Faculdade Estácio - POLO FREGUESIA - RIO DE JANEIRO - RJ

Curso: Desenvolvimento Full Stack
Disciplina: Iniciando O Caminho Pelo Java
Número da Turma: 9001
Semestre Letivo: 2025.1
Integrante: Gabriel Galvão Reis
Repositório: https://github.com/galvao-reis/estacio-missao-pratica-3-1

Iniciando o Caminho pelo Java

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

Objetivo da Prática

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, utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

Sumário

Arquivos desenvolvidos pela Prática

1. cadastroPOO.java

```java
package cadastropoo;

import model.*;
import java.io.*;

public class CadastroPOO {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

            repo1.inserir(new PessoaFisica(1,"Ana","11111111111",25));
            repo1.inserir(new PessoaFisica(2,"Carlos","22222222222",52));
            try{
                repo1.persistir( "RepositorioPessoasFisicas" );
            }
            catch(IOException exception){
                System.out.println(exception.toString());
            }

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            try{
                repo2.recuperar( "RepositorioPessoasFisicas" );
            }
            catch(IOException | ClassNotFoundException exception){
                System.out.println(exception.toString());
            }

            for (PessoaFisica pessoa : repo2.obterTodos()){
                pessoa.exibir();
            }

            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
            repo3.inserir(new PessoaJuridica(3,"XPTO Sales","33333333333333"));
            repo3.inserir(new PessoaJuridica(4,"XPTO Solutions","44444444444444444"));

            try{
                repo3.persistir( "RepositorioPessoasJuridicas" );
            }
            catch(IOException exception){
                System.out.println(exception.toString());
            }

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            try{
                repo4.recuperar( "RepositorioPessoasJuridicas" );
            }
            catch( IOException | ClassNotFoundException exception){
                System.out.println(exception.toString());
            }

            for (PessoaJuridica pessoa : repo4.obterToddos()){
                pessoa.exibir();
            }
        }
    }
```

2. Pessoa.java

```java
package model;

import java.io.Serializable;
/**
 *
 * @author Gabriel
 */
public class Pessoa implements Serializable {

    private int id;
    private String nome;

    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;
    }

    public void exibir(){
        System.out.println( "id: " + this.getId() + ", nome: " + this.getNome());
    }
}
```

3. PessoaFisica.java

```java
package model;

import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {

    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;
    }

    public String getCpf(){
        return this.cpf;
    }

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

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

    @Override
    public void exibir(){
        System.out.println(
                "Id: "+ this.getId() +
                "\nNome: " + this.getNome() +
                "\nCPF: " + this.getCpf() +
                ",\nIdade: " + this.getIdade());
    }

}

```

4. PessoaJuridica.java

```java
package model;

import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {

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

    public String getCnpj(){
        return this.cnpj;
    }

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

    @Override
    public void exibir(){
        System.out.println(
                    "Id: " + this.getId() +
                    "\nNome: " + this.getNome() +
                    "\nCNPJ: " + this.getCnpj());
    }

}

