }

    }

}

```

***Criando a classe PessoaJuridicaRepo.java

- A classe **PessoaJuridicaRepo** é um repositório para armazenar objetos da classe **PessoaJuridica**.
- Oferece métodos para realizar operações como inserção, alteração, exclusão, obtenção, obtenção de todos os objetos e persistência/deserialização desses objetos.

```

package model;

import java.io.*;

import java.util.ArrayList;

// Classe que representa um repositório de PessoaJuridica, gerenciando operações como inserção, alteração, exclusão e persistência em arquivo.
public class PessoaJuridicaRepo {

    private ArrayList<PessoaJuridica> listPessoasJuridicas = new ArrayList<>();

    // Verifica se uma pessoa com o ID fornecido existe no repositório.
    public boolean pessoaExiste(int id) {

        for (PessoaJuridica pessoa : listPessoasJuridicas) {

            if (pessoa.getId() == id) {

                return true;

            }

        }

    }

```

```
}

return false;

}

// Insere uma nova PessoaJuridica no repositório.

public void inserir(PessoaJuridica pessoaJuridica) {

    listPessoasJuridicas.add(pessoaJuridica);

}

// Altera os dados de uma PessoaJuridica com base no ID fornecido.

public void alterar(int id, String novoCNPJ, String novoNome) {

    for (PessoaJuridica pessoaJuridica : listPessoasJuridicas) {

        if (pessoaJuridica.getId() == id) {

            // Pessoa encontrada, faça a alteração

            pessoaJuridica.setNome(novoNome);

            pessoaJuridica.setCNPJ(novoCNPJ);

            return; // Encerra a busca após encontrar a pessoa e realizar a alteração

        }

    }

}

// Exclui uma PessoaJuridica com base no ID fornecido.

public void excluir(int id) {

    for (PessoaJuridica pessoaJuridica : listPessoasJuridicas) {

        if (pessoaJuridica.getId() == id) {

            listPessoasJuridicas.remove(pessoaJuridica);

            return; // Encerra a busca após remover a pessoa

        }

    }

}

// Obtém uma PessoaJuridica com base no ID fornecido.

public PessoaJuridica obter(int id) {

    for (PessoaJuridica pessoa : listPessoasJuridicas) {

        if (pessoa.getId() == id) {

            return pessoa;

        }

    }

    return null; // Retorna null se a pessoa não for encontrada

}

// Obtém todas as PessoaJuridica armazenadas no repositório.

public ArrayList<PessoaJuridica> obterTodos() {

    return listPessoasJuridicas;

}

// Persiste as informações do repositório em um arquivo.

public void persistir(String nomeArquivo) throws IOException {

    try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(nomeArquivo))) {

        out.writeObject(listPessoasJuridicas);

    }

}
```

```

}

// Recupera as informações do repositório a partir de um arquivo.

public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {

    try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivo))) {

        listPessoasJuridicas = (ArrayList<PessoaJuridica>) in.readObject();

    }

}

}

```

***Importância:

1. **Abstração e Reutilização de Código:** O uso de classes abstratas (como **Pessoa**) permite a modelagem de entidades comuns e a reutilização de código para entidades específicas (como **PessoaFisica** e **PessoaJuridica**).
2. **Polimorfismo:** O polimorfismo é aplicado ao sobrescrever o método **exibir()** nas subclasses, permitindo chamar o mesmo método em objetos de tipos diferentes com comportamentos específicos.
3. **Persistência de Dados:** As classes de repositório (**PessoaFisicaRepo** e **PessoaJuridicaRepo**) facilitam a persistência de objetos em arquivos, permitindo que os dados sejam armazenados e recuperados.
4. **Organização e Manutenção:** O código está organizado de forma modular, facilitando a manutenção e expansão do sistema ao separar responsabilidades específicas em diferentes classes.

Essas classes e repositórios formam uma estrutura que permite o gerenciamento eficiente de informações relacionadas a pessoas físicas e jurídicas em um sistema. O uso de herança, interfaces, e persistência em arquivo contribuem para a flexibilidade e extensibilidade do sistema.

A Classe principal :

