



UNIVERSIDADE
Estácio de Sá

Universidade	Estácio de Sá
Campus	Polo Barra World / Rio de Janeiro – RJ
Nome do Curso	Desenvolvimento Full Stack
Nome da Disciplina	RPG0014 - Iniciando o Caminho Pelo Java
Turma	9001
Semestre	Primeiro Semestre de 2024
Integrantes do Grupo	Lucas Rodrigues Garcia
Matrícula	202309321807

**VILA VELHA
2024**

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

1- Procedimento | Criação das Entidades e Sistema de Persistência

2- Objetivo 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.

3 - Todos os códigos solicitados neste roteiro de aula:

Códigos da Classe Pessoa

```
package model;

import java.io.Serializable;

/**
 *
 * @author LucasRodrigues
 */
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());
    }
}
```

Códigos da Classe Pessoa Física

```
package model;

import java.io.Serializable;

/**
 *
 * @author LucasRodrigues
 */

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");
    }
}
```

Códigos da Classe Pessoa Jurídica

```
package model;

import java.io.Serializable;

/**
 *
 * @author LucasRodrigues
 */
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 exhibir() {
        super.exibir();
        System.out.println("CNPJ: " + getcnpj() + "\n");
    }
}
```

No pacote model criar os gerenciadores, com as seguintes características: Códigos da Classe PessoaFisicaRepo

```
package model;

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

/**
 *
 * @author LucasRodrigues
 */
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”

Classe PessoaFisicaRepo, contendo um ArrayList de PessoaFisica, nível de acesso privado, e métodos públicos inserir, alterar, excluir, obter e obterTodos, para gerenciamento das entidades contidas no ArrayList.

Códigos da Classe PessoaJuridicaRepo

```
package model;

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

/**
 *
 * @author LucasRodrigues
 */
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”

Classe PessoaJuridicaRepo, com um ArrayList de PessoaJuridica, nível de acesso privado, e métodos públicos inserir, alterar, excluir, obter e obterTodos, para gerenciamento das entidades contidas no ArrayList.

Alterar o método Main da classe principal para testar os repositórios:

- Instanciar um repositório de pessoas físicas (repo1).

- Adicionar duas pessoas físicas, utilizando o construtor completo.
- Invocar o método de persistência em repo1, fornecendo um nome de
 - arquivo fixo, através do código.
- Instanciar outro repositório de pessoas físicas (repo2).
- Invocar o método de recuperação em repo2, fornecendo o mesmo
 - nome de arquivo utilizado anteriormente.
- Exibir os dados de todas as pessoas físicas recuperadas.
- Instanciar um repositório de pessoas jurídicas (repo3).
- Adicionar duas pessoas jurídicas, utilizando o construtor completo.
- Invocar o método de persistência em repo3, fornecendo um nome de
 - arquivo fixo, através do código.
- Instanciar outro repositório de pessoas jurídicas (repo4).
- Invocar o método de recuperação em repo4, fornecendo o mesmo
 - nome de arquivo utilizado anteriormente.
- Exibir os dados de todas as pessoas jurídicas recuperadas.

Códigos da Classe Main com as devidas alterações conforme solicitado acima.


```

package model;

import java.io.IOException;

/**
 *
 * @author LucasRodrigues
 */

public class Main_01 {

    public static void main(String[] args) {
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        repo1.inserir(new PessoaFisica(1, "Fulano Ciclano", "123.456.789-00", 49));
        repo1.inserir(new PessoaFisica(2, "Beltrano da Silva", "987.654.321.11", 21));

        try {
            repo1.persistir("lukinha.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("lukinha.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, "ronaldinho empreendedor", "32.632.644/0001-55"));
        repo3.inserir(new PessoaJuridica(4, "developers", "88.835.732/0001-66"));

        try {
            repo3.persistir("lukinha.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("lukinha.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();
            }
        }
    }
}

```

4 - Os resultados da execução dos códigos também devem ser apresentados:

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.1\lib\idea_rt.jar=58820:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.1\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath C:\Users\Lucas\IdeaProjects\Trabalho-Mundo-3-nivel-1-main\out\production\Trabalho-Mundo-3-nivel-1-main model.Main_01
*Dados de Pessoa Física Armazenados.*
*Dados de pessoa física recuperada.*
ID: 1
Nome: Fulano Ciclano
CPF: 123.456.789-00
Idade: 49

ID: 2
Nome: Beltrano da Silva
CPF: 987.654.321.11
Idade: 21

*Dados de Pessoa Jurídica Armazenados.*
*Pessoas Jurídicas Recuperadas.*
ID: 3
Nome: ronaldinho empreendedor
CNPJ: 32.632.644/0001-55

