Estácio

Relatório discente de acompanhamento

Universidade	Estácio de Sá
Campus	Campo Grande / Cariacica / ES
Nome do Curso	Desenvolvimento Full Stack
Nome da Disciplina	RPG0014 – Iniciando o Caminho Pelo Java
Turma	9001
Semestre	2024.3
Integrantes	Antonio Vitor Serra dos Santos
Matrícula	202307014834

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

※ 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.
- Persistência em arquivos binários.

Repositório GitHub: http://github.com/t8ninho/Mundo-3-Nivel-1/

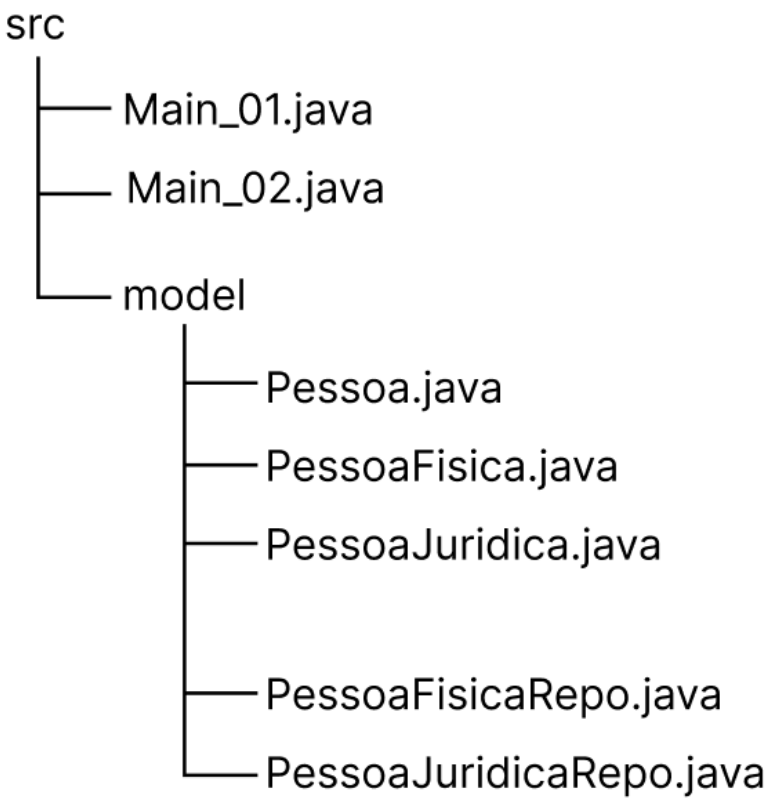
CARIACICA

2024

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

※ Resumo de arquivos criados na execução do trabalho:

Classe Principal: Main_01.java e Main_02.java
Entidades: Pessoa.java, PessoaFisica.java e PessoaJuridica.java.
Gerenciadores: PessoaFisicaRepo.java, PessoaJuridicaRepo.java



※ Todos os códigos solicitados neste roteiro de aula:

✳ Códigos da Entidade / Classe *Pessoa*:

Pessoa.java

```java
package model;

import java.io.Serializable;

/**
 *
 * @author AntonioSantos
 */

public class Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private int id;
    private String nome;

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

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

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

    // getters
    public int getId() {
        return id;
    }

    public String getNome() {
        return nome;
    }

    public void exibir() {
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
    }
}
```

Descrição do Código: A classe Pessoa é uma classe base que implementa Serializable e contém os atributos id e nome. Ela possui métodos getters e setters para esses atributos, além de um método exibir que imprime o id e o nome da pessoa. O construtor inicializa esses valores ao criar um objeto Pessoa.

