

UNIVERSIDADE ESTÁCIO DE SÁ

FULLSTACK

Nível 1: Iniciando o Caminho Pelo Java

MAIARA ACCACIO MACHADO

202204268183 | Turma 9001

RIO DE JANEIRO – RJ

2023

# SUMÁRIO

[SUMÁRIO 1](#_Toc140703274)

[1. INTRODUÇÃO 2](#_Toc140703275)

[1.1 OBJETIVOS DA PRÁTICA 2](#_Toc140703276)

[2. PROCEDIMENTO 1 3](#_Toc140703277)

[2.1 Classe Pessoa 3](#_Toc140703278)

[2.2 Classe PessoaFisica 3](#_Toc140703279)

[2.3 Classe PessoaJuridica 4](#_Toc140703280)

[2.4 Classe PessoaFisicaRepo 5](#_Toc140703281)

[2.5 Classe PessoaJuridicaRepo 7](#_Toc140703282)

[2.6 Classe Principal 10](#_Toc140703283)

[2.7 Saída da execução 11](#_Toc140703284)

[2.8 Análise e Conclusão 12](#_Toc140703285)

# INTRODUÇÃO

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

## OBJETIVOS DA PRÁTICA

* Utilizar herança e polimorfismo na definição de entidades.
* Utilizar persistência de objetos em arquivos binários.
* Implementar uma interface cadastral em modo texto.
* Utilizar o controle de exceções da plataforma Java.
* 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.

# PROCEDIMENTO 1

O procedimento consiste na Criação das Entidades e Sistema de Persistência

## Classe Pessoa

package model;  
  
import java.io.Serializable;  
  
public class Pessoa implements Serializable {  
 private int id;  
 private String nome;  
  
 public Pessoa(int id, String nome){  
 this.nome = nome;  
 this.id = id;  
 }  
  
 public void exibir(){  
 System.*out*.println("Id: "+this.getId());  
 System.*out*.println("Nome: "+this.getNome());  
 }  
  
 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;  
 }  
}

## Classe PessoaFisica

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 cpf;  
 }  
  
 public void setCpf(String cpf) {  
 this.cpf = cpf;  
 }  
  
 public int getIdade() {  
 return idade;  
 }  
  
 public void setIdade(int idade) {  
 this.idade = idade;  
 }  
  
 public void exibir(){  
 super.exibir();  
 System.*out*.println("CPF: "+this.getCpf());  
 System.*out*.println("Idade: "+this.getIdade());  
 }  
}

## Classe PessoaJuridica

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 cnpj;  
 }  
  
 public void setCnpj(String cnpj) {  
 this.cnpj = cnpj;  
 }  
  
 public void exibir(){  
 super.exibir();  
 System.*out*.println("CNPJ: "+this.getCnpj());  
 }  
}

## Classe PessoaFisicaRepo

package model;  
  
  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.util.ArrayList;  
  
  
public class PessoaFisicaRepo {  
 private ArrayList<PessoaFisica> pessoasFisicas;  
  
 public PessoaFisicaRepo() {  
 pessoasFisicas = new ArrayList<>();  
 }  
  
 //*TODO: método inserir* public void inserir(PessoaFisica pessoaFisica) {  
 pessoasFisicas.add(pessoaFisica);  
 System.*out*.println("Cadastro realizado com sucesso.");  
  
 }  
  
 //*TODO: método alterar* public void alterar(PessoaFisica pessoaFisica) {  
 int id = pessoaFisica.getId();  
 excluir(id);  
 pessoasFisicas.add(pessoaFisica);  
 System.*out*.println("Cadastro atualizado com sucesso.");  
 }  
  
 //*TODO: método excluir* public void excluir(int id) {  
 PessoaFisica pfexclusao = null;  
 for (PessoaFisica pessoafisica : pessoasFisicas) {  
 if (pessoafisica.getId() == id) {  
 pfexclusao = pessoafisica;  
 break;  
 }  
 }  
 if (pfexclusao != null) {  
 pessoasFisicas.remove(pfexclusao);  
 System.*out*.println("Cadastro excluído com sucesso!");  
 } else {  
 System.*out*.println("Não foi possível excluir o cadastro. Id não encontrado.");  
 }  
 }  
  
  
 //*TODO: método obter* public void obter(int id) {  
 boolean encontrado = false;  
 for (PessoaFisica pessoaFisica : pessoasFisicas  
 ) {  
 if (pessoaFisica.getId() == id) {  
 pessoaFisica.exibir();  
 encontrado = true;  
  
 break;  
 }  
 }  
 if (!encontrado) {  
 System.*out*.println("Cadastro não encontrado. Por favor, verifique o id.");  
 }  
 }  
  
 //*TODO: método obterTodos* public void obterTodos() {  
 System.*out*.println("------- Lista de Pessoas Físicas cadastradas -------");  
 for (PessoaFisica pessoaFisica : pessoasFisicas  
 ) {  
 pessoaFisica.exibir();  
 System.*out*.println("------------------------");  
 }  
 }  
  
  
 //*TODO: método persistir* public void persistir(String arquivo) throws Exception {  
 try (FileOutputStream saida = new FileOutputStream(arquivo);  
 ObjectOutputStream objeto = new ObjectOutputStream(saida)) {  
  
 objeto.writeObject(pessoasFisicas);  
 System.*out*.println("Dados de Pessoa Física Armazenados.");  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 public void recuperar(String arquivo) throws Exception {  
 try (FileInputStream entrada = new FileInputStream(arquivo);  
 ObjectInputStream objeto = new ObjectInputStream(entrada)) {  
  
 pessoasFisicas = (ArrayList<PessoaFisica>) objeto.readObject();  
 System.*out*.println("Dados de Pessoa Física recuperados.");  
  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

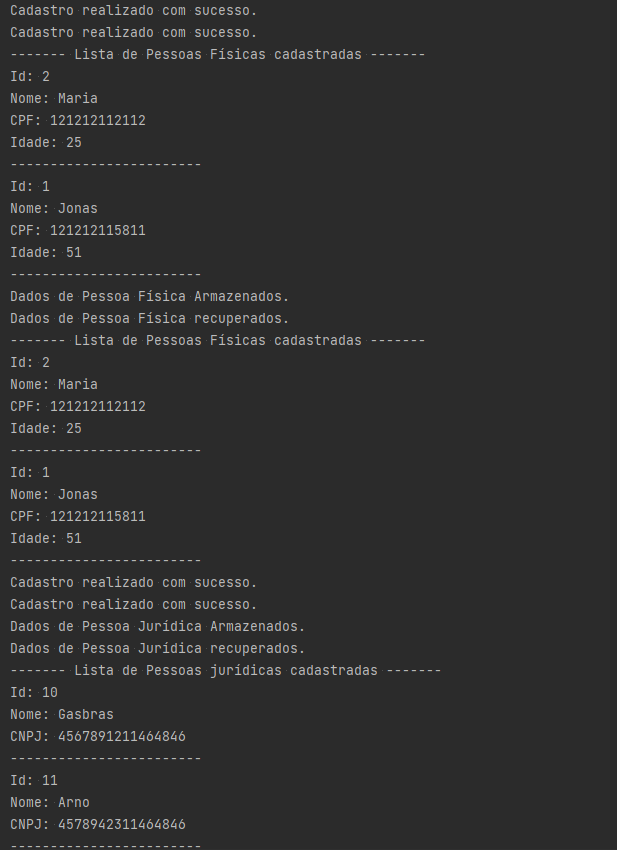
## Classe PessoaJuridicaRepo

package model;  
  
import java.io.FileInputStream;  
import java.io.FileOutputStream;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.util.ArrayList;  
  
public class PessoaJuridicaRepo {  
  
 private ArrayList<PessoaJuridica> pessoasJuridicas;  
  
 public PessoaJuridicaRepo() {  
 pessoasJuridicas = new ArrayList<>();  
 }  
  
 //*TODO: método inserir* public void inserir(PessoaJuridica pessoaJuridica) {  
 pessoasJuridicas.add(pessoaJuridica);  
 System.*out*.println("Cadastro realizado com sucesso.");  
  
 }  
  
 //*TODO: método alterar* public void alterar(PessoaJuridica pessoaJuridica) {  
 int id = pessoaJuridica.getId();  
 excluir(id);  
 pessoasJuridicas.add(pessoaJuridica);  
 System.*out*.println("Cadastro atualizado com sucesso.");  
 }  
  
 //*TODO: método excluir* public void excluir(int id) {  
 PessoaJuridica pjexclusao = null;  
 for (PessoaJuridica pessoafisica : pessoasJuridicas) {  
 if (pessoafisica.getId() == id) {  
 pjexclusao = pessoafisica;  
 break;  
 }  
 }  
 if (pjexclusao != null) {  
 pessoasJuridicas.remove(pjexclusao);  
 System.*out*.println("Cadastro excluído com sucesso!");  
 } else {  
 System.*out*.println("Não foi possível excluir o cadastro. Id não encontrado.");  
 }  
 }  
  
  
 //*TODO: método obter* public void obter(int id) {  
 boolean encontrado = false;  
 for (PessoaJuridica pessoaJuridica : pessoasJuridicas  
 ) {  
 if (pessoaJuridica.getId() == id) {  
 pessoaJuridica.exibir();  
 encontrado = true;  
  
 break;  
 }  
 }  
 if (!encontrado) {  
 System.*out*.println("Cadastro não encontrado. Por favor, verifique o id.");  
 }  
 }  
  
 //*TODO: método obterTodos* public void obterTodos() {  
 System.*out*.println("------- Lista de Pessoas jurídicas cadastradas -------");  
 for (PessoaJuridica pessoaFisica : pessoasJuridicas  
 ) {  
 pessoaFisica.exibir();  
 System.*out*.println("------------------------");  
 }  
 }  
  
  
 //*TODO: método persistir* public void persistir(String arquivo) throws Exception {  
 try (FileOutputStream saida = new FileOutputStream(arquivo);  
 ObjectOutputStream objeto = new ObjectOutputStream(saida)) {  
  
 objeto.writeObject(pessoasJuridicas);  
 System.*out*.println("Dados de Pessoa Jurídica Armazenados.");  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 public void recuperar(String arquivo) throws Exception {  
 try (FileInputStream entrada = new FileInputStream(arquivo);  
 ObjectInputStream objeto = new ObjectInputStream(entrada)) {  
  
 pessoasJuridicas = (ArrayList<PessoaJuridica>) objeto.readObject();  
 System.*out*.println("Dados de Pessoa Jurídica recuperados.");  
  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

## Classe Principal

package model;  
  
import java.util.List;  
  
public class Principal {  
 public static void main(String[] args) throws Exception {  
 PessoaFisicaRepo repo1 = new PessoaFisicaRepo();  
 PessoaFisicaRepo repo2 = new PessoaFisicaRepo();  
  
 PessoaFisica jonas = new PessoaFisica(1, "Jonas", "121212115811", 51);  
 PessoaFisica maria = new PessoaFisica(2, "Maria", "121212112112", 25);  
  
 repo1.inserir(maria);  
 repo1.inserir(jonas);  
  
 repo1.obterTodos();  
 repo1.persistir("pessoasFisicas");  
 repo2.recuperar("pessoasFisicas");  
 repo2.obterTodos();  
  
  
 PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();  
  
 PessoaJuridica gasbras = new PessoaJuridica(10, "Gasbras", "4567891211464846");  
 PessoaJuridica arno = new PessoaJuridica(11, "Arno", "4578942311464846");  
 repo3.inserir(gasbras);  
 repo3.inserir(arno);  
  
 repo3.persistir("pessoasJuridicas");  
  
 PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();  
  
 repo4.recuperar("pessoasJuridicas");  
 repo4.obterTodos();  
  
 }  
}

## Saída da execução



## Análise e Conclusão

**Quais as vantagens e desvantagens do uso de herança?**

Como vantagem temos a possibilidade de reutilização de códigos, evitando duplicidade de código e desorganização, o que facilita a manutenção. Além disso, a herança oferece flexibilidade por permitir o uso de novos atributos e métodos nas classes filhas sem interferência na estrutura da classe Pai. Outra vantagem é o polimorfismo que é possibilitado pela herança.

Como desvantagens temos o acoplamento entre a classe Pai e seus filhos, o que pode gerar impactos indesejados quando uma alteração é feita na classe Pai. Outra desvantagem é a alta complexidade que pode ser ocasionada por uma grande quantidade de heranças.

**Por que a interface Serializable é necessária ao efetuar persistência em**

**arquivos binários?**

A interface Serializable indica que uma classe pode ser convertida em uma sequência de bytes (serialização) para ser salva no arquivo e recuperada posteriormente para seu formato original no processo de deserialização.

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

Funções de ordem superior, que são funções que podem receber outras funções como argumento ou retorná-las. A API stream também utiliza operações comuns do paradigma funcional como map (transforma cada elemento da coleção em um novo valor de acordo com a definição da função), filter (filtra os elementos de acordo com a condição estabelecida), reduce (reduz a coleção a um único valor aplicando uma função acumuladora) e a lazy evaluation que adia as operações até que o resultado seja necessário.

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

O padrão mais comum é o DAO (Data Access Object) que separa a lógica de acesso a dados da lógica de negócios da aplicação.

# PROCEDIMENTO 2

O procedimento consiste na Criação do cadastro em modo texto.