ID: 4
Nome: developers
CNPJ: 88.835.732/0001-66

Process finished with exit code 0
```

5 – Análise e Conclusão:

† Quais as vantagens e desvantagens do uso de herança?

Vantagens

- **Reutilização de código:** Com a herança, você pode criar uma nova classe que é baseada em uma classe existente. Isso permite a reutilização do código já escrito na classe base.

- **Extensibilidade:** A herança permite estender o comportamento de uma classe existente, adicionando novos métodos e propriedades à subclasse.
- **Polimorfismo:** Classes derivadas podem ser tratadas como a classe base em determinadas situações, o que facilita o polimorfismo e a flexibilidade do código.

Desvantagens

- **Acoplamento forte:** Muita herança pode levar a um acoplamento forte entre classes, o que torna o código mais difícil de entender e manter.
- **Herança múltipla não suportada:** Algumas linguagens, como Java, não suportam a herança múltipla de classes. Isso pode limitar a flexibilidade do design.
- **Complexidade:** Uma hierarquia de classes muito profunda pode levar à complexidade do código e dificultar a compreensão.

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

Ao implementar a interface Serializable, você está indicando que a classe pode ser serializada e desserializada, ou seja, pode ser transformada em bytes e depois recriada a partir desses bytes. Isso é útil em diversas situações, como quando você precisa enviar objetos entre diferentes processos Java ou quando precisa armazenar objetos em um cache distribuído. Segue abaixo alguns exemplos:

- **Persistência de Objetos:** A interface `Serializable` em Java é usada para persistir objetos em um fluxo de bytes. Quando uma classe implementa a interface `Serializable`, os objetos dessa classe podem ser salvos em arquivos binários.
- **Comunicação de Rede:** A serialização é importante quando se trata de enviar objetos pela rede. Uma vez que os objetos são serializados, podem ser transferidos pela rede e recriados em outra máquina.
- **Armazenamento de Objetos:** A serialização permite que os objetos sejam armazenados em arquivos, para que possam ser recuperados posteriormente.

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

A API Stream em Java adota o paradigma funcional para operações de processamento de dados. Utilizando expressões lambda, a API Stream permite operações poderosas e concisas em coleções, contribuindo para um código mais limpo e legível. Expressões Lambda permitem a definição de funções anônimas concisamente. Operações de filtragem e mapeamento, como os métodos `filter` e `map` aplicam operações funcionalmente em elementos da coleção.

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

Ao trabalhar com Java, o padrão de desenvolvimento comumente adotado para persistência de dados em arquivos é o padrão DAO (Data Access Object). O DAO abstrai e encapsula todos os acessos aos dados, permitindo uma separação clara entre a lógica de negócios e as operações de acesso aos dados. Isso facilita a manutenção e a escalabilidade do código.

2 - Procedimento | Criação do Cadastro em Modo Texto

Objetivo 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.**

Os resultados da execução dos códigos também devem ser apresentados:

Resultado do Código Incluir Pessoa

```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
Digite a opção desejada: 1
Escolha o tipo (F para Pessoa Física, J para Pessoa Jurídica): F
Digite o ID: 1
Digite o nome: Lucas
Digite o CPF: 164.139.127-82
Digite a idade: 21
```

Resultado do Código Exibir Por ID

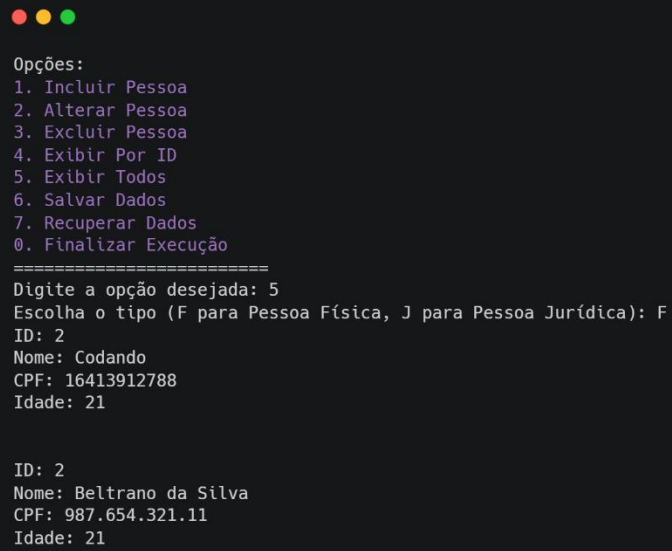
```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
Digite a opção desejada: 4
Escolha o tipo (F para Pessoa Física, J para Pessoa Jurídica): F
Digite o ID: 1
ID: 1
Nome: Fulano Ciclano
CPF: 123.456.789-00
Idade: 49
```

Resultado do Código Alterar Pessoa

```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
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: Fulano Ciclano
CPF: 123.456.789-00
Idade: 49

Digite os novos dados:
Digite o novo ID: 2
Digite o novo nome: Codando
Digite o novo CPF: 16413912788
Digite a nova Idade: 21
Dados atualizados com sucesso:
ID: 2
Nome: Codando
CPF: 16413912788
Idade: 21
```

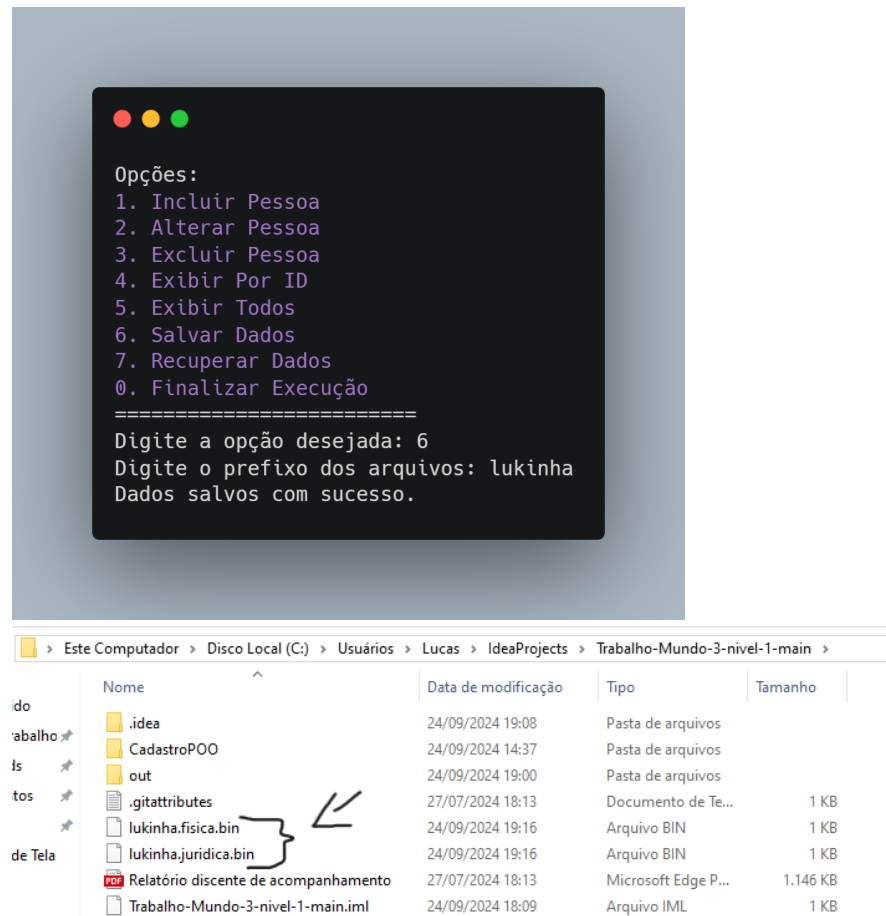
Resultado do Código Exibir Todos



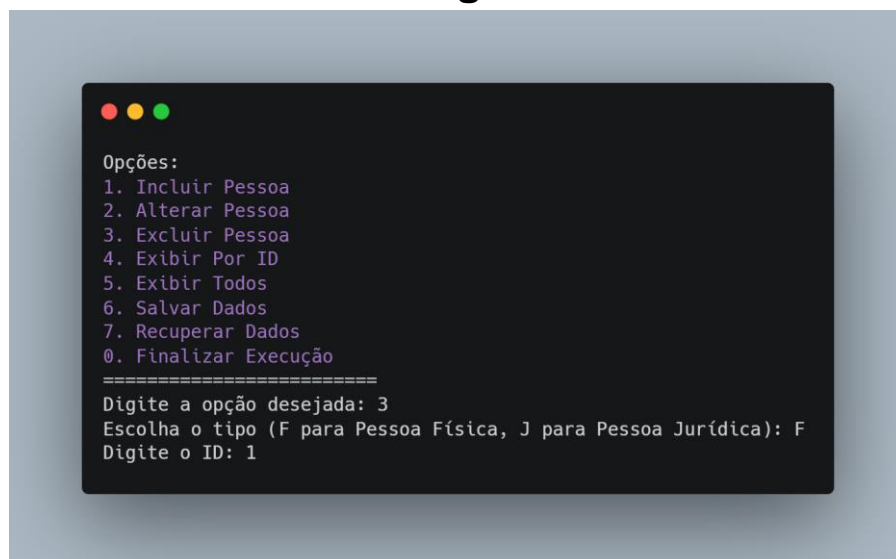
```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
Digite a opção desejada: 5
Escolha o tipo (F para Pessoa Física, J para Pessoa Jurídica): F
ID: 2
Nome: Codando
CPF: 16413912788
Idade: 21