✳ Códigos da Entidade / Classe *PessoaFisica*:

PessoaFisica.java

```java
package model;

import java.io.Serializable;

/**
 *
 * @author AntonioSantos
 */

public class PessoaFisica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cpf;
    private int idade;

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

    // setters
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }

    // getters
    public String getCpf() {
        return cpf;
    }
    public int getIdade() {
        return idade;
    }

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CPF: " + getCpf());
        System.out.println("Idade: " + getIdade() + "\n");
    }
}
```

Descrição do Código: A classe PessoaFisica estende Pessoa, adicionando os atributos CPF e idade. Ela tem métodos getters e setters para manipular esses atributos e sobrescreve o método exibir para mostrar o CPF e a idade junto com os dados herdados de Pessoa. O construtor inicializa id, nome, CPF e idade.

✳ Códigos da Entidade / Classe *PessoaJuridica*:

Pessoajuridica.java

```java
package model;

import java.io.Serializable;

/**
 *
 * @author AntonioSantos
 */

public class PessoaJuridica extends Pessoa implements Serializable {

    private static final long serialVersionUID = 1L;
    private String cnpj;

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

    // setter
    public void setcnpj(String cnpj) {
        this.cnpj = cnpj;
    }

    // getter
    public String getcnpj() {
        return cnpj;
    }

    @Override
    public void exibir() {
        super.exibir();
        System.out.println("CNPJ: " + getcnpj() + "\n");
    }
}
```

Descrição do Código: A classe PessoaJuridica estende Pessoa, adicionando o atributo CNPJ. Ela inclui métodos getter e setter para manipular o CNPJ e sobrescreve o método exibir para mostrar os dados da pessoa jurídica junto com o CNPJ. O construtor inicializa os atributos id, nome e CNPJ.

✳ Códigos do Gerenciador / Classe *PessoaFisicaRepo*:

PessoaFisicaRepo.java

```java
package model;

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

/**
 *
 * @author AntonioSantos
 */

public class PessoaFisicaRepo implements Serializable {

    private static final long serialVersionUID = 1L;
    private final List<PessoaFisica> pessoasFisicas = new ArrayList<>();

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

    public void alterar(PessoaFisica pessoa) {
        int index = obterIndexPorId(pessoa.getId());
        if (index != -1) {
            pessoasFisicas.set(index, pessoa);
        }
    }

    public void excluir(int id) {
        PessoaFisica pessoa = obter(id);
        if (pessoa != null) {
            pessoasFisicas.remove(pessoa);
        }
    }

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

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

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

    public static PessoaFisicaRepo recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            return (PessoaFisicaRepo) inputStream.readObject();
        }
    }

    private int obterIndexPorId(int id) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == id) {
                return i;
            }
        }
        return -1;
    }
}
```

Descrição do Código: Esse código gerencia um repositório de PessoaFisica, permitindo adicionar, alterar, excluir e recuperar pessoas físicas. Ele salva e recupera os dados em um arquivo binário usando os métodos persistir e recuperar. A lista interna de pessoas é manipulada usando métodos de busca por ID.

✳ Códigos do Gerenciador / Classe *PessoaJuridicaRepo*:

PessoaJuridicaRepo.java

```java
package model;

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

/**
 *
 * @author AntonioSantos
 */

public class PessoaJuridicaRepo implements Serializable {

    private static final long serialVersionUID = 1L;
    private List<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

    public void inserir(PessoaJuridica pessoa) {
        pessoasJuridicas.add(pessoa);
    }

    public void alterar(PessoaJuridica pessoa) {
        int index = obterIndexPorId(pessoa.getId());
        if (index != -1) {
            pessoasJuridicas.set(index, pessoa);
        }
    }

    public void excluir(int id) {
        PessoaJuridica pessoa = obter(id);
        if (pessoa != null) {
            pessoasJuridicas.remove(pessoa);
        }
    }

    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null;
    }

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

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

    public static PessoaJuridicaRepo recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            return (PessoaJuridicaRepo) inputStream.readObject();
        }
    }

    private int obterIndexPorId(int id) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            if (pessoasJuridicas.get(i).getId() == id) {
                return i;
            }
        }
        return -1;
    }
}
```

Descrição do Código: Esse código gerencia uma lista de PessoaJuridica, permitindo adicionar, alterar, excluir e recuperar

registros. Ele salva e recupera os dados de um arquivo binário usando serialização. O repositório é persistido com métodos persistir e recuperar para armazenar e restaurar os dados.