```

package cadastrapoo;
import model.PessoaFisica;
import model.PessoaJuridica;
import model.PessoaFisicaRepo;
import model.PessoaJuridicaRepo;
import java.io.IOException;
import java.io.PrintStream;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;
public class CadastroPOO {
    public static void main(String[] args) throws ClassNotFoundException, IOException {
        System.setOut(new PrintStream(System.out, true, "UTF-8"));
        Scanner scanner = new Scanner(System.in);
        PessoaFisicaRepo repoPessoaFisica = new PessoaFisicaRepo();
        PessoaJuridicaRepo repoPessoaJuridica = new PessoaJuridicaRepo();
        int opcao;
        while (true) { //Esse While foi criado para garantir que o programa continue rodando até o usuario acertar a opção
            try {
                System.out.println("=====");
                System.out.println("1 - Incluir Pessoa");
                System.out.println("2 - Alterar Pessoa");
                System.out.println("3 - Excluir Pessoa");
                System.out.println("4 - Procurar pelo ID");
                System.out.println("5 - Exibir Todos");
                System.out.println("6 - Persistir Dados");
                System.out.println("7 - Recuperar Dados");
                System.out.println("0 - Finalizar Programa");
                System.out.println("=====");
                System.out.print("Escolha uma opção: ");
                opcao = scanner.nextInt();
                scanner.nextLine(); //Consome a linha pendente no buffer
                switch (opcao) {
                    case 1:
                        String tipo = ""; // Inicializa 'tipo' com um valor padrão vazio
                        while (!tipo.equals("F") && !tipo.equals("J")) {
                            System.out.println("Escolha o tipo de pessoa:");
                            System.out.printf("F - Pessoa Física | J - Pessoa Jurídica: ");
                            tipo = scanner.nextLine().toUpperCase();
                            if (!tipo.equals("F") && !tipo.equals("J")) {
                                System.out.println("Opção inválida. Por favor, escolha F ou J.");

```



```

    }
}
// Esse if/Else define qual tipo de pessoa o usuario quer manipular.
if (tipo.equals("F")) {
    System.out.print("ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();
    // Verifica se a pessoa com o mesmo ID já existe
    if (repoPessoaFisica.pessoaExiste(id)) {
        System.out.println("Já existe um registro com esse ID. Cadastro não adicionado.");
    } else {
        System.out.print("Nome: ");
        String nome = scanner.nextLine();
        System.out.print("CPF: ");
        String cpf = scanner.nextLine();
        System.out.print("Idade: ");
        int idade = scanner.nextInt();
        scanner.nextLine(); // Consome a nova linha
        PessoaFisica pf = new PessoaFisica(id, nome, cpf, idade);
        repoPessoaFisica.inserir(pf);
        System.out.println("Cadastro adicionado com sucesso.");
    }
}
else if (tipo.equals("J")) {
    System.out.print("ID: ");
    int id = scanner.nextInt();
    scanner.nextLine();
    // Verifica se a pessoa com o mesmo ID já existe
    if (repoPessoaJuridica.pessoaExiste(id)) {
        System.out.println("Já existe um registro com esse ID. Cadastro não adicionado.");
    } else {
        System.out.print("Nome da Empresa: ");
        String nomeEmpresa = scanner.nextLine();
        System.out.print("CNPJ: ");
        String cnpj = scanner.nextLine();
        PessoaJuridica pj = new PessoaJuridica(id, nomeEmpresa, cnpj);
        repoPessoaJuridica.inserir(pj);
    }
}
}
else { // mostra a mensagem caso o usuario não digite a informação esperada.
    System.out.println();
    System.out.println("***** Opção inválida. Por favor, escolha F ou J. *****");
    System.out.println();
}
break;
case 2:
    // Essa parte implementa a opção Alterar Pessoa.

