Trabalho Prático | PROGRAMACAO BACK-END COM JAVA

Érica Abrantes de Oliveira Lima Ignatios 202405045475

Campus Conselheiro Lafaiete - MG

***DGT2821 Programação Back*-end com Java – 4 Semestre**

Objetivo da Prática

O objetivo desta prática foi implementar um sistema de cadastro de clientes em Java utilizando o paradigma de Orientação a Objetos. O projeto focou na utilização de herança e polimorfismo para definição das entidades (Pessoa, Pessoa Física e Pessoa Jurídica) e na implementação da persistência de dados em arquivos binários (serialização), permitindo salvar e recuperar as informações do disco. Além disso, buscou-se exercitar o tratamento de exceções e a manipulação de fluxos de entrada e saída na linguagem Java

1º Procedimento | Criação das Entidades e Sistema de Persistência

1º Procedimento | Criação das Entidades e Sistema de Persistência

Abaixo estão os códigos fonte desenvolvidos para a criação do modelo de dados e dos repositórios responsáveis pela persistência, lembrando que os códigos se encontram no repositório do github no link : https://github.com/ericaabrantes/cadastroPOO-Java

1. Classe Pessoa.java (Superclasse)

Pessoa.java

Source History

```java
package model;

import java.io.Serializable;

public class Pessoa implements Serializable {
    private int id;
    private 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; }
}
```

2. Classe PessoaFisica.java

PessoaFisica.java

Source History

```java
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    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;
    }

    @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; }
}
```

3. Classe PessoaJuridica.java

PessoaJuridica.java

Source History

```java
package model;

import java.io.Serializable;

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

    public PessoaJuridica() {
    }

    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; }
}
```

4. Classe PessoaFisicaRepo.java (Repositório)

PessoaFisicaRepo.java

Source History

```java
package model;
import java.io.*;
import java.util.ArrayList;

public class PessoaFisicaRepo {
    private ArrayList<PessoaFisica> lista = new ArrayList<>();

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

    public void alterar(PessoaFisica pessoaNova) {
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i).getId() == pessoaNova.getId()) {
                lista.set(i, pessoaNova);
                return;
            }
        }
    }

    public void excluir(int id) {
        lista.removeIf(p -> p.getId() == id);
    }

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

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

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

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            lista = (ArrayList<PessoaFisica>) input.readObject();
        }
    }
}
```

5. Classe PessoaJuridicaRepo.java (Repositório)

PessoaJuridicaRepo.java

Source History

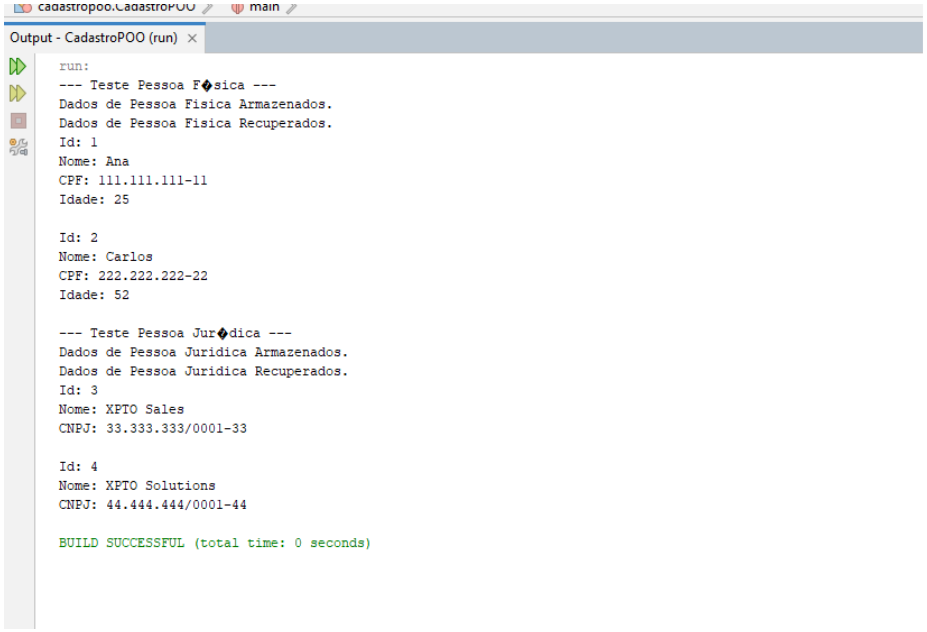
```java
package model;

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

public class PessoaJuridicaRepo {
    private ArrayList<PessoaJuridica> lista = new ArrayList<>();

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

    public void alterar(PessoaJuridica pessoaNova) {
        for (int i = 0; i < lista.size(); i++) {
            if (lista.get(i).getId() == pessoaNova.getId()) {
                lista.set(i, pessoaNova);
                return;
            }
        }
    }

    public void excluir(int id) {
        lista.removeIf(p -> p.getId() == id);
    }

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

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

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

    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            lista = (ArrayList<PessoaJuridica>) input.readObject();
        }
    }
}
```

6. Classe CadastroPOO.java

CadastroPOO.java

```java
package cadastropoo;
import model.*;
import java.io.IOException;

public class CadastroPOO {

    public static void main(String[] args) {
        try {
            System.out.println("--- Teste Pessoa Física ---");
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
            repo1.inserir(new PessoaFisica(1, "Ana", "111.111.111-11", 25));
            repo1.inserir(new PessoaFisica(2, "Carlos", "222.222.222-22", 52));

            repo1.persistir("pessoas-fisicas.bin");
            System.out.println("Dados de Pessoa Fisica Armazenados.");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoas-fisicas.bin");
            System.out.println("Dados de Pessoa Fisica Recuperados.");

            for (PessoaFisica p : repo2.obterTodos()) {
                p.exibir();
                System.out.println();
            }

            System.out.println("--- Teste Pessoa Jurídica ---");

            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
            repo3.inserir(new PessoaJuridica(3, "XPTO Sales", "33.333.333/0001-33"));
            repo3.inserir(new PessoaJuridica(4, "XPTO Solutions", "44.444.444/0001-44"));


            repo3.persistir("pessoas-juridicas.bin");
            System.out.println("Dados de Pessoa Juridica Armazenados.");


            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar("pessoas-juridicas.bin");
            System.out.println("Dados de Pessoa Juridica Recuperados.");


                for (PessoaJuridica p : repo4.obterTodos()) {
                    p.exibir();
                    System.out.println();
                }

            } catch (IOException | ClassNotFoundException e) {
                System.out.println("Erro: " + e.getMessage());
                e.printStackTrace();
            }
    }
}
```

Output da Missao 1



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

A principal vantagem é o reaproveitamento de código, permitindo que atributos e métodos comuns (como id e nome na classe Pessoa) sejam definidos uma única vez e herdados pelas subclasses (PessoaFisica, PessoaJuridica). Isso também facilita o uso do polimorfismo. A desvantagem é o acoplamento forte criado entre as classes; alterações na superclasse afetam todas as subclasses, o que pode tornar a manutenção complexa e propagar erros se não for bem planejada.

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

Ela atua como uma interface de marcação para a Java Virtual Machine (JVM). Ela informa que a classe está autorizada a ter seu estado (objetos) convertido em uma sequência de bytes. Sem implementar essa interface, o uso das classes de fluxo de objetos (como ObjectOutputStream) lançaria uma exceção do tipo NotSerializableException ao tentar gravar os dados

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

Ele é utilizado através de Expressões Lambda e da API de Streams para manipular coleções de forma declarativa e concisa. No projeto, utilizamos esse conceito no método excluir dos repositórios: lista.removeIf(p -> p.getId() == id);. Nesse caso, passa-se uma função (predicado) como argumento para o método removeIf, evitando a necessidade de escrever laços de repetição explícitos e focando no "o que" deve ser feito, e não no "como".

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

Neste projeto, adotou-se o padrão DAO (Data Access Object) ou Repository Pattern. Criou-se classes específicas (PessoaFisicaRepo e PessoaJuridicaRepo) cuja única responsabilidade é gerenciar o acesso aos dados (CRUD) e a persistência (salvar/recuperar). Isso promove uma separação clara entre a lógica de persistência e o restante da aplicação, facilitando a organização e a manutenção do código.

2º Procedimento | Criação do Cadastro em Modo Texto

CadastroPOO.java [-/M]

Source | History

```
1  package cadastropoo;
2
3  import model.*;
4  import java.io.IOException;
5  import java.util.Scanner;
6
7  public class CadastroPOO {
8
9      public static void main(String[] args) {
10         PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();
11         PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();
12         Scanner scanner = new Scanner(System.in);
13
14         int opcao = -1;
15
16         while (opcao != 0) {
17             System.out.println("\n================================");
18             System.out.println("1 - Incluir Pessoa");
19             System.out.println("2 - Alterar Pessoa");
20             System.out.println("3 - Excluir Pessoa");
21             System.out.println("4 - Buscar pelo Id");
22             System.out.println("5 - Exibir Todos");
23             System.out.println("6 - Persistir Dados");
24             System.out.println("7 - Recuperar Dados");
25             System.out.println("0 - Finalizar Programa");
26             System.out.println("================================");
27             System.out.print("Opção: ");
28
29             opcao = scanner.nextInt();
30
30             try {
31                 switch (opcao) {
33                     case 1: // INCLUIR
34                         System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
35                         String tipo1 = scanner.next().toUpperCase();
36                         System.out.print("Digite o id da pessoa: ");
37                         int id1 = scanner.nextInt();
38                         System.out.println("Insira os dados...");
39                         System.out.print("Nome: ");
40                         String nome1 = scanner.next();
41
42                         if (tipo1.equals("F")) {
43                             System.out.print("CPF: ");
44                             String cpf = scanner.next();
45                             System.out.print("Idade: ");
46                             int idade = scanner.nextInt();
47                             repoFisica.inserir(new PessoaFisica(id1, nome1, cpf, idade));
48                         } else if (tipo1.equals("J")) {
49                             System.out.print("CNPJ: ");
50                             String cnpj = scanner.next();
51                             repoJuridica.inserir(new PessoaJuridica(id1, nome1, cnpj));
52                         }
53                         break;
54
```

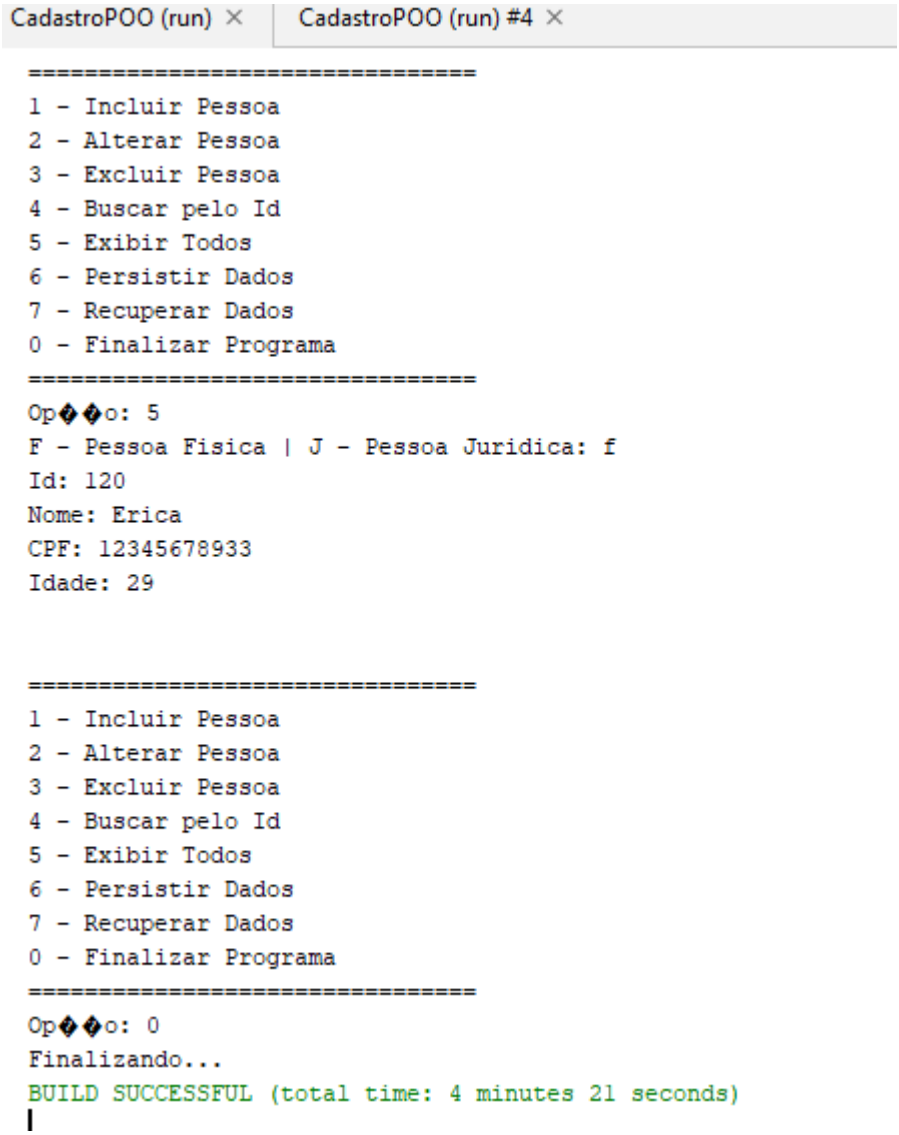
```
                case 2: // ALTERAR
                    System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
                    String tipo2 = scanner.next().toUpperCase();
                    System.out.print("Digite o id da pessoa: ");
                    int id2 = scanner.nextInt();

                    if (tipo2.equals("F")) {
                        PessoaFisica p = repoFisica.obter(id2);
                        if (p != null) {
                            System.out.println("Dados atuais:"); p.exibir();
                            System.out.print("Novo Nome: "); String n = scanner.next();
                            System.out.print("Novo CPF: "); String c = scanner.next();
                            System.out.print("Nova Idade: "); int i = scanner.nextInt();
                            repoFisica.alterar(new PessoaFisica(id2, n, c, i));
                        }
                    } else if (tipo2.equals("J")) {
                        PessoaJuridica p = repoJuridica.obter(id2);
                        if (p != null) {
                            System.out.println("Dados atuais:"); p.exibir();
                            System.out.print("Novo Nome: "); String n = scanner.next();
                            System.out.print("Novo CNPJ: "); String c = scanner.next();
                            repoJuridica.alterar(new PessoaJuridica(id2, n, c));
                        }
                    }
                    break;

                case 3: // EXCLUIR
                    System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
                    String tipo3 = scanner.next().toUpperCase();
                    System.out.print("Digite o id da pessoa: ");
                    int id3 = scanner.nextInt();
                    if (tipo3.equals("F")) repoFisica.excluir(id3);
                    else if (tipo3.equals("J")) repoJuridica.excluir(id3);
                    break;
```

```
                case 4: // BUSCAR
                    System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
                    String tipo4 = scanner.next().toUpperCase();
                    System.out.print("Digite o id da pessoa: ");
                    int id4 = scanner.nextInt();
                    if (tipo4.equals("F")) {
                        PessoaFisica p = repoFisica.obter(id4);
                        if(p!=null) p.exibir();
                    } else if (tipo4.equals("J")) {
                        PessoaJuridica p = repoJuridica.obter(id4);
                        if(p!=null) p.exibir();
                    }
                    break;

                case 5: // EXIBIR TODOS
                    System.out.print("F - Pessoa Fisica | J - Pessoa Juridica: ");
                    String tipo5 = scanner.next().toUpperCase();
                    if (tipo5.equals("F")) for(PessoaFisica p : repoFisica.obterTodos()) { p.exibir(); System.out.println(); }
                    else if (tipo5.equals("J")) for(PessoaJuridica p : repoJuridica.obterTodos()) { p.exibir(); System.out.println(); }
                    break;

                case 6: // SALVAR
                    System.out.print("Digite o prefixo dos arquivos: ");
                    String prefixoSalvar = scanner.next();
                    repoFisica.persistir(prefixoSalvar + ".fisica.bin");
                    repoJuridica.persistir(prefixoSalvar + ".juridica.bin");
                    System.out.println("Dados salvos.");
                    break;

                case 7: // RECUPERAR
                    System.out.print("Digite o prefixo dos arquivos: ");
                    String prefixoRec = scanner.next();
                    repoFisica.recuperar(prefixoRec + ".fisica.bin");
                    repoJuridica.recuperar(prefixoRec + ".juridica.bin");
                    System.out.println("Dados recuperados.");
                    break;

                case 0:
                    System.out.println("Finalizando...");
                    break;
```

```
130
131                         default:
132                             System.out.println("Opção inválida.");
133                     }
                    } catch (Exception e) {
135                     System.out.println("Erro: " + e.getMessage());
136                 }
137             }
138             scanner.close();
139         }
140 }
```

Output da Missão 2

```
Output - CadastroPOO (run)
run:

==================================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
==================================
Op��o: 1
F - Pessoa Fisica | J - Pessoa Juridica: f
Digite o id da pessoa: 120
Insira os dados...
Nome: Erica
CPF: 12345678933
Idade: 29

==================================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
==================================
Op��o:
```



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

Elementos estáticos (static) em Java são membros (atributos ou métodos) que pertencem à classe em si, e não a uma instância específica (objeto) dessa classe. Isso significa que eles podem ser acessados sem a necessidade de criar um objeto. O método main adota esse modificador porque ele é o ponto de entrada da aplicação; a Java Virtual Machine (JVM) precisa invocá-lo para iniciar o programa antes que qualquer objeto da classe principal (CadastroPOO) tenha sido criado (instanciado).

b. Para que serve a classe Scanner?

Está presente no pacote java.util, é utilizada para ler e analisar textos (strings) e tipos primitivos usando expressões regulares. Neste trabalho prático, ela foi utilizada para capturar os dados de entrada fornecidos pelo usuário através do console (fluxo de entrada padrão System.in), permitindo a interatividade do menu (leitura de opções numéricas, nomes, CPFs, etc.).

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

O uso das classes de repositório (PessoaFisicaRepo e PessoaJuridicaRepo) foi fundamental para aplicar o princípio de Separação de Responsabilidades. Elas encapsularam toda a lógica de manipulação de dados (inserir, excluir, buscar) e a complexidade da persistência em arquivos binários. Com isso, a classe principal (CadastroPOO) ficou responsável apenas pela interface com o usuário (Menu), tornando o código mais organizado, legível e facilitando a manutenção futura.

Conclusão

A execução desta Missão Prática permitiu consolidar os fundamentos essenciais da Programação Orientada a Objetos (POO) e sua aplicação em sistemas de persistência de dados. A implementação do sistema "CadastroPOO" demonstrou a eficácia do uso de herança e polimorfismo, permitindo a criação de um código modular e reutilizável, onde a classe genérica Pessoa centralizou atributos comuns, facilitando a manutenção e a extensão para as entidades PessoaFisica e PessoaJuridica.

Do ponto de vista da persistência, o uso da interface Serializable em conjunto com os fluxos de entrada e saída (ObjectInputStream e ObjectOutputStream) provou ser uma solução nativa e eficiente para o armazenamento de objetos em arquivos binários. Embora essa abordagem não substitua um Banco de Dados Relacional (SGBD) em cenários de alta complexidade ou concorrência, ela é fundamental para compreender como a Java Virtual Machine (JVM) manipula o estado dos objetos e a importância da serialização no tráfego de dados.

A organização do projeto, com a separação clara entre as classes de modelo (Model), as classes de repositório (lógica de acesso a dados) e a classe principal (interface com o usuário), evidenciou a importância de padrões de projeto e da separação de responsabilidades. Isso resultou em um código limpo, onde a lógica de negócio não se mistura com a lógica de apresentação.

Por fim, o desenvolvimento do menu interativo em modo texto proporcionou uma visão prática sobre o tratamento de exceções e o controle de fluxo, reforçando a importância de validar entradas do usuário para garantir a robustez da aplicação. O trabalho foi fundamental para servir de base sólida para futuras implementações.