Campus: 1197 - POLO CENTRO - SÃO LOURENÇO DA MATA - PE

Curso: Desenvolvimento Full Stack - Graduação Tecnóloga Disciplina: RPG0014 - Iniciando o caminho pelo Java Turma: 9001 Semestre: 2024.1

Matrícula: 2023.01.53256-6 Aluno: Gilvan Pereira de Oliveira Repositório GitHub: [GilvanPOliveira/CadastroPoo (github.com)](https://github.com/GilvanPOliveira/CadastroPoo)

# Relatório discente de acompanhamento

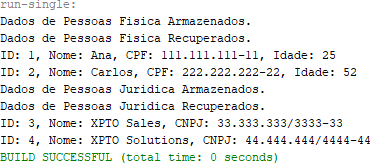
**1⁰ Procedimento - Criação das Entidades e Sistema de Persistência Objetivo da prática:**

Nesta etapa o objetivo foi criar um sistema de persistência, armazenar e recuperar, informações de entidades para um cadastro de pessoas físicas e jurídicas em Java, utilizando POO, herança, polimorfismo e manipulação de arquivos. O sistema permitirá operações de inserção, alteração, exclusão, recuperação e obtenção das entidades armazenadas, proporcionando uma maneira eficiente e segura de gerenciar e recuperar informações.

# Prática:

* Criação das Entidades: Pessoