Curso:
Desenvolvimento Full Stack

Campus:
POLO JARDIM BRASÍLIA - ÁGUAS LINDAS DE GOIÁS - GO

Disciplina:
Iniciando o caminho pelo Java

Turma:
23.2

Aluno:
BRUNO SANTIAGO DE OLIVEIRA

Missão Prática | Nível 1 | Mundo

Objetivo da Prática:

1º Procedimento | Criação das Entidades e Sistema de Persistência
2º Procedimento | Criação do Cadastro em Modo Texto
Bom de modo geral eu juntei os dois códigos do procedimento, e resolvi trazer em apenas 1 código tudo junto. segue o mesmo abaixo.

```
import java.io.*;
import java.util.*;

interface Persistencia {
    void persistir(String nomeArquivo) throws IOException;
    void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException;
}

class Pessoa implements Persistencia, Serializable {
    int id;
    String nome;

    public Pessoa() {
    }

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

    public int getId() {
        return id;
    }

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

```
    public String getNome() {
        return nome;
    }

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

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

    @Override
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            Pessoa pessoa = (Pessoa) in.readObject();
            this.id = pessoa.id;
            this.nome = pessoa.nome;
        }
    }

    @Override
    public String toString() {
        return "ID: " + id + "\nNome: " + nome;
    }
}

class PessoaFisica extends Pessoa {
    private String cpf;
    private int idade;

    public PessoaFisica() {
    }

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

```
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    @Override
    public String toString() {
        return super.toString() + "\nCPF: " + cpf + "\nIdade: " + idade;
    }

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

    @Override
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            PessoaFisica pessoaFisica = (PessoaFisica) in.readObject();
            this.id = pessoaFisica.getId();
            this.nome = pessoaFisica.nome;
            this.cpf = pessoaFisica.cpf;
            this.idade = pessoaFisica.idade;
        }
    }
}

class PessoaJuridica extends Pessoa {
    private String cnpj;
```

```
    public PessoaJuridica() {
    }

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

    public String getCnpj() {
        return cnpj;
    }

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

    @Override
    public String toString() {
        return super.toString() + "\nCNPJ: " + cnpj;
    }

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

    @Override
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            PessoaJuridica pessoaJuridica = (PessoaJuridica) in.readObject();
            this.id = pessoaJuridica.getId();
            this.nome = pessoaJuridica.nome;
            this.cnpj = pessoaJuridica.cnpj;
        }
    }
}

class PessoaFisicaRepo {
    private final ArrayList<PessoaFisica> pessoasFisicas = new ArrayList<>();
    private int nextId = 1;
```

```
    public boolean inserir(PessoaFisica pessoa) {
        if (existeId(pessoa.getId())) {
            System.out.println("ID já existe! Digite um ID diferente.");
            return false;
        }

        pessoasFisicas.add(pessoa);
        nextId = Math.max(nextId, pessoa.getId() + 1);
        return true;
    }

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

    public boolean excluir(int id) {
        return pessoasFisicas.removeIf(pessoa -> pessoa.getId() == id);
    }

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

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

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

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

```
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            ArrayList<PessoaFisica> lista = (ArrayList<PessoaFisica>) in.readObject();
            pessoasFisicas.clear();
            pessoasFisicas.addAll(lista);
            nextId = pessoasFisicas.isEmpty() ? 1 :
pessoasFisicas.get(pessoasFisicas.size() - 1).getId() + 1;
        }
    }

    public boolean existeId(int id) {
        for (PessoaFisica pessoa : pessoasFisicas) {
            if (pessoa.getId() == id) {
                return true;
            }
        }
        return false;
    }
}

class PessoaJuridicaRepo {
    private final ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<>();
    private int nextId = 1;

    public boolean inserir(PessoaJuridica pessoa) {
        if (existeId(pessoa.getId())) {
            System.out.println("ID já existe! Digite um ID diferente.");
            return false;
        }

        pessoasJuridicas.add(pessoa);
        nextId = Math.max(nextId, pessoa.getId() + 1);
        return true;
    }

    public void alterar(PessoaJuridica pessoa) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            if (pessoasJuridicas.get(i).getId() == pessoa.getId()) {
                pessoasJuridicas.set(i, pessoa);
                break;
            }
        }
    }

    public boolean excluir(int id) {
        return pessoasJuridicas.removeIf(pessoa -> pessoa.getId() == id);
```

```
    }

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

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

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

    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            ArrayList<PessoaJuridica> lista = (ArrayList<PessoaJuridica>) in.readObject();
            pessoasJuridicas.clear();
            pessoasJuridicas.addAll(lista);
            nextId = pessoasJuridicas.isEmpty() ? 1 :
pessoasJuridicas.get(pessoasJuridicas.size() - 1).getId() + 1;
        }
    }

    public boolean existeId(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
            if (pessoa.getId() == id) {
                return true;
            }
        }
        return false;
    }
}

public class CadastroPOO {
    public static void main(String[] args) throws UnsupportedEncodingException {
```

```
        System.setOut(new PrintStream(System.out, true, "UTF-8"));
        try (Scanner scanner = new Scanner(System.in).useDelimiter("\n")) {
            PessoaFisicaRepo repoPF = new PessoaFisicaRepo();
            PessoaJuridicaRepo repoPJ = new PessoaJuridicaRepo();

            int opcao;
            do {
                System.out.println("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 a opção: ");

                try {
                    opcao = scanner.nextInt();
                    scanner.nextLine(); // Limpar o buffer

                    switch (opcao) {
                        case 1 -> {
                            System.out.print("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica): ");
                            String tipo = scanner.nextLine().toUpperCase();
                            if (tipo.equals("F")) {
                                PessoaFisica pf = lerDadosPessoaFisica(scanner, repoPF);
                                if (pf != null) {
                                    if (repoPF.inserir(pf)) {
                                        System.out.println("Pessoa Física adicionada com sucesso!");
                                    }
                                }
                            } else if (tipo.equals("J")) {
                                PessoaJuridica pj = lerDadosPessoaJuridica(scanner, repoPJ);
                                if (pj != null) {
                                    if (repoPJ.inserir(pj)) {
                                        System.out.println("Pessoa Jurídica adicionada com
sucesso!");
                                    }
                                }
                            } else {
                                System.out.println("Opção inválida!");
                            }
                        }
```

```
                case 2 -> {
                    System.out.print("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica): ");
                    String tipo = scanner.nextLine().toUpperCase();
                    if (tipo.equals("F")) {
                        PessoaFisica pfAlterar = lerDadosPessoaFisica(scanner, repoPF);
                        if (pfAlterar != null) {
                            System.out.print("Digite o ID da Pessoa Física a ser alterada: ");
                            int idPessoaFisicaAlterar = scanner.nextInt();
                            pfAlterar.setId(idPessoaFisicaAlterar);
                            repoPF.alterar(pfAlterar);
                            System.out.println("Pessoa Física alterada com sucesso!");
                        }
                    } else if (tipo.equals("J")) {
                        PessoaJuridica pjAlterar = lerDadosPessoaJuridica(scanner,
repoPJ);
                        if (pjAlterar != null) {
                            System.out.print("Digite o ID da Pessoa Jurídica a ser alterada:
");
                            int idPessoaJuridicaAlterar = scanner.nextInt();
                            pjAlterar.setId(idPessoaJuridicaAlterar);
                            repoPJ.alterar(pjAlterar);
                            System.out.println("Pessoa Jurídica alterada com sucesso!");
                        }
                    } else {
                        System.out.println("Opção inválida!");
                    }
                }
                case 3 -> {
                    System.out.print("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica): ");
                    String tipo = scanner.nextLine().toUpperCase();
                    if (tipo.equals("F")) {
                        System.out.print("Digite o ID da Pessoa Física a ser excluída: ");
                        int idPessoaFisicaExcluir = scanner.nextInt();
                        if (repoPF.excluir(idPessoaFisicaExcluir)) {
                            System.out.println("Pessoa Física excluída com sucesso!");
                        } else {
                            System.out.println("Pessoa Física não encontrada.");
                        }
                    } else if (tipo.equals("J")) {
                        System.out.print("Digite o ID da Pessoa Jurídica a ser excluída: ");
                        int idPessoaJuridicaExcluir = scanner.nextInt();
                        if (repoPJ.excluir(idPessoaJuridicaExcluir)) {
                            System.out.println("Pessoa Jurídica excluída com sucesso!");
                        } else {
```

```
                        System.out.println("Pessoa Jurídica não encontrada.");
                    }
                } else {
                    System.out.println("Opção inválida!");
                }
            }
            case 4 -> {
                System.out.print("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica): ");
                String tipo = scanner.nextLine().toUpperCase();
                if (tipo.equals("F")) {
                    System.out.print("Digite o ID da Pessoa Física a ser exibida: ");
                    int idPessoaFisicaExibir = scanner.nextInt();
                    PessoaFisica pessoaFisica = repoPF.obter(idPessoaFisicaExibir);
                    if (pessoaFisica != null) {
                        System.out.println(pessoaFisica);
                    } else {
                        System.out.println("Pessoa Física não encontrada.");
                    }
                } else if (tipo.equals("J")) {
                    System.out.print("Digite o ID da Pessoa Jurídica a ser exibida: ");
                    int idPessoaJuridicaExibir = scanner.nextInt();
                    PessoaJuridica pessoaJuridica =
repoPJ.obter(idPessoaJuridicaExibir);
                    if (pessoaJuridica != null) {
                        System.out.println(pessoaJuridica);
                    } else {
                        System.out.println("Pessoa Jurídica não encontrada.");
                    }
                } else {
                    System.out.println("Opção inválida!");
                }
            }
            case 5 -> {
                System.out.print("Escolha o tipo (F - Pessoa Física, J - Pessoa
Jurídica): ");
                String tipo = scanner.nextLine().toUpperCase();
                if (tipo.equals("F")) {
                    listarPessoasFisicas(repoPF);
                } else if (tipo.equals("J")) {
                    listarPessoasJuridicas(repoPJ);
                } else {
                    System.out.println("Opção inválida!");
                }
            }
            case 6 -> {
```

```
                    repoPF.persistir("pessoasFisicas.dat");
                    repoPJ.persistir("pessoasJuridicas.dat");
                    System.out.println("Dados salvos com sucesso!");
                }
                case 7 -> {
                    repoPF.recuperar("pessoasFisicas.dat");
                    repoPJ.recuperar("pessoasJuridicas.dat");
                    System.out.println("Dados recuperados com sucesso!");
                }
                case 0 -> System.out.println("Saindo...");
                default -> System.out.println("Opção inválida!");
            }
        } catch (InputMismatchException e) {
            System.out.println("Opção inválida! Digite um número.");
            scanner.nextLine(); // Limpar o buffer
            opcao = -1; // Para continuar o loop
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao salvar/recuperar dados: " + e.getMessage());
            opcao = -1; // Para continuar o loop
        }
    } while (opcao != 0);
  }
}

  private static PessoaFisica lerDadosPessoaFisica(Scanner scanner,
PessoaFisicaRepo repo) {
    System.out.print("Digite o ID: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer

    if (repo.existeId(id)) {
        System.out.println("ID já existe! Digite um ID diferente.");
        return null;
    }

    System.out.print("Digite o nome: ");
    String nome = scanner.nextLine();

    System.out.print("Digite o CPF: ");
    String cpf = scanner.nextLine();

    System.out.print("Digite a idade: ");
    int idade = scanner.nextInt();
    scanner.nextLine(); // Limpar o buffer

    return new PessoaFisica(id, nome, cpf, idade);
```

```
    }

    private static PessoaJuridica lerDadosPessoaJuridica(Scanner scanner,
PessoaJuridicaRepo repo) {
        System.out.print("Digite o ID: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Limpar o buffer

        if (repo.existeId(id)) {
            System.out.println("ID já existe! Digite um ID diferente.");
            return null;
        }

        System.out.print("Digite o nome: ");
        String nome = scanner.nextLine();

        System.out.print("Digite o CNPJ: ");
        String cnpj = scanner.nextLine();

        return new PessoaJuridica(id, nome, cnpj);
    }

    private static void listarPessoasFisicas(PessoaFisicaRepo repo) {
        ArrayList<PessoaFisica> pessoasFisicas = repo.obterTodos();
        if (pessoasFisicas.isEmpty()) {
            System.out.println("Nenhuma Pessoa Física cadastrada.");
        } else {
            for (PessoaFisica pessoa : pessoasFisicas) {
                System.out.println(pessoa);
                System.out.println();
            }
        }
    }

    private static void listarPessoasJuridicas(PessoaJuridicaRepo repo) {
        ArrayList<PessoaJuridica> pessoasJuridicas = repo.obterTodos();
        if (pessoasJuridicas.isEmpty()) {
            System.out.println("Nenhuma Pessoa Jurídica cadastrada.");
        } else {
            for (PessoaJuridica pessoa : pessoasJuridicas) {
                System.out.println(pessoa);
                System.out.println();
            }
        }
    }
}
```

Análise e Conclusão:

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

Em programação, elementos estáticos geralmente se referem a membros (variáveis ou métodos) de uma classe que pertencem à classe em si, em vez de pertencerem a instâncias individuais dessa classe. O modificador `static` é usado para definir elementos estáticos em muitas linguagens de programação, como Java, C++, C#, etc. O motivo para o método `main` frequentemente ser definido como estático em muitas linguagens tem a ver com a natureza da função `main` em um programa.

Aqui estão algumas características e motivos para o uso de elementos estáticos:

1. **Compartilhamento de dados:** Elementos estáticos são compartilhados por todas as instâncias da classe. Isso significa que eles podem ser usados para armazenar dados ou funcionalidades que são comuns a todas as instâncias da classe, em vez de replicá-los em cada instância.

2. **Acesso direto:** Elementos estáticos podem ser acessados diretamente através do nome da classe, sem a necessidade de criar uma instância da classe. Isso é útil para funções ou variáveis que não dependem do estado de uma instância específica.

3. **Método `main`:** O método `main` é o ponto de entrada para muitos programas em linguagens como Java e C#. Ele é chamado pelo sistema quando o programa é iniciado e, portanto, não faz sentido associá-lo a uma instância específica da classe. Deve ser estático para ser chamado sem a criação de um objeto da classe que o contém.

Por exemplo, em Java, a assinatura típica do método `main` é:

```java
public static void main(String[] args)
```

Aqui, o modificador `static` indica que o método `main` pertence à classe em si, em vez de ser associado a uma instância específica. Isso permite que o sistema chame o método `main` diretamente ao iniciar o programa, sem a necessidade de criar um objeto da classe que contém o método.

Em resumo, os elementos estáticos são usados para dados ou funcionalidades compartilhadas por todas as instâncias de uma classe e o método `main` é declarado

como estático para ser o ponto de entrada do programa, acessível sem criar uma instância da classe principal do programa.

Para que serve a classe Scanner?

A classe `Scanner` é uma classe em Java (e em algumas outras linguagens de programação) que é usada para ler dados de entrada a partir de várias fontes, como o teclado, um arquivo ou uma string. Ela faz parte da biblioteca padrão de Java e é muito útil para interagir com o usuário e processar dados de entrada.

A classe `Scanner` fornece vários métodos para ler diferentes tipos de dados, como inteiros, números de ponto flutuante, strings e muito mais. Alguns dos métodos mais comuns incluem `nextInt()`, `nextDouble()`, `nextLine()`, entre outros. Você pode criar uma instância de `Scanner` associada a uma fonte de entrada específica e, em seguida, usar os métodos apropriados para ler os dados dessa fonte.

Aqui estão alguns exemplos de uso da classe `Scanner` em Java:

1. **Leitura de um número inteiro a partir do teclado:**

```java
import java.util.Scanner;

public class ExemploScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numero = scanner.nextInt();

        System.out.println("Você digitou: " + numero);

        scanner.close(); // Fechar o scanner quando não for mais necessário.
    }
}
```

2. **Leitura de uma linha de texto a partir do teclado:**

```java
import java.util.Scanner;

public class ExemploScanner {
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma frase: ");
        String frase = scanner.nextLine();

        System.out.println("Você digitou: " + frase);

        scanner.close(); // Fechar o scanner quando não for mais necessário.
    }
}
```

3. **Leitura de dados a partir de um arquivo:**

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class LeituraDeArquivo {
    public static void main(String[] args) {
        try {
            File arquivo = new File("arquivo.txt");
            Scanner scanner = new Scanner(arquivo);

            while (scanner.hasNextLine()) {
                String linha = scanner.nextLine();
                System.out.println(linha);
            }

            scanner.close();
        } catch (FileNotFoundException e) {
            System.err.println("Arquivo não encontrado: " + e.getMessage());
        }
    }
}
```

Em resumo, a classe `Scanner` é uma ferramenta poderosa para ler dados de entrada de várias fontes e é frequentemente usada em programas Java para interação com o usuário e processamento de dados.

Como o uso de classes de repositório impactou a organização do código?

O uso de classes de repositório pode ter um impacto significativo na organização do código, especialmente em aplicativos que lidam com acesso a dados complexos. Aqui estão alguns dos principais impactos positivos que as classes de repositório têm na organização do código:

1. **Separação de preocupações (Separation of Concerns):** Uma das principais vantagens das classes de repositório é a separação clara entre a lógica de acesso a dados e a lógica de negócios. Isso significa que a parte do código responsável pelo acesso e manipulação de dados é isolada em classes de repositório, enquanto a lógica de negócios pode ser desenvolvida separadamente. Isso torna o código mais modular e facilita a manutenção e a compreensão, pois cada parte tem uma responsabilidade bem definida.

2. **Testabilidade:** Classes de repositório bem projetadas são mais fáceis de testar, pois permitem a criação de testes unitários mais eficazes. Ao abstrair o acesso a dados em classes de repositório, você pode criar implementações de repositório de teste que não dependem de uma fonte de dados real, tornando os testes mais previsíveis e controláveis.

3. **Reutilização de Código:** As classes de repositório podem ser reutilizadas em várias partes do aplicativo. Isso significa que a lógica de acesso a dados pode ser compartilhada em diferentes partes do aplicativo, evitando a duplicação de código e garantindo consistência na manipulação de dados.

4. **Manutenção Simplificada:** Quando ocorrem mudanças na estrutura do banco de dados ou na fonte de dados, muitas das alterações podem ser isoladas nas classes de repositório. Isso significa que as mudanças na camada de acesso a dados têm menos impacto nas outras partes do aplicativo.

5. **Flexibilidade:** O uso de classes de repositório torna mais fácil trocar a fonte de dados sem afetar a lógica de negócios. Por exemplo, você pode mudar de um banco de dados relacional para um banco de dados NoSQL ou para uma API REST sem alterar a lógica de negócios, desde que as interfaces dos repositórios permaneçam consistentes.

6. **Documentação e Compreensão:** A organização do código em classes de repositório facilita a documentação e a compreensão do sistema. É mais claro entender onde e como os dados são acessados e manipulados quando essa lógica é encapsulada em classes específicas.

Em resumo, o uso de classes de repositório é uma prática de design de software que pode melhorar significativamente a organização do código, tornando-o mais modular, testável, reutilizável e flexível. Isso contribui para um código mais limpo, mais fácil de

manter e menos propenso a erros. É especialmente útil em aplicativos complexos que envolvem acesso a dados, onde a separação de preocupações é essencial.