DGT2811 Programação Back-end Com Java

João Victor de Amorim Martins

Matrícula: 202410064229

Inserir aqui o Campus
Programação Back-end Com Java – 9001 – 2024.5

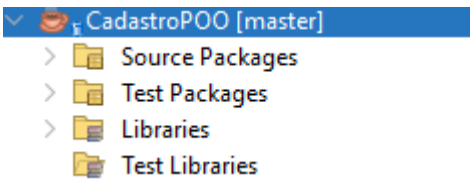
Objetivo da Prática

O objetivo da prática foi implementar um sistema de cadastro em Java, utilizando programação orientada a objetos, com foco no cadastro de pessoas físicas ou jurídicas por meio de um menu em modo de texto gerado diretamente no output do NetBeans. O sistema também permite realizar operações de CRUD (inclusão, alteração, exclusão e consulta em nosso caso) além de salvar e recuperar os dados salvos em arquivos aplicando os conceitos de persistência e tratamento de exceções.

O sistema foi estruturado de uma forma em que foi possível separar as entidades PessoaFisica e PessoaJuridica das classes que gerenciam os dados "PessoaJuridicaRepo" e "PessoaFisicaRepo". Com essa separação a organização do código ficou mais fluida tanto para manutenção quanto para o reaproveitamento das funcionalidades (Heranças) que era um dos focos da prática.

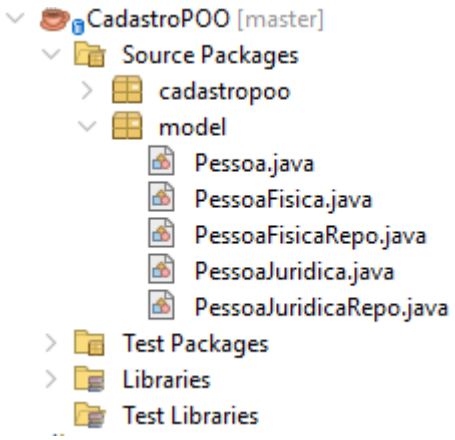
1º Procedimento | Criação das Entidades e Sistema de Persistência

Ao início do procedimento foi solicitado a criação do projeto do tipo Ant.. Java Application no NetBeans, utilizando o nome CadastroPOO para o projeto.



Obs: Em meu caso aparece [master] ao lado pois já foi feito o commit e o push para o github.

Seguindo o desenvolvimento prático foi solicitado a criação do package com nome “model”, para as entidades e gerenciadores: “Pessoa.java”, “PessoaFisica.java”, “PessoaFisicaRepo.java”, “PessoaJuridica.java” e “PessoaJuridicaRepo.java”.



Começando pela superclasse usada como base para definir os atributos e comportamentos mais comuns:

Superclasse “Pessoa.java” localizada no pacote inicial chamado “model”:

```
public class Pessoa implements Serializable {
    protected int id;
    protected String nome;

    public Pessoa(){}
    public Pessoa(int id, String nome){
        this.id = id;
        this.nome = nome;
    }
    public void exibir(){
        System.out.println("ID: " + id);
        System.out.println("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;
    }
}
```

Partindo para a Entidade "PessoaFisica.java" localizada no pacote "model":

```
public class PessoaFisica extends Pessoa implements Serializable {
   private String cpf;
   private int idade;

   public PessoaFisica(){
    super();
   }
   public PessoaFisica(int id, String nome, String cpf, int idade){
       super(id, nome);
       this.cpf = cpf;
       this.idade = idade;
   }
   @Override
   public void exibir(){
       super.exibir();
       System.out.println("CPF: " + cpf);
       System.out.println("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;
    }
}
```

Repositorio "PessoaFisicaRepo.java" responsavel pelo gerenciamento das pessoas físicas:

```
public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoas;

    public PessoaFisicaRepo(){
        pessoas = new ArrayList<>();
    }
    public void inserir(PessoaFisica pessoa){
        pessoas.add(pessoa);
    }
    public void alterar(PessoaFisica pessoa){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == pessoa.getId()){
            pessoas.set(i, pessoa);
            return;
            }
```

```
        }
    }

    public void excluir(int id){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == id){
            pessoas.remove(i);
            return;
            }
        }
    }
    public PessoaFisica obter( int id){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == id){
                return pessoas.get(i);
            }
        }
        return null;
    }
    public ArrayList<PessoaFisica> obterTodos(){
        return pessoas;
    }
    public void persistir(String nomeArquivo) throws Exception{
                            ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo));
        out.writeObject(pessoas);
        out.close();
    }
    public void recuperar(String nomeArquivo) throws Exception{
                                ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo));
        pessoas = (ArrayList<PessoaFisica>) in.readObject();
        in.close();
    }
}
```

Dando sequência aos códigos localizados no nosso pacote inicial “model”, ainda temos a Entidade e o repositório referente às Pessoas Jurídicas…

Sendo a Entidade “PessoaJuridica.java”:

```
public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;

    public PessoaJuridica(){
        super();
    }
    public PessoaJuridica(int id, String nome, String cnpj){
        super(id, nome);
        this.cnpj = cnpj;
    }
    @Override
    public void exibir(){
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }
    public String getCnpj(){
        return cnpj;
    }
    public void setCnpj(String cnpj){
        this.cnpj = cnpj;
    }
}
```

Repositório referente a “PessoaJuridica.java” sendo “PessoaJuridicaRepo.java”:

```
public class PessoaJuridicaRepo {

    private ArrayList<PessoaJuridica> pessoas;

    public PessoaJuridicaRepo(){
        pessoas = new ArrayList<>();
    }
    public void inserir(PessoaJuridica pessoa){
        pessoas.add(pessoa);
    }
    public void alterar(PessoaJuridica pessoa){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == pessoa.getId()){
            pessoas.set(i, pessoa);
            return;
            }
        }
    }

    public void excluir(int id){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == id){
            pessoas.remove(i);
            return;
            }
        }
    }
    public PessoaJuridica obter(int id){
        for(int i = 0; i < pessoas.size(); i++){
            if(pessoas.get(i).getId() == id){
                return pessoas.get(i);
            }
        }
        return null;
```

```
    }
    public ArrayList<PessoaJuridica> obterTodos(){
        return pessoas;
    }
    public void persistir(String nomeArquivo) throws Exception{
                        ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(nomeArquivo));
        out.writeObject(pessoas);
        out.close();
    }
    public void recuperar(String nomeArquivo) throws Exception{
                            ObjectInputStream in = new ObjectInputStream(new
FileInputStream(nomeArquivo));
        pessoas = (ArrayList<PessoaJuridica>) in.readObject();
        in.close();
    }
}
```

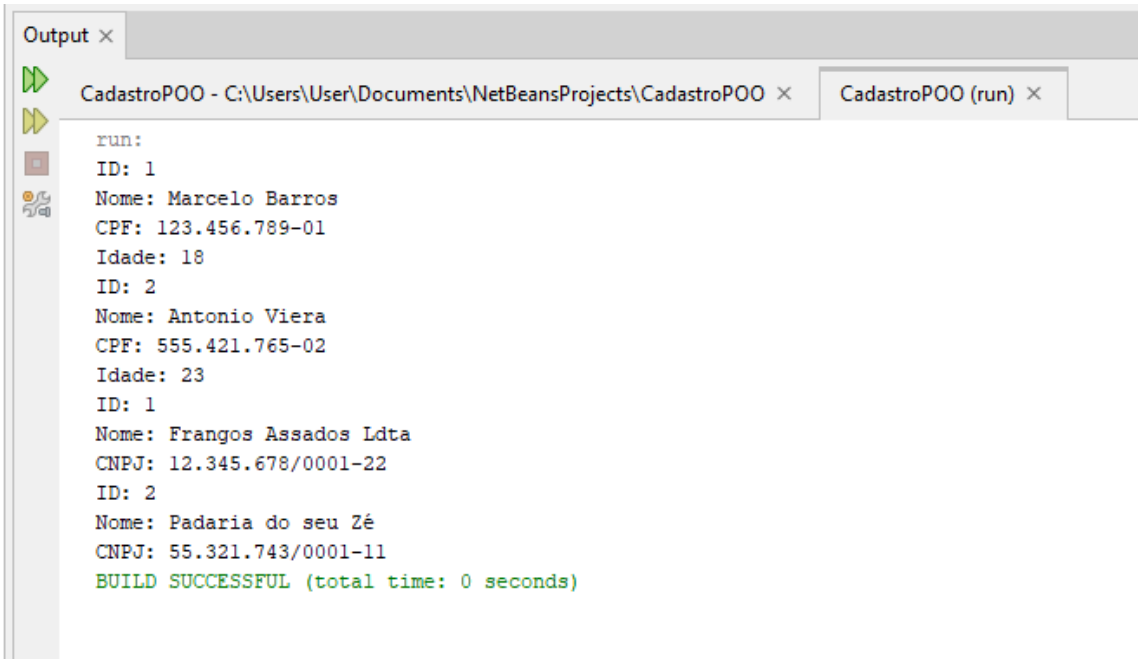
Obs: Ao copiar os códigos alguns espaçamentos foram alterados para que o código ficasse o mais semântico possível e simples de entender. O código original estará disponível no github para as devidas inspeções.

CadastroPOO.java que contem os códigos referente aos procedimentos:

Inicialmente ao criar o projeto Ant.. Java Application, foi gerado o pacote “Cadastropoo” com CadastroPOO.java responsável por armazenar o método main.

```
public static void main(String[] args) throws Exception {

PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Marcelo Barros", "123.456.789-01", 18);
repo1.inserir(pessoaFisica1);

PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Antonio Viera", "555.421.765-02", 23);
repo1.inserir(pessoaFisica2);

repo1.persistir("pessoas_fisicas.bin");

PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
repo2.recuperar("pessoas_fisicas.bin");

ArrayList<PessoaFisica> lista = repo2.obterTodos();
for(int i = 0; i < lista.size(); i++){
   lista.get(i).exibir();
}
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

PessoaJuridica pessoaJuridica1 = new PessoaJuridica(1, "Frangos Assados Ldta", "12.345.678/0001-22");
repo3.inserir(pessoaJuridica1);

PessoaJuridica pessoaJuridica2 = new PessoaJuridica(2,"Padaria do seu Zé", "55.321.743/0001-11");
repo3.inserir(pessoaJuridica2);
repo3.persistir("pessoas_juridicas.bin");
```

```
        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        repo4.recuperar("pessoas_juridicas.bin");

        ArrayList<PessoaJuridica> listaPessoaJuridica = repo4.obterTodos();
        for(int i = 0; i < listaPessoaJuridica.size(); i++){
            listaPessoaJuridica.get(i).exibir();
        }
    }
```



Resultando nesse Output que apresenta uma execução simples, imprimindo somente os objetos estáticos que criamos durante o primeiro procedimento.

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

Começando pelas vantagens do uso de herança está o reaproveitamento de métodos e atributos de uma classe, podemos mencionar a organização lógica em que acaba influenciando a estrutura das classes de forma hierárquica. Podemos considerar o Polimorfismo como uma das principais vantagens da herança em que as subclasses uma acaba não interferindo na outra, podendo ser implementada de forma genérica apenas herdando da superclasse.

Partindo para as desvantagens da herança podemos mencionar que ao utilizar herança acaba gerando uma grande dependência da superclasse, sendo assim qualquer mudança feita na superclasse poderia "quebrar" as subclasses e desencadear um problema complexo se implementado herança em cadeia. Vale mencionar que também existe o risco de criar hierárquicas complexas e difíceis de manter, como na "herança em cadeia", o uso da herança excessiva pode tornar o sistema mais rígido e difícil de futuramente ser refatorado ou até mesmo evoluído se a aplicação começar a crescer.

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

A interface Serializable é necessária porque ela permite que os objetos de uma classe sejam convertidos para um formato que pode ser gravado em arquivo e posteriormente reconstruídos em memória. Para que o java consiga fazer isso de forma segura utilizamos a interface Serializable para que o processo ocorra com segurança ao transformar o objeto em uma sequência de bytes, afinal qualquer "erro", pode gerar arquivos corrompidos ou até mesmo em dados guardados incorretamente.

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

A api Stream utiliza o paradigma funcional no java para permitir que coleções de dados sejam manipuladas por meio de funções em vez de estruturas de repetição tradicionais. Dessa forma, operações como filtragem, mapeamento e até mesmo percorrer dados é possível de forma mais direta sem o uso extensivo de laços "for" tornando o código mais limpo e legível.

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

Quando trabalhamos com java, a persistência de dados em arquivos normalmente segue o padrão DAO (data access object). Esse padrão tem como objetivo de separar a lógica de negócio da lógica de acesso e manipulação dos dados persistidos. Na prática, a responsabilidade de salvar, recuperar, alterar ou

excluir dados em um arquivo fica concentrada em classes específicas, enquanto as entidades representam apenas os dados. Com essa separação de funções o código acaba ficando mais organizado para futuras manutenções ou até mesmo alterações na forma de persistência.

2º Procedimento | Criação do Cadastro em Modo Texto

Código completo referente ao “CadastroPOO.java”:

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

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

                PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Marcelo Barros", "123.456.789-01", 18);
        repo1.inserir(pessoaFisica1);

                PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Antonio Viera", "555.421.765-02", 23);
        repo1.inserir(pessoaFisica2);

        repo1.persistir("pessoas_fisicas.bin");

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        repo2.recuperar("pessoas_fisicas.bin");

        ArrayList<PessoaFisica> lista = repo2.obterTodos();
        for(int i = 0; i < lista.size(); i++){
            lista.get(i).exibir();
        }
```

```
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

 PessoaJuridica pessoaJuridica1 = new PessoaJuridica(1, "Frangos Assados Ldta",
"12.345.678/0001-22");
repo3.inserir(pessoaJuridica1);

   PessoaJuridica pessoaJuridica2 = new PessoaJuridica(2,"Padaria do seu Zé",
"55.321.743/0001-11");
repo3.inserir(pessoaJuridica2);

repo3.persistir("pessoas_juridicas.bin");

PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
repo4.recuperar("pessoas_juridicas.bin");

ArrayList<PessoaJuridica> listaPessoaJuridica = repo4.obterTodos();
for(int i = 0; i < listaPessoaJuridica.size(); i++){
   listaPessoaJuridica.get(i).exibir();
}

Scanner scan = new Scanner(System.in);
int option = -1;

PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();

while(option != 0){

   // menus fixos para guia.
   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 - 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 - Finalizar Programa");
System.out.println("========================");
System.out.print("Escolher uma opção: ");

option = scan.nextInt();




// if das opções do menu acima.
if(option == 1){
   System.out.println("Tipo (F - Fisica / J - Juridica): ");
   String tipo = scan.next().toUpperCase();

   if(tipo.equals("F")){
      System.out.println("Cadastro de pessoa Física selecionado.");

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

      System.out.print("Nome: ");
      scan.nextLine();
      String nome = scan.nextLine();

      System.out.print("CPF: ");
      String cpf = scan.next();

      System.out.print("Idade: ");
      int idade = scan.nextInt();

      PessoaFisica pFisica = new PessoaFisica(id, nome, cpf, idade);
      repoFisica.inserir(pFisica);
```

```
            System.out.println("Pessoa Física cadastrada com sucesso.");

        } else if(tipo.equals("J")){
            System.out.println("Cadastro de pessoa Juridica selecionado.");

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

            System.out.print("Nome: ");
            scan.nextLine();
            String nome = scan.nextLine();

            System.out.print("CNPJ: ");
            String cnpj = scan.next();

            PessoaJuridica pJuridica = new PessoaJuridica(id, nome, cnpj);
            repoJuridica.inserir(pJuridica);

            System.out.println("Pessoa Juridica cadastrada com sucesso.");

        } else {
            System.out.println("Selecione uma opção válida.");
        }

    } else if(option == 2){
        System.out.println("Tipo (F - Fisica / J - Juridica): ");
        String tipo = scan.next().toUpperCase();

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

        if(tipo.equals("F")){
            System.out.println("Alterar pessoa Fisica");
```

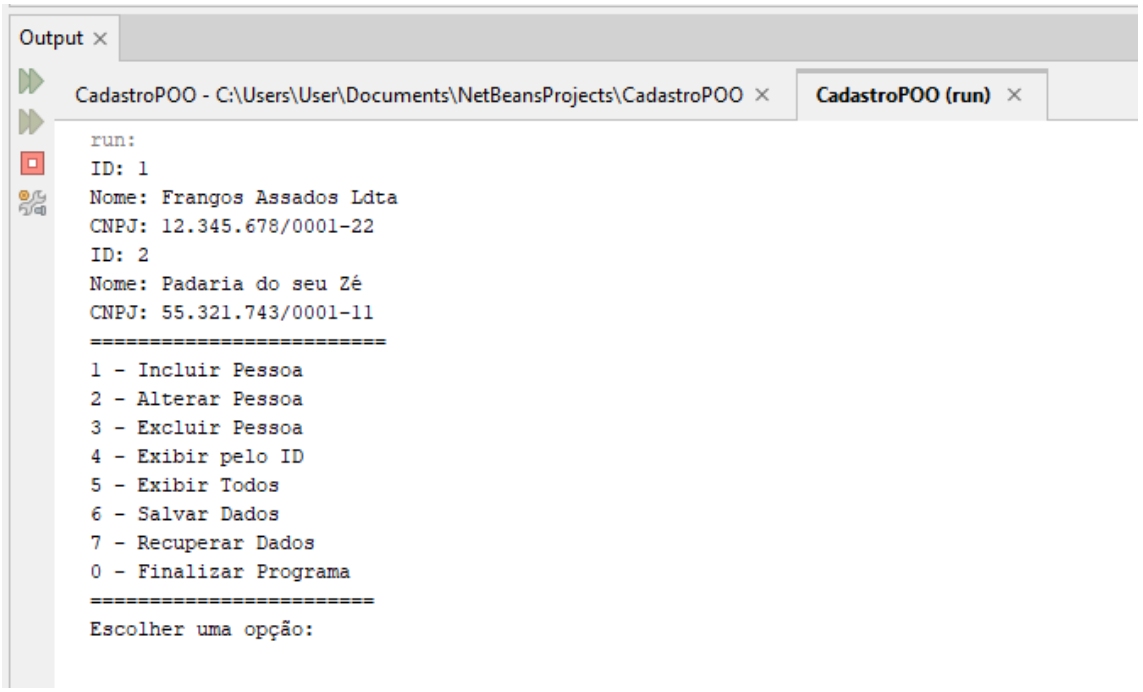
```
        PessoaFisica pFisica = repoFisica.obter(id);
        if(pFisica != null){
            System.out.println("Dados atuais: ");
            pFisica.exibir();

            System.out.println("Informe os novos dados: ");
            System.out.print("Nome: ");
            scan.nextLine();
            String nome = scan.nextLine();

            System.out.print("CPF: ");
            String cpf = scan.next();
            System.out.print("Idade: ");
            int idade = scan.nextInt();

            pFisica.setNome(nome);
            pFisica.setCpf(cpf);
            pFisica.setIdade(idade);

            System.out.println("Pessoa fisica alterada com sucesso.");
        } else{
            System.out.println("Pessoa fisica não encontrada.");
        }
    } else if(tipo.equals("J")){
        System.out.println("Alterar pessoa Juridica.");

        PessoaJuridica pJuridica = repoJuridica.obter(id);
        if(pJuridica != null){
            System.out.println("Dados atuais: ");
            pJuridica.exibir();

            System.out.println("Informe os novos dados: ");

            System.out.print("Nome: ");
            scan.nextLine();
```

```
            String nome = scan.nextLine();

            System.out.print("CNPJ: ");
            String cnpj = scan.next();

            pJuridica.setNome(nome);
            pJuridica.setCnpj(cnpj);

            System.out.println("Pessoa juridica alterada com sucesso.");
        } else{
            System.out.println("Pessoa Juridica não encontrada.");
        }

    } else{
        System.out.println("Selecione uma opção valida.");
    }
} else if (option == 3){
    System.out.println("Tipo (F - Fisica / J - Juridica): ");
    String tipo = scan.next().toUpperCase();

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

    if(tipo.equals("F")){
        repoFisica.excluir(id);
        System.out.println("Pessoa Fisica excluida com sucesso.");
    } else if(tipo.equals("J")){
        repoJuridica.excluir(id);
        System.out.println("Pessoa Juridica excluida com sucesso.");
    } else{
        System.out.println("Tipo invalido.");
    }

} else if(option == 4){
    System.out.println("Tipo (F - Fisica / J - Juridica): ");
```

```
            String tipo = scan.next().toUpperCase();

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

            if(tipo.equals("F")){
                PessoaFisica pFisica = repoFisica.obter(id);
                if(pFisica != null){
                    pFisica.exibir();
                } else{
                    System.out.println("Pessoa Fisica não encontrada.");
                }
            } else if(tipo.equals("J")){
                PessoaJuridica pJuridica = repoJuridica.obter(id);
                if(pJuridica != null){
                    pJuridica.exibir();
                } else{
                    System.out.println("Pessoa Juridica não encontrada.");
                }
            } else{
                System.out.println("Tipo invalido.");
            }

        } else if(option == 5){
            System.out.println("Tipo (F Física / J - Juridica): ");
            String tipo = scan.next().toUpperCase();

            if(tipo.equals("F")){
                ArrayList<PessoaFisica> listaFisica = repoFisica.obterTodos();
                if(listaFisica.isEmpty()){
                    System.out.println("Nenhuma pessoa fisica cadastrada.");
                } else{
                    for(int i = 0; i < listaFisica.size(); i++){
                        listaFisica.get(i).exibir();
                    }
```

```
            }
        } else if(tipo.equals("J")){
            ArrayList<PessoaJuridica> listaJuridica = repoJuridica.obterTodos();
            if(listaJuridica.isEmpty()){
                System.out.println("Nenhuma pessoa Juridica cadastrada.");
            } else{
                for(int i = 0; i < listaJuridica.size(); i++){
                    listaJuridica.get(i).exibir();
                }
            }
        } else{
            System.out.println("Tipo invalido.");
        }

    } else if(option == 6){
        System.out.print("Informe o prefixo dos arquivos para salvar: ");
        String prefixo = scan.next();

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

            System.out.println("Dados salvos com sucesso.");
        } catch(Exception e){
            System.out.println("Erro ao salvar os dados.");
        }
    } else if(option == 7){
        System.out.print("Informe o prefixo dos arquivos para recuperar: ");
        scan.nextLine();
        String prefixo = scan.nextLine();

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

```
            System.out.println("Dados recuperados com sucesso.");
            } catch(Exception e){
                System.out.println("Erro ao recuperar.");
            }
        } else if(option == 0){
            System.out.println("Programa Finalizado.");
        } else {
            System.out.println("Opção invalida.");
        }
    }
    scan.close();
}
```



O Output mostrado acima foi o resultado de todas as operações presentes no segundo procedimento, apenas com ocultação dos objetos referentes ao primeiro procedimento, entretanto o código completo e original está disponível no github para verificação.

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

Elementos estáticos são atributos ou métodos que pertencem a classe e não a um objeto específico, isso significa que eles podem ser acessados sem que seja necessário criar uma instância da classe. No java, um elemento estático existe quando a classe estiver carregada na memória, independente da criação de objetos.

No sistema em que desenvolvemos durante a prática, o método main usa o modificador static (public static void main) porque ele precisa ser executado como ponto inicial da aplicação, antes mesmo da criação de qualquer objeto. Como não temos instâncias da classe CadastroPOO criadas previamente, o método main precisa ser estático para que o programa possa ser iniciado diretamente pela classe.

b. Para que serve a classe Scanner?

A classe Scanner serve para ler dados de entrada permitindo que o programa receba informações digitadas pelo usuário ou até mesmo receber informações de arquivos. No contexto da prática em que fizemos, a classe Scanner é utilizada para capturar os dados informados pelo usuário no console, como as opções de (Jurídica ou Física), id, nome, CPF, CNPJ e outras informações necessárias para realizar as operações de CRUD (cadastro, alteração, exclusão, consulta). Dessa forma o Scanner possibilita a interação entre usuário e o sistema por meio do modo de texto.

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

O uso de classes de repositório impactou de forma positiva a organização do código ao separar claramente as responsabilidades do sistema. As entidades “PessoaFisica.java” e “PessoaJuridica.java” ficaram responsaveis apenas por representar os dados e seus comportamentos basicos, enquanto as classes de repositorio “PessoaFisicaRepo.java” e “PessoaJuridicaRepo.java” passaram a concentrar toda a lógica de gerenciamento desses dados.

Com essa separação o código ficou mais legível, organizado e fácil de manter, pois as operações (inserir, alterar, excluir, buscar, persistir) ficaram centralizadas em um único local. Além disso, a abordagem que utilizamos facilitou a realização das funcionalidades e reduziu a repetição de código no main.

Conclusão

A missão prática foi fundamental para entender os conceitos de programação orientada a objetos aplicados em java, pois permitiu pôr em prática e consolidar a parte de situações reais no desenvolvimento. Durante a implementação de um sistema de cadastro completo, foi possível compreender melhor a importância da organização do código, da separação de responsabilidades entre classes e do uso correto de entidades e repositórios.

Durante o desenvolvimento, um ponto que exigiu maior atenção e pesquisa, foi o uso da classe Scanner, especialmente no controle da leitura de dados e na chamada dos métodos como next(), nextInt() e nextLine(). Inicialmente a falta de compreensão sobre a permanência do caractere de quebra de linha no buffer de entrada causou comportamentos inesperados como campos não preenchidos corretamente. Com a prática foi possível entender a necessidade de realizar leituras adicionais para “limpar” o buffer e garantir que os dados digitados pelo usuário fossem capturados corretamente, o que reforçou a importância de compreender não apenas a sintaxe, mas também o funcionamento interno da leitura de dados em Java.

Outras dificuldades se destacaram como o controle do fluxo do programa por meio de menus em modo texto, o tratamento de entradas do usuário com a classe Scanner e a necessidade de atenção ao uso correto de tipos de dados. Além disso, a implementação da persistência em arquivos binários mostrou a importância da interface Serializable e do tratamento de exceções para garantir a integridade dos dados. Outro ponto importante foi perceber como a utilização de repositórios tornou o código mais limpo, simples e de fácil manutenção. Essa estrutura ajudou a reduzir a complexidade do método main, mesmo ele concentrando o controle geral da aplicação.

Por fim, a prática contribuiu significativamente para o entendimento de como um sistema simples pode ser estruturado de forma mais profissional, respeitando conceitos como herança, polimorfismo, encapsulamento e persistência de dados.