    System.out.print("Digite o ID da pessoa que deseja alterar: ");
    int idParaAlterar = scanner.nextInt();
    scanner.nextLine(); // Consome a linha pendente no buffer

    // Verifica se a pessoa com o ID especificado existe na lista
    if (repoPessoaFisica.pessoaExiste(idParaAlterar)) {
        System.out.print("Digite o novo nome: ");
        String novoNome = scanner.nextLine();
        System.out.print("Digite o novo CPF: ");
        String novoCPF = scanner.nextLine();
        System.out.print("Digite a nova idade: ");
        int novaldade = scanner.nextInt();
        scanner.nextLine(); // Consome a linha pendente no buffer

        // Chama o método para alterar na classe PessoaFisicaRepo
        repoPessoaFisica.alterar(idParaAlterar, novoNome, novoCPF, novaldade);

        System.out.println("Dados da pessoa física atualizados com sucesso.");

        // Verifica se a pessoa com o ID especificado existe na lista
    } else if (repoPessoaJuridica.pessoaExiste(idParaAlterar)) {
        System.out.print("Digite o novo nome da empresa: ");
        String novoNome = scanner.nextLine();
        System.out.print("Digite o novo CNPJ: ");
        String novoCNPJ = scanner.nextLine();

        // Chama o método para alterar na classe PessoaJuridicaRepo
        repoPessoaJuridica.alterar(idParaAlterar, novoCNPJ, novoNome);

        System.out.println("Dados da pessoa jurídica atualizados com sucesso.");

    } else {
        System.out.println("Pessoa com o ID especificado não encontrada.");
    }
}
break;

```

```

case 3:// Essa parte implementa a opção Excluir Pessoa.
System.out.print("Digite o ID da pessoa que deseja excluir: ");
int idParaExcluir = scanner.nextInt();
scanner.nextLine(); //Consome a linha pendente no buffer

if (repoPessoaFisica.pessoaExiste(idParaExcluir)) {
    repoPessoaFisica.excluir(idParaExcluir);
    System.out.println("Pessoa excluída com sucesso.");}
else if (repoPessoaJuridica.pessoaExiste(idParaExcluir)) {
    repoPessoaJuridica.excluir(idParaExcluir);
    System.out.println("Pessoa excluída com sucesso.");}
else {
    System.out.println("Pessoa com o ID especificado não encontrada.");
}
break;

case 4:
// // Essa parte implementa a opção Procurar pelo ID.
System.out.print("Digite o ID da pessoa que deseja procurar: ");
int idParaProcurar = scanner.nextInt();
scanner.nextLine();

// Verifique a existência do ID em ambas as listas
boolean existeEmPessoaFisica = repoPessoaFisica.pessoaExiste(idParaProcurar);
boolean existeEmPessoaJuridica = repoPessoaJuridica.pessoaExiste(idParaProcurar);

if (existeEmPessoaFisica) {
    // A pessoa foi encontrada em Pessoa Física
    PessoaFisica pessoaEncontradaF = repoPessoaFisica.obter(idParaProcurar);
    System.out.println();
    System.out.println("Cadastro encontrado em Pessoa Física:");
    System.out.println("-----");
    pessoaEncontradaF.exibir();
} else if (existeEmPessoaJuridica) {
    // A pessoa foi encontrada em Pessoa Jurídica
    PessoaJuridica pessoaEncontradaJ = repoPessoaJuridica.obter(idParaProcurar);
    System.out.println();
    System.out.println("Cadastro encontrado em Pessoa Jurídica:");
    System.out.println("-----");
    pessoaEncontradaJ.exibir();
} else {
    System.out.println("Pessoa com o ID especificado não encontrada.");
}
break;

case 5: // Essa parte implementa a opção Exibir Todos.
System.out.println("Exibindo todos os cadastros:");
System.out.println();
System.out.println("Pessoas Físicas:");
System.out.println();
List<PessoaFisica> pessoasFisicas = repoPessoaFisica.obterTodos();
for (PessoaFisica pf : pessoasFisicas) {
    pf.exibir();
    System.out.println("-----");
}