✳ Códigos do Método principal para persistir dados:

Main_01.java

```
package model;

import java.io.IOException;

/**
 *
 * @author AntonioSantos
 */

public class Main_01 {

    public static void main(String[] args) {
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        repo1.inserir(new PessoaFisica(1, "Valdirene Sampaio", "123.456.789-00", 49));
        repo1.inserir(new PessoaFisica(2, "Junior de Caxias", "987.654.321.11", 21));

        try {
            repo1.persistir("toninho.fisica.bin");
            System.out.println("*Dados de Pessoa Física Armazenados.*");
        } catch (IOException e) {
            System.out.println("Erro ao persistir dados de pessoas físicas: " +
e.getMessage());
        }

        PessoaFisicaRepo repo2 = null;
        try {
            repo2 = PessoaFisicaRepo.recuperar("toninho.fisica.bin");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar dados de pessoas físicas: " +
e.getMessage());
        }

        if (repo2 != null) {
            System.out.println("*Dados de pessoa física recuperada.*");
            for (PessoaFisica pessoa : repo2.obterTodos()) {
                pessoa.exibir();
            }
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        repo3.inserir(new PessoaJuridica(3, "Mamae e bebe LTDA", "59.456.277/0001-55"));
        repo3.inserir(new PessoaJuridica(4, "JavaisTarde ME", "22.456.857/0001-66"));

        try {
            repo3.persistir("toninho.juridica.bin");
            System.out.println("*Dados de Pessoa Jurídica Armazenados.*");
        } catch (IOException e) {
            System.out.println("Erro ao persistir dados de pessoas jurídicas: " +
e.getMessage());
        }

        PessoaJuridicaRepo repo4 = null;
        try {
            repo4 = PessoaJuridicaRepo.recuperar("toninho.juridica.bin");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar dados de pessoas jurídicas: " +
e.getMessage());
        }

        if (repo4 != null) {
            System.out.println("*Pessoas Jurídicas Recuperadas.*");
            for (PessoaJuridica pessoa : repo4.obterTodos()) {
                pessoa.exibir();

            }
        }
    }
}
```

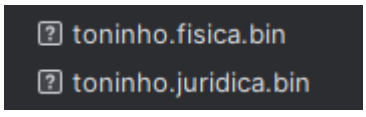
Descrição do código: Este código cria e manipula dois repositórios, um de pessoas físicas e outro de pessoas jurídicas, inserindo

dados em ambos. Ele persiste os dados de cada repositório em arquivos binários e depois os recupera para exibição. Usa tratamento de exceções para garantir que a persistência e recuperação sejam segura, exibindo mensagens de erro caso algo dê errado.

✳ Execução do Método Principal (console):

```
*Dados de Pessoa Física Armazenados.*
*Dados de pessoa física recuperada.*
ID: 1
Nome: Valdirene Sampaio
CPF: 123.456.789-00
Idade: 49

ID: 2
Nome: Junior de Caxias
CPF: 987.654.321.11
Idade: 21

*Dados de Pessoa Jurídica Armazenados.*
*Pessoas Jurídicas Recuperadas.*
ID: 3
Nome: Mamae e bebe LTDA
CNPJ: 59.456.277/0001-55

ID: 4
Nome: JavaisTarde ME
CNPJ: 22.456.857/0001-66


Process finished with exit code 0
```

Arquivos gerados após execução do Método Principal:



✳ Análise e Conclusão:

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

Vantagens:

1. ***Reutilização de Código****: Permite que classes derivadas herdem métodos e atributos de classes base, reduzindo a duplicação de código.*
2. ***Organização****: Facilita a organização do código em uma estrutura hierárquica, tornando-o mais compreensível.*
3. ***Polimorfismo****: Permite que métodos da classe base sejam sobrecarregados ou substituídos, permitindo que diferentes classes se comportem de maneira semelhante.*
4. ***Facilidade de Manutenção****: Alterações em uma classe base podem ser refletidas automaticamente nas classes derivadas, simplificando a manutenção.*

Desvantagens:

1. ***Acoplamento****: A dependência entre classes pode aumentar, dificultando a modificação e a manutenção do sistema.*
2. ***Complexidade****: Estruturas de herança complexas podem tornar o código difícil de entender e seguir, especialmente em hierarquias profundas.*
3. ***Fragilidade****: Mudanças na classe base podem impactar inesperadamente as classes derivadas, levando a bugs.*
4. ***Dificuldade em Testar****: Testar classes que dependem fortemente de herança pode ser complicado, especialmente se houver muitos níveis na hierarquia.*

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

pois permite que objetos sejam convertidos em uma sequência de bytes, facilitando seu armazenamento e recuperação. Sem essa interface, o Java não conseguiria serializar ou deserializar objetos, impossibilitando a gravação e leitura do estado original dos dados. Além disso, ela possibilita o controle de versão dos objetos, garantindo a compatibilidade durante mudanças na estrutura da classe.

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

é utilizado pela API Stream no Java através do uso de funções como objetos de primeira classe, permitindo operações de manipulação de coleções de dados de forma mais declarativa e concisa. A API Stream oferece métodos como map, filter e reduce, que permitem realizar transformações e agregações de dados sem a necessidade de loops explícitos. Isso promove um estilo de programação mais limpo e expressivo, facilitando a leitura e a manutenção do código.

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

No Java, o padrão de desenvolvimento adotado para persistência de dados em arquivos é o DAO (Data Access Object), que separa a lógica de acesso a dados da lógica de negócios. Além disso, utiliza-se a

interface Serializable para facilitar a serialização de objetos em arquivos binários. Essa abordagem promove melhor organização do código e flexibilidade na troca de fontes de dados.

2º Procedimento | Criação do Cadastro em Modo Texto:

