POLO COHABINAL - PARNAMIRIM - RN

CURSO: **DESENVOLVIMENTO FULL STACK**
DISCIPLINA: **INICIANDO O CAMINHO PELO JAVA**
TURMA: **2024.1** SEMESTRE: **3°**
ALUNO: **ENILDO ARAÚJO DE OLIVEIRA JÚNIOR** MAT.: **202402159101**
GIT: **https://github.com/Enildo/java1**

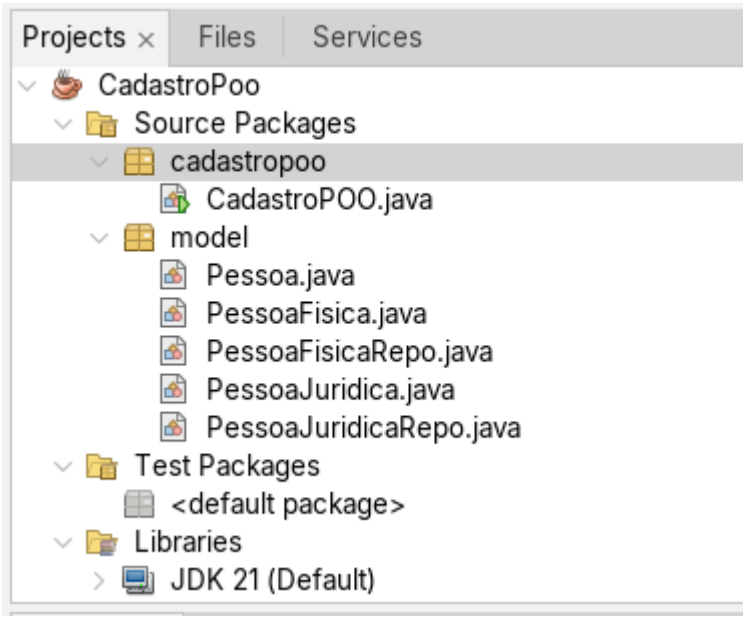
Missão Prática - Nível 1

Objetivos

1. Utilizar herança e polimorfismo na definição de entidades.
2. Utilizar persistência de objetos em arquivos binários.
3. Implementar uma interface cadastral em modo texto.
4. Utilizar o controle de exceções da plataforma Java.
5. No final do projeto, o aluno terá implementado um sistema cadastral em Java,
6. utilizando os recursos da programação orientada a objetos e a persistência
7. em arquivos binários.

CÓDIGOS FONTE

Estrutura - IDE NetBeans



Main
CadastroPOO.java

```
package cadastropoo;

import java.util.Scanner;
import model.*;

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

```
Scanner scanner = new Scanner(System.in);
PessoaFisicaRepo pfRepo = new PessoaFisicaRepo();
PessoaJuridicaRepo pjRepo = new PessoaJuridicaRepo();

int opcao;
do {
    System.out.println("\n================================");
    System.out.println("1 - Incluir Pessoa");
    System.out.println("2 - Alterar Pessoa");
    System.out.println("3 - Excluir Pessoa");
    System.out.println("4 - Buscar pelo Id");
    System.out.println("5 - Exibir Todos");
    System.out.println("6 - Persistir Dados");
    System.out.println("7 - Recuperar Dados");
    System.out.println("0 - Finalizar Programa");
    System.out.println("================================");
    System.out.print("Opção: ");
    opcao = Integer.parseInt(scanner.nextLine());

    switch (opcao) {
        case 1 -> {
            System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
            String tipo = scanner.nextLine().toUpperCase();
            System.out.print("Digite o id da pessoa: ");
            int id = Integer.parseInt(scanner.nextLine());
            System.out.println("Insira os dados...");
            System.out.print("Nome: ");
            String nome = scanner.nextLine();
            if (tipo.equals("F")) {
                System.out.print("CPF: ");
                String cpf = scanner.nextLine();
                System.out.print("Idade: ");
                int idade = Integer.parseInt(scanner.nextLine());
                pfRepo.inserir(new PessoaFisica(id, nome, cpf, idade));
            } else if (tipo.equals("J")) {
                System.out.print("CNPJ: ");
                String cnpj = scanner.nextLine();
                pjRepo.inserir(new PessoaJuridica(id, nome, cnpj));
            }
        }

        case 2 -> {
            System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
            String tipo = scanner.nextLine().toUpperCase();
            System.out.print("Digite o id da pessoa: ");
            int id = Integer.parseInt(scanner.nextLine());
            if (tipo.equals("F")) {
                PessoaFisica pf = pfRepo.obter(id);
                if (pf != null) {
                    pf.exibir();
                    System.out.print("Novo nome: ");
                    String nome = scanner.nextLine();
                    System.out.print("Novo CPF: ");
                    String cpf = scanner.nextLine();
                    System.out.print("Nova idade: ");
                    int idade = Integer.parseInt(scanner.nextLine());
                    pfRepo.alterar(new PessoaFisica(id, nome, cpf, idade));
                }
            } else if (tipo.equals("J")) {
                PessoaJuridica pj = pjRepo.obter(id);
```

```
            if (pj != null) {
                pj.exibir();
                System.out.print("Novo nome: ");
                String nome = scanner.nextLine();
                System.out.print("Novo CNPJ: ");
                String cnpj = scanner.nextLine();
                pjRepo.alterar(new PessoaJuridica(id, nome, cnpj));
            }
        }
    }

    case 3 -> {
        System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
        String tipo = scanner.nextLine().toUpperCase();
        System.out.print("Digite o id da pessoa: ");
        int id = Integer.parseInt(scanner.nextLine());
        if (tipo.equals("F")) {
            pfRepo.excluir(id);
        } else if (tipo.equals("J")) {
            pjRepo.excluir(id);
        }
    }

    case 4 -> {
        System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
        String tipo = scanner.nextLine().toUpperCase();
        System.out.print("Digite o id da pessoa: ");
        int id = Integer.parseInt(scanner.nextLine());
        if (tipo.equals("F")) {
            PessoaFisica pf = pfRepo.obter(id);
            if (pf != null) pf.exibir();
        } else if (tipo.equals("J")) {
            PessoaJuridica pj = pjRepo.obter(id);
            if (pj != null) pj.exibir();
        }
    }

    case 5 -> {
        System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");
        String tipo = scanner.nextLine().toUpperCase();
        if (tipo.equals("F")) {
            for (PessoaFisica pf : pfRepo.obterTodos()) {
                pf.exibir();
                System.out.println();
            }
        } else if (tipo.equals("J")) {
            for (PessoaJuridica pj : pjRepo.obterTodos()) {
                pj.exibir();
                System.out.println();
            }
        }
    }

    case 6 -> {
        System.out.print("Digite o prefixo do arquivo: ");
        String prefixo = scanner.nextLine();
        try {
            pfRepo.persistir(prefixo + ".fisica.bin");
            pjRepo.persistir(prefixo + ".juridica.bin");
            System.out.println("Dados salvos com sucesso.");
```

```
                } catch (Exception e) {
                    System.out.println("Erro ao salvar dados: " + e.getMessage());
                }
            }

            case 7 -> {
                System.out.print("Digite o prefixo do arquivo: ");
                String prefixo = scanner.nextLine();
                try {
                    pfRepo.recuperar(prefixo + ".fisica.bin");
                    pjRepo.recuperar(prefixo + ".juridica.bin");
                    System.out.println("Dados recuperados com sucesso.");
                } catch (Exception e) {
                    System.out.println("Erro ao recuperar dados: " + e.getMessage());
                }
            }

            case 0 -> System.out.println("Programa finalizado.");

            default -> System.out.println("Opção inválida.");
        }
    } while (opcao != 0);

    scanner.close();
  }
}
```

Modelos

Pessoa.java

```
package model;

import java.io.Serializable;

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; }
}
```

PessoaFisica.java

```
package model;

public 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;
    }

    @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; }
}
```

PessoaFisicaRepo.java

```
package model;

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

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

    public void inserir(PessoaFisica p) { pessoas.add(p); }
    public void alterar(PessoaFisica p) {
        excluir(p.getId());
        inserir(p);
    }
    public void excluir(int id) { pessoas.removeIf(p -> p.getId() == id); }
    public PessoaFisica obter(int id) {
        return pessoas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
    }
    public List<PessoaFisica> obterTodos() { return pessoas; }

    public void persistir(String nomeArquivo) throws Exception {
        try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo)))
{
            oos.writeObject(pessoas);
        }
    }

    public void recuperar(String nomeArquivo) throws Exception {
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoas = (List<PessoaFisica>) ois.readObject();
        }
    }
}
```

PessoaJuridica.java

```
package model;
```

```
public class PessoaJuridica extends Pessoa {
   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; }
}
```

PessoaJuridicaRepo.java

```
package model;

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

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

   public void inserir(PessoaJuridica p) { pessoas.add(p); }
   public void alterar(PessoaJuridica p) {
      excluir(p.getId());
      inserir(p);
   }
   public void excluir(int id) { pessoas.removeIf(p -> p.getId() == id); }
   public PessoaJuridica obter(int id) {
      return pessoas.stream().filter(p -> p.getId() == id).findFirst().orElse(null);
   }
   public List<PessoaJuridica> obterTodos() { return pessoas; }

   public void persistir(String nomeArquivo) throws Exception {
       try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(nomeArquivo)))
{
          oos.writeObject(pessoas);
       }
   }

   public void recuperar(String nomeArquivo) throws Exception {
      try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
         pessoas = (List<PessoaJuridica>) ois.readObject();
      }
   }
}
```

EXECUÇÃO - RESULTADOS

```
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
==========
Opção: 5
F - Pessoa Fisica | J - Pessoa Juridica
F

==========
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
==========
Opção: 1
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o id da pessoa: 01
Insira os dados...
Nome: ENILDO
CPF: 123.123.123-11
Idade: 23

==========
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
==========
Opção: 1
F - Pessoa Fisica | J - Pessoa Juridica
J
Digite o id da pessoa: 01
Insira os dados...
Nome: EULA LTDA
CNPJ: 12.123.123/0001-12

==========
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

==========
Opção: 1
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o id da pessoa: 02
Insira os dados...
Nome: JUNIOR
CPF: 222.333.444-55
Idade: 34

==========
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
==========
Opção: 3
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o id da pessoa: 02

==========
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
==========
Opção: 4
F - Pessoa Fisica | J - Pessoa Juridica
F
Digite o id da pessoa: 01
Id: 1
Nome: ENILDO
CPF: 123.123.123-11
Idade: 23

==========
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
==========
Opção: 5
F - Pessoa Fisica | J - Pessoa Juridica
F
Id: 1
Nome: ENILDO
CPF: 123.123.123-11

Idade: 23

==========
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
==========
Opção: 5
F - Pessoa Fisica | J - Pessoa Juridica
J
Id: 1
Nome: EULA LTDA
CNPJ: 12.123.123/0001-12

==========
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
==========
Opção: 6
Digite o prefixo do arquivo: pratica
Dados salvos com sucesso.

==========
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
==========
Opção: 7
Digite o prefixo do arquivo: pratica
Dados recuperados com sucesso.

==========
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
==========
Opção: 0

Programa finalizado.
BUILD SUCCESSFUL (total time: 2 minutes 38 seconds)

ANÁLISE E CONCLUSÃO

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

Vantagens:

A herança traz várias facilidades no desenvolvimento, principalmente pela **reutilização de código**. Ao centralizar atributos e métodos em uma superclasse, evitamos repetições desnecessárias nas subclasses. Além disso, facilita bastante a manutenção: se for preciso mudar algo comum a várias classes, basta alterar na superclasse.

Outro ponto positivo é o **polimorfismo**, que permite trabalhar com objetos de diferentes subclasses como se fossem da superclasse. Isso torna o código mais flexível e fácil de expandir. Um exemplo seria a classe `Pessoa` como base, sendo estendida por `PessoaFisica` e `PessoaJuridica`.

Desvantagens:

Apesar de útil, a herança também tem seus pontos negativos. O principal é o **forte acoplamento** entre as classes: mudanças na superclasse podem acabar afetando todas as subclasses, o que exige muito cuidado.

Além disso, nem sempre é fácil reaproveitar subclasses fora da hierarquia em que foram criadas. Também é possível acabar quebrando princípios importantes, como o de substituição de Liskov, se a herança não for bem planejada. E quando a hierarquia fica muito profunda, o código pode ficar difícil de entender e testar.

---

b) Por que a interface `Serializable` é necessária ao gravar dados em arquivos binários?

No Java, para que um objeto possa ser salvo em um arquivo binário (ou transmitido pela rede), ele precisa ser transformado em uma sequência de bytes. A interface `Serializable` serve exatamente para isso: ela informa à JVM que a classe pode ser "quebrada" em bytes.

Se uma classe não implementar `Serializable`, o Java simplesmente não consegue salvar ou recuperar seus objetos desse tipo de arquivo.

---

c) Como o paradigma funcional é utilizado na API Stream do Java?

A API `Stream` do Java traz vários conceitos da programação funcional. Um deles é a **imutabilidade**, já que as operações feitas com stream não alteram a estrutura original dos dados.

Também usamos **funções como argumentos**, especialmente com expressões lambda. E o mais legal é que a abordagem é **declarativa** — a gente diz *o que* quer fazer com os dados, e não *como* exatamente fazer. Isso deixa o código mais limpo e direto ao ponto.

---

d) Qual padrão de desenvolvimento é utilizado na persistência de dados com arquivos no Java?

O padrão mais comum é o **DAO (Data Access Object)**. Ele separa a parte de acesso aos dados da lógica principal da aplicação. Isso ajuda bastante na organização, facilita testes e manutenção, e deixa o código mais limpo.

No nosso projeto, as classes `PessoaFisicaRepo` e `PessoaJuridicaRepo` funcionam como DAOs, cuidando da inclusão, exclusão, alteração e salvamento dos dados.

---

e) O que são elementos estáticos e por que o método `main` é `static`?

Elementos `static` pertencem à **classe** e não a um objeto específico. O método `main` precisa ser `static` porque ele é o ponto de entrada da aplicação — ou seja, o Java precisa executá-lo **sem precisar criar uma instância da classe**.

---

f) Para que serve a classe `Scanner`?

A classe `Scanner` é usada para **ler dados digitados pelo usuário** (ou até de arquivos e strings). No nosso caso, ela é fundamental para capturar informações no console, usando métodos como `nextLine()`, `nextInt()`, entre outros.

---

g) Como o uso de classes de repositório ajudou na organização do código?

As classes de repositório, como `PessoaFisicaRepo` e `PessoaJuridicaRepo`, foram essenciais para manter o código bem organizado. Elas **isolam a lógica de persistência**, deixando a parte da interface (como o menu principal) mais limpa e fácil de entender.

Com isso, ficou mais fácil **manter** e **reaproveitar** o código, além de permitir mudanças futuras (como trocar arquivos por um banco de dados) com o mínimo de impacto. Isso segue boas práticas, como o padrão DAO.