ID: 2
Nome: Beltrano da Silva
CPF: 987.654.321.11
Idade: 21
```

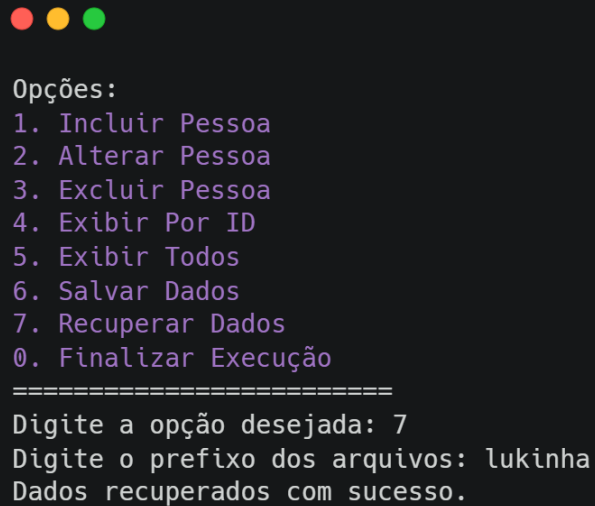

Resultado do Código Salvar Dados



Resultado do Código Excluir Pessoa

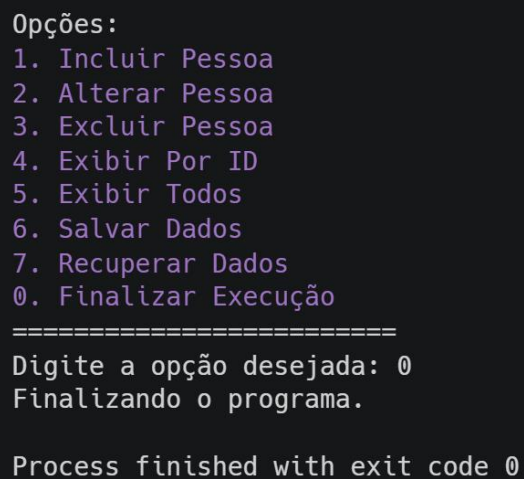


Resultado do Código Recuperar Dados



```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
Digite a opção desejada: 7
Digite o prefixo dos arquivos: lukinha
Dados recuperados com sucesso.
```

Resultado do Código Finalizar Execução



```
Opções:
1. Incluir Pessoa
2. Alterar Pessoa
3. Excluir Pessoa
4. Exibir Por ID
5. Exibir Todos
6. Salvar Dados
7. Recuperar Dados
0. Finalizar Execução
=====
Digite a opção desejada: 0
Finalizando o programa.

Process finished with exit code 0
```

5 - Análise e Conclusão:

‡ 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, em vez de pertencerem a instâncias específicas da classe.

Isso significa que eles podem ser acessados sem a necessidade de criar um objeto da classe. Exemplos de elementos estáticos são métodos e variáveis estáticas.

O método main em Java é declarado como estático para que possa ser invocado sem a necessidade de instanciar a classe. Isso permite que o programa seja executado sem a criação de um objeto da classe principal, facilitando o início da execução do programa.

‡ Para que serve a classe Scanner?

A classe Scanner em Java é usada para ler dados de entrada, como texto digitado pelo usuário no teclado ou informações de arquivos. Ela é frequentemente utilizada para permitir a interação do usuário com programas, facilitando a leitura e análise de dados formatados. Em resumo, o Scanner é uma ferramenta importante para a entrada de dados interativa em programas Java.

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

O uso de classes de repositório em um projeto Java contribui para a organização eficiente do código, promovendo a separação nítida da lógica de acesso a dados. Isso resulta em uma melhor reutilização de código, abstração da camada de armazenamento, maior testabilidade, clareza e manutenção do código, além da capacidade de organizar hierarquicamente as operações de dados para refletir a estrutura de dados do projeto, simplificando o desenvolvimento e a escalabilidade do software.