```

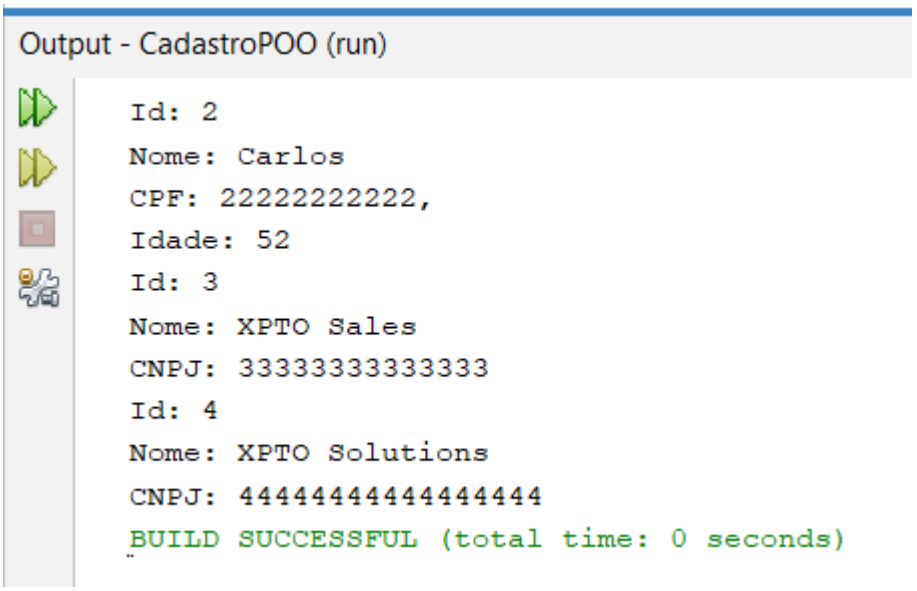
5. PessoaFisicaRepo.java

```java
package model;

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

public class PessoaFisicaRepo{
    private ArrayList<PessoaFisica> pessoasFisicas;

    public PessoaFisicaRepo(){
        this.pessoasFisicas = new ArrayList<PessoaFisica>();
    }

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

    public void alterar( PessoaFisica pessoa ){
        this.excluir(pessoa.getId());
        this.inserir(pessoa);
    }

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

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

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

    public void persistir(String nomeArquivo)throws IOException{
        FileOutputStream fos = new FileOutputStream(nomeArquivo);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(this.pessoasFisicas);
    }

    public void recuperar ( String nomeArquivo) throws IOException, ClassNotFoundException{
        FileInputStream fis = new FileInputStream(nomeArquivo);
        ObjectInputStream ois = new ObjectInputStream(fis);
        this.pessoasFisicas = (ArrayList<PessoaFisica>) ois.readObject();
    }
}

```

6. PessoaJuridicaRepo.java

```java
package model;

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

public class PessoaJuridicaRepo{

    private ArrayList<PessoaJuridica> pessoasJuridicas;

    public PessoaJuridicaRepo(){
        this.pessoasJuridicas = new ArrayList<PessoaJuridica>();
    }

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

    public void alterar(PessoaJuridica pessoa){
        this.excluir(pessoa.getId());
        this.inserir(pessoa);
    }

    public void excluir( int id ){
        this.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 this.pessoasJuridicas;
    }

    public void persistir( String nomeArquivo ) throws IOException{
        FileOutputStream fos = new FileOutputStream(nomeArquivo);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(this.pessoasJuridicas);
    }

    public void recuperar( String nomeArquivo) throws IOException, ClassNotFoundException {
        FileInputStream fis = new FileInputStream(nomeArquivo);
        ObjectInputStream ois = new ObjectInputStream(fis);
        this.pessoasJuridicas = (ArrayList<PessoaJuridica>) ois.readObject();

    }

}

```

Resultado da execução do código



Análise e Conclusão

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

A herança permite a reutilização de código. Por permitir que as classes filhas tenham as mesmas propriedades e atributos da classe mãe, ela evita que haja duplicação de código, facilitando a manutenção e implementação de eventuais modificações.

No entanto, com o aumento da complexidade do projeto, a herança pode dificultar a modificação e manutenção do código, uma vez que é possível se criar hierarquias complexas. Outro problema da herança é que ela quebra o princípio de encapsulamento, já que a classe filha sempre depende da classe mãe, tornando o sistema mais frágil.

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

A interface Serializable é necessária pois ela sinaliza que os objetos da classe que a implementa são capazes de serem serializados. Caso não implementada, a tentativa de serializar um objeto irá causar uma exceção do tipo “NotSerializableException”

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

O API Stream do Java é utilizado para processar coleções de objetos. Uma Stream é uma sequência de objetos que suportam diversos métodos que podem ser encadeados para gerar o resultado desejado sem que ocorram modificações nos dados originais.

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

Para tratar da persistência de dados em arquivos, utiliza-se o JPA - Java Persistence API. No entanto, ele requer que se utilize um provedor, tal quais Hibernate, Apache OpenJPA e ObjectDB dentre outros.