Main_02.java

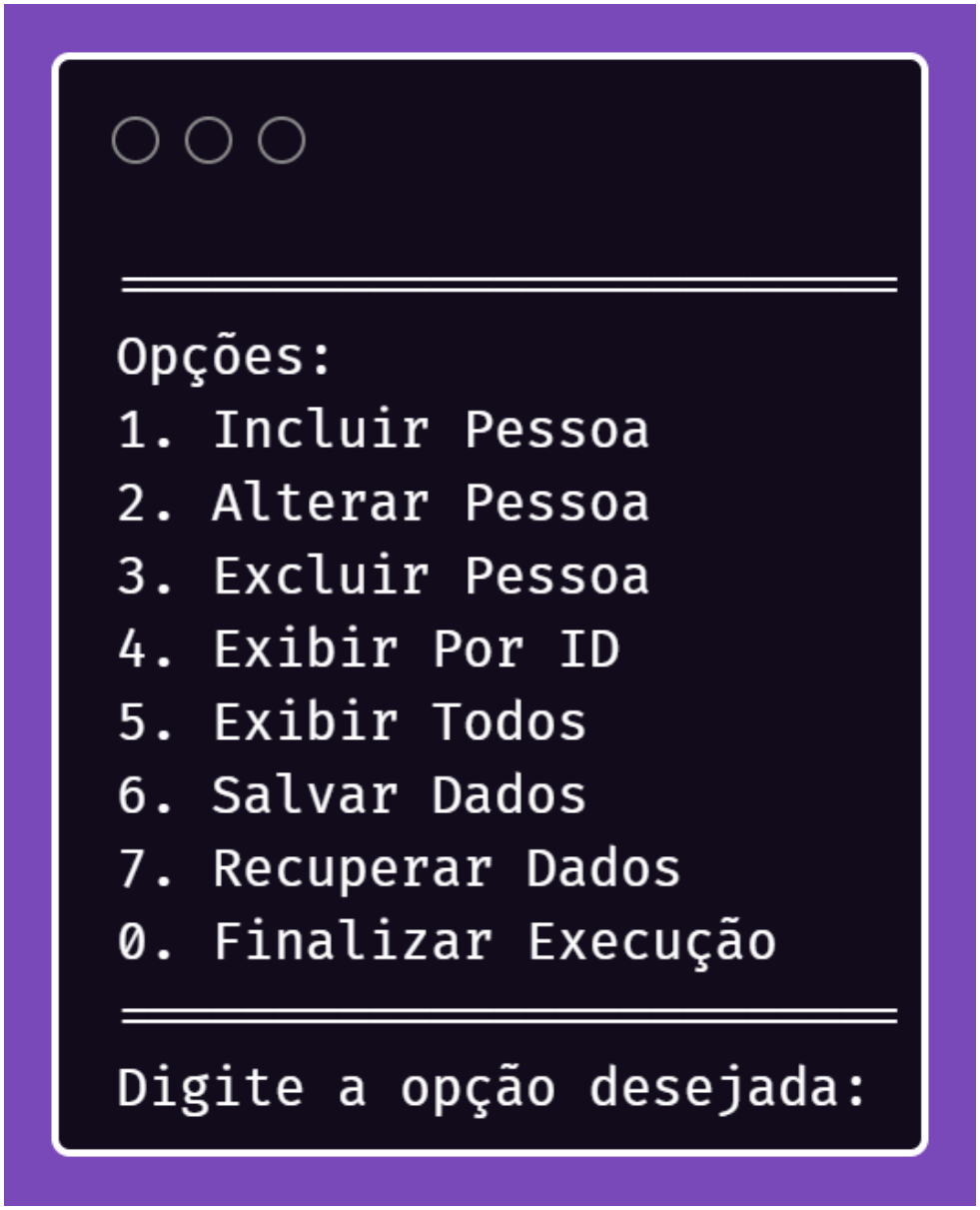
```java
package model;

import java.io.IOException;
import java.util.Scanner;

/**
 *
 * @author AntonioSantos
 */

public class Main_02 {

    private static Scanner scanner = new Scanner(System.in);
    private static PessoaFisicaRepo repoPessoaFisica = new PessoaFisicaRepo();
    private static PessoaJuridicaRepo repoPessoaJuridica = new PessoaJuridicaRepo();

    public static void main(String[] args) {
        int opcao;
        do {
            System.out.println("=========================");
            System.out.println("Opções:");
            System.out.println("1. Incluir Pessoa");
            System.out.println("2. Alterar Pessoa");
            System.out.println("3. Excluir Pessoa");
            System.out.println("4. Exibir Por ID");
            System.out.println("5. Exibir Todos");
            System.out.println("6. Salvar Dados");
            System.out.println("7. Recuperar Dados");
            System.out.println("0. Finalizar Execução");
            System.out.println("=========================");

            opcao = lerInteiro("Digite a opção desejada: ");

            switch (opcao) {
                case 1:
                    incluir();
                    break;
                case 2:
                    alterar();
                    break;
                case 3:
                    excluir();
                    break;
                case 4:
                    exibirPorId();
                    break;
                case 5:
                    exibirTodos();
                    break;
                case 6:
                    salvarDados();
                    break;
                case 7:
                    recuperarDados();
                    break;
                case 0:
                    System.out.println("Finalizando o programa.");
                    break;
                default:
                    System.out.println("Opção inválida. Tente novamente.");
            }
        } while (opcao != 0);
    }

    private static void incluir() {
        String tipo = lerTipoValido(); // Método para ler o tipo (F ou J)
        if (tipo.equalsIgnoreCase("F")) {
            incluirPessoaFisica();
        } else if (tipo.equalsIgnoreCase("J")) {
            incluirPessoaJuridica();
        } else {
            System.out.println("Tipo inválido.");
        }
    }
```

Main_02.java

```java
private static void incluirPessoaFisica() {
        int id = lerInteiro("Digite o ID: ");
        String nome = lerString("Digite o nome: ");
        String cpf = lerString("Digite o CPF: ");
        int idade = lerInteiro("Digite a idade: ");
        repoPessoaFisica.inserir(new PessoaFisica(id, nome, cpf, idade));
    }

    private static void incluirPessoaJuridica() {
        int id = lerInteiro("Digite o ID: ");
        String nome = lerString("Digite o nome: ");
        String cnpj = lerString("Digite o CNPJ: ");
        repoPessoaJuridica.inserir(new PessoaJuridica(id, nome, cnpj));
    }

    private static void alterar() {
        String tipo = lerTipoValido(); // Método para ler o tipo (F ou J)
        int id = lerInteiro("Digite o ID: ");

        if (tipo.equalsIgnoreCase("F")) {
            PessoaFisica pessoa = repoPessoaFisica.obter(id); // Obtendo a pessoa
            if (pessoa != null) {
                System.out.println("Dados atuais da pessoa física:");
                pessoa.exibir();
                System.out.println("Digite os novos dados:");

                // Leitura dos novos dados
                int novoId = lerInteiro("Digite o novo ID: ");
                String novoNome = lerString("Digite o novo nome: ");
                String novoCpf = lerString("Digite o novo CPF: ");
                int novaIdade = lerInteiro("Digite a nova Idade: ");

                // Usando os setters para atualizar os dados
                pessoa.setId(novoId);
                pessoa.setNome(novoNome);
                pessoa.setCpf(novoCpf);
                pessoa.setIdade(novaIdade);

                // Chamando o método alterar
                repoPessoaFisica.alterar(pessoa);

                System.out.println("Dados atualizados com sucesso:");
                pessoa.exibir(); // Exibir os dados atualizados
            } else {
                System.out.println("Pessoa física não encontrada.");
            }
        } else if (tipo.equalsIgnoreCase("J")) {
            PessoaJuridica pessoa = repoPessoaJuridica.obter(id); // Obtendo a empresa
            if (pessoa != null) {
                System.out.println("Dados atuais da pessoa jurídica:");
                pessoa.exibir();
                System.out.println("Digite os novos dados:");

                // Leitura dos novos dados
                int novoId = lerInteiro("Digite o novo ID: ");
                String novoNome = lerString("Digite o novo nome: ");
                String novoCnpj = lerString("Digite o novo CNPJ: ");

                // Usando os setters para atualizar os dados
                pessoa.setId(novoId);
                pessoa.setNome(novoNome);
                pessoa.setcnpj(novoCnpj);

                // Chamando o método alterar
                repoPessoaJuridica.alterar(pessoa);

                System.out.println("Dados atualizados com sucesso:");
                pessoa.exibir(); // Exibir os dados atualizados
            } else {
                System.out.println("Pessoa jurídica não encontrada.");
            }
        } else {
            System.out.println("Tipo inválido.");
        }
    }
```

Descrição do Código: Esse código é um menu interativo que permite gerenciar pessoas físicas e jurídicas em um repositório. Ele oferece opções para incluir, alterar, excluir, exibir, salvar e recuperar dados de pessoas, diferenciando entre "Pessoa Física" e "Pessoa Jurídica". O programa faz uso de classes que representam essas pessoas, e utiliza leitura de dados via Scanner para interações com o usuário, além de persistir os dados em arquivos binários.

✳ Execução do Método Principal (console):



✳ Opção 1: Incluir Pessoa (console):

```
Digite a opção desejada: 1
Escolha o tipo (F para Pessoa Física, J para Pessoa Jurídica): F
Digite o ID: 1
Digite o nome: Antonio Vitor
Digite o CPF: 123.456.789-00
Digite a idade: 24
```

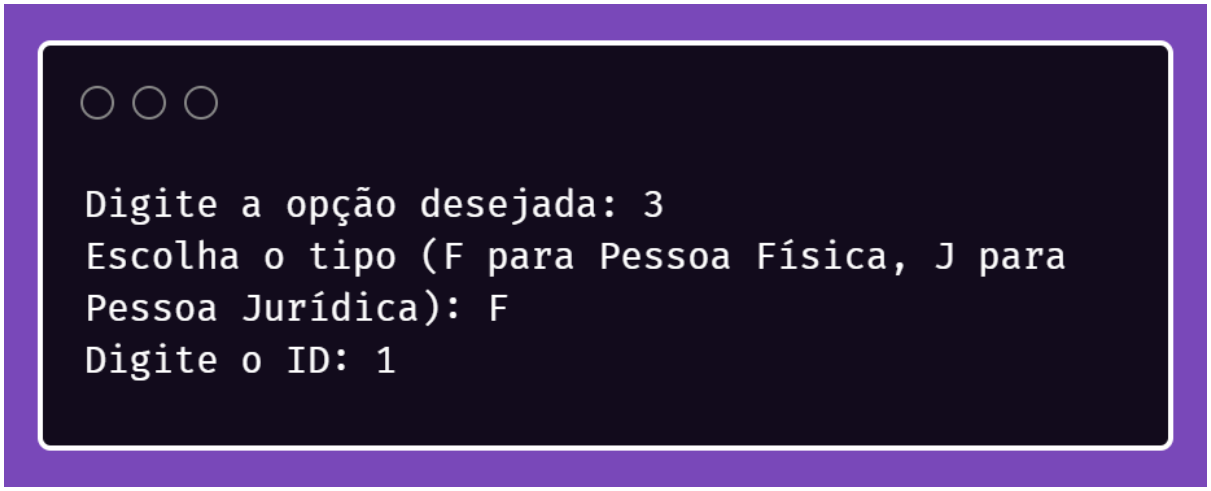
```
Digite a opção desejada: 1
Escolha o tipo (F para Pessoa Física, J para Pessoa Jurídica): J
Digite o ID: 1
Digite o nome: T8ninho Solution LTDA
Digite o CNPJ: 123.456.789/0001-01
```

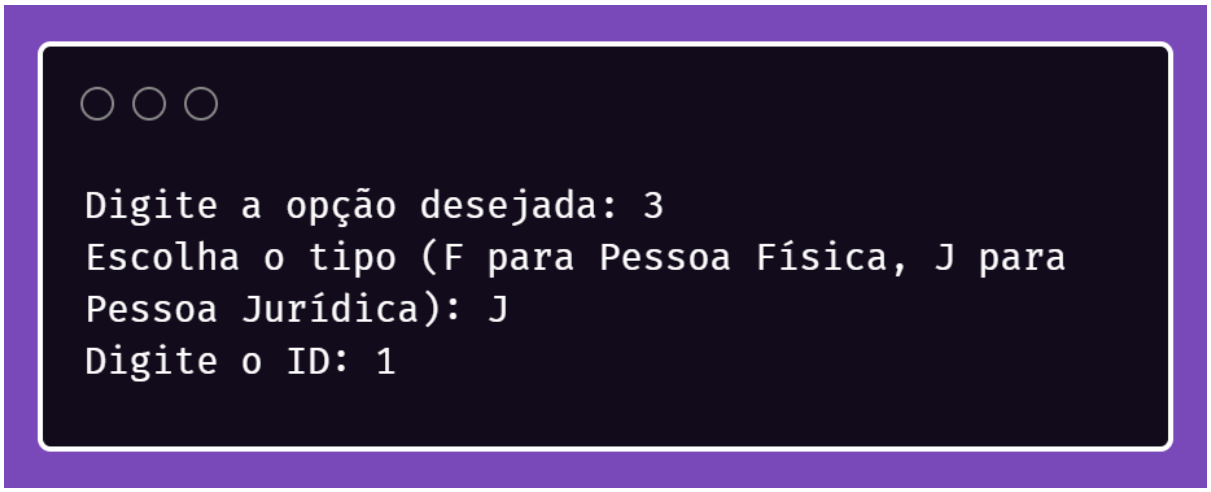
✳ Opção 2: Alterar Pessoa (console):

```
Digite a opção desejada: 2
Escolha o tipo (F para Pessoa Física, J
para Pessoa Jurídica): F
Digite o ID: 1
Dados atuais da pessoa física:
ID: 1
Nome: Antonio Vitor
CPF: 123.456.789-00
Idade: 24

Digite os novos dados:
Digite o novo ID: 2
Digite o novo nome: Antonio Victor
Digite o novo CPF: 222.333.444-55
Digite a nova Idade: 23
Dados atualizados com sucesso:
ID: 2
Nome: Antonio Victor
CPF: 222.333.444-55
Idade: 23
```

```
Digite a opção desejada: 2
Escolha o tipo (F para Pessoa Física, J
para Pessoa Jurídica): J
Digite o ID: 1
Dados atuais da pessoa jurídica:
ID: 1
Nome: T8ninho Solution LTDA
CNPJ: 123.456.789/0001-01

Digite os novos dados:
Digite o novo ID: 3
Digite o novo nome: T8ninho Solucoes LTDA
Digite o novo CNPJ: 111.222.333/0001-33
Dados atualizados com sucesso:
ID: 3
Nome: T8ninho Solucoes LTDA
CNPJ: 111.222.333/0001-33
```