System.out.println();
System.out.println("Pessoas Jurídicas:");
System.out.println();
List<PessoaJuridica> pessoasJuridicas = repoPessoaJuridica.obterTodos();
for (PessoaJuridica pj : pessoasJuridicas) {
    pj.exibir();
    System.out.println("-----");
}
break;

case 6: //// Essa parte implementa a opção Persistir Dados.
System.out.print("Digite o nome do arquivo: ");
String nomeArquivo = scanner.next();
String nomeArquivoF = (nomeArquivo + ".fisica.bin");
String nomeArquivoJ = (nomeArquivo + ".juridica.bin");// Nome do arquivo para persistência

try {
    repoPessoaFisica.persistir(nomeArquivoF);
    System.out.println("Dados de pessoas físicas persistidos com sucesso.");
} catch (IOException e) {
    System.out.println("Ocorreu um erro ao persistir os dados de pessoas físicas: " + e.getMessage());
}

try {
    repoPessoaJuridica.persistir(nomeArquivoJ);
    System.out.println("Dados de pessoas jurídicas persistidos com sucesso.");
} catch (IOException e) {

```


6. Funcionalidades Principais:

Inclusão de Pessoa: O usuário pode adicionar uma pessoa física ou jurídica ao cadastro, informando dados como ID, nome, CPF, idade, nome da empresa e CNPJ.

Alteração de Pessoa: Permite a modificação dos dados de uma pessoa existente.

Exclusão de Pessoa: Remove uma pessoa do cadastro com base no ID.

Procura pelo ID: Localiza e exibe os dados de uma pessoa pelo ID.

Exibir Todos: Mostra todos os registros de pessoas físicas e jurídicas no cadastro.

Persistir Dados: Salva os dados do cadastro em arquivos binários.

Recuperar Dados: Restaura dados salvos anteriormente a partir de arquivos binários.

7. Encerramento do Programa:

O programa pode ser finalizado escolhendo a opção "0", que exibe a mensagem "Programa finalizado." e fecha o scanner.

8. Tratamento de Exceções:

O código lida com exceções como entradas inválidas durante a interação com o usuário.

Essencialmente, esse código representa a interação de um usuário com um sistema de cadastro que permite gerenciar informações de pessoas físicas e jurídicas, persistindo e recuperando dados conforme necessário.

RESULTADOS:

Exibindo o Menu:

```
=====
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Procurar pelo ID
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
=====
```

Incluindo uma Pessoa Física:

```
=====
Escolha uma opção: 1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: F
ID: 010101
Nome: ANA MARIA DA SILVA
CPF: 111111111111
Idade: 25
Cadastro adicionado com sucesso.
=====
```

Incluindo uma Pessoa Jurídica:

```
=====
Escolha uma opção: 1
Escolha o tipo de pessoa:
F - Pessoa Física | J - Pessoa Jurídica: J
ID: 090909
Nome da Empresa: MICROSOFT
CNPJ: 99999999999999
=====
```

Exibindo cadastros:

```
=====
Escolha uma opção: 5
Exibindo todos os cadastros:

Pessoas Físicas:

ID: 10101
Nome: ANA MARIA DA SILVA
CPF: 11111111111
Idade: 25
-----

Pessoas Jurídicas:

ID: 90909
Nome: 999999999999999
CNPJ: MICROSOFT
-----
=====
```

Alterando dados de Pessoa Física:

```
=====
Escolha uma opção: 2
Digite o ID da pessoa que deseja alterar: 010101
Digite o novo nome: JOAO DA SILVA
Digite o novo CPF: 2222222222
Digite a nova idade: 45
Dados da pessoa física atualizados com sucesso.
=====
```

Exibir cadastros(alterados)

```
=====
Escolha uma opção: 5
Exibindo todos os cadastros:

Pessoas Físicas:

ID: 10101
Nome: JOAO DA SILVA
CPF: 2222222222
Idade: 45
-----

Pessoas Jurídicas:

ID: 90909
Nome: 999999999999999
CNPJ: MICROSOFT
-----
=====
```

Alterando dados de Pessoa Jurídica:

```
=====
Escolha uma opção: 2
Digite o ID da pessoa que deseja alterar: 090909
Digite o novo nome da empresa: IBM
Digite o novo CNPJ: 444444444444444
Dados da pessoa jurídica atualizados com sucesso.
=====
```

Exibindo dados (alterados)

```
=====
Escolha uma opção: 5
Exibindo todos os cadastros:

Pessoas Físicas:

ID: 10101
Nome: JOAO DA SILVA
CPF: 222222222
Idade: 45
-----

Pessoas Jurídicas:

ID: 90909
Nome: IBM
CNPJ: 444444444444444
-----
=====
```









Tentativa de alterar dados com ID inexistente:

```
=====
Escolha uma opção: 2
Digite o ID da pessoa que deseja alterar: 999
Pessoa com o ID especificado não encontrada.
=====
```

Persistindo dados com nome Exemplo:

```
=====
Escolha uma opção: 6
Digite o nome do arquivo: Exemplo
Dados de pessoas físicas persistidos com sucesso.
Dados de pessoas jurídicas persistidos com sucesso.
=====
```

Verificando arquivos salvos em pasta:

sco Local > DEV > Netbeans > CadastroPOO				
Nome	^	Data de modificação	Tipo	Tamanho
 build		25/10/2023 23:32	Pasta de arquivos	
 nbproject		25/10/2023 23:32	Pasta de arquivos	
 src		25/10/2023 23:32	Pasta de arquivos	
 test		18/10/2023 22:10	Pasta de arquivos	
 build		18/10/2023 22:04	Documento XML	4 KB
 Exemplo.fisica.bin		10/11/2023 14:45	Arquivo BIN	1 KB
 Exemplo.juridica.bin		10/11/2023 14:45	Arquivo BIN	1 KB
 manifest.mf		18/10/2023 22:04	Arquivo MF	1 KB

Finalizando programa:

```
=====
Escolha uma opção: 0
Programa finalizado.
BUILD SUCCESSFUL (total time: 5 minutes 5 seconds)
```

Recuperando dados Pessoa Física:

```
=====
Escolha uma opção: 7
Escolha o tipo de arquivo para recuperar:
F - Pessoa Física | J - Pessoa Jurídica: f
Digite o nome do arquivo: Exemplo.fisica.bin
Dados de pessoas físicas recuperados com sucesso do arquivo 'Exemplo.fisica.bin'.
=====
```

Verificando se dados foram recuperados:

```
=====
Escolha uma opção: 5
Exibindo todos os cadastros:

Pessoas Físicas:

ID: 10101
Nome: JOAO DA SILVA
CPF: 2222222222
Idade: 45
-----

Pessoas Jurídicas:

=====
```

Recuperando dados Pessoa Jurídica:

```
=====
Escolha uma opção: 7
Escolha o tipo de arquivo para recuperar:
F - Pessoa Física | J - Pessoa Jurídica: j
Digite o nome do arquivo: Exemplo.juridica.bin
Dados de pessoas Jurídicas recuperados com sucesso do arquivo 'Exemplo.juridica.bin'.
=====
```

Verificando se dados foram recuperados:

```
=====
Escolha uma opção: 5
Exibindo todos os cadastros:

Pessoas Físicas:

ID: 10101
Nome: JOAO DA SILVA
CPF: 2222222222
Idade: 45
-----

Pessoas Jurídicas:

ID: 90909
Nome: IBM
CNPJ: 4444444444444444
-----
=====
```

1º PROCEDIMENTO - Análise e Conclusão

a) Quais as vantagens e desvantagens do uso de herança?

Vantagens: Reutilização de código, polimorfismo, estrutura hierárquica, manutenção simplificada.

Desvantagens: Acoplamento forte, herança múltipla complexa, hierarquia profunda, violação do princípio de substituição de Liskov.

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

Interface Serializable é necessária para sinalizar a capacidade de serialização de objetos, essencial ao salvar objetos em arquivos binários.

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

Incorpora expressões lambda e operações de alta ordem para operações mais declarativas e flexíveis em coleções.

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

Usamos classes de serialização (`ObjectInputStream` e `ObjectOutputStream`) para converter objetos em bytes, facilitando a gravação e leitura de objetos em arquivos binários.

2º PROCEDIMENTO - Análise e Conclusão

a) O que são elementos estáticos e qual o motivo para o método main adotar esse modificador?

Elementos estáticos são associados à classe. O método `main` é estático para ser chamado sem criar instâncias.

b) Para que serve a classe Scanner?

`Scanner` em Java facilita a entrada de dados do usuário.

c) Como o uso de classes de repositório impactou na organização do código?

Melhorou em relação a organização, centralizando operações específicas e isso facilita a manutenção.