✳ Opção 3: Excluir Pessoa (console):





✳ Opção 4: Exibir Por ID (console):

```
Digite a opção desejada: 4
Escolha o tipo (F para Pessoa Física, J
para Pessoa Jurídica): F
Digite o ID: 2
ID: 2
Nome: Junior de Caxias
CPF: 987.654.321.11
Idade: 21
```

```
Digite a opção desejada: 4
Escolha o tipo (F para Pessoa Física, J para
Pessoa Jurídica): J
Digite o ID: 4
ID: 4
Nome: JavaisTarde ME
CNPJ: 22.456.857/0001-66
```

✳ Opção 5: Exibir Todos (console):

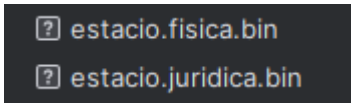
```
Digite a opção desejada: 5
Escolha o tipo (F para Pessoa Física, J
para Pessoa Jurídica): F
ID: 1
Nome: Valdirene Sampaio
CPF: 123.456.789-00
Idade: 49


ID: 2
Nome: Junior de Caxias
CPF: 987.654.321.11
Idade: 21
```

```
Digite a opção desejada: 5
Escolha o tipo (F para Pessoa Física, J
para Pessoa Jurídica): J
ID: 3
Nome: Mamae e bebe LTDA
CNPJ: 59.456.277/0001-55


ID: 4
Nome: JavaisTarde ME
CNPJ: 22.456.857/0001-66
```

* Opção 6: Salvar Dados (console):

```
Digite a opção desejada: 6
Digite o prefixo dos arquivos: estacio
Dados salvos com sucesso.
```

Arquivos gerados pelo comando:



✳ Opção 7: Recuperar Dados (console):

```
Digite a opção desejada: 7
Digite o prefixo dos arquivos: estacio
Dados recuperados com sucesso.
```

✳ Opção 0: Finalizar Execução (console):

```
Digite a opção desejada: 0
Finalizando o programa.
```

✳ Análise e Conclusão:

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

Elementos estáticos em Java são aqueles que pertencem à classe em si, permitindo o acesso sem precisar instanciar objetos. O método main é estático porque é o ponto de partida do programa, e a JVM precisa chamá-lo diretamente antes de criar qualquer instância da classe. Isso garante que a execução do programa comece de forma adequada, independentemente de objetos serem criados.

B) Para que serve a classe Scanner?

É uma ferramenta para entrada de dados em programas Java, sendo eles Inteiros, Strings, Floats, Arquivos.

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

O uso de classes de repositório em um projeto Java melhora a organização do código ao separar claramente a lógica de acesso a dados. Isso facilita a reutilização, a testabilidade e a manutenção, além de oferecer uma melhor abstração da camada de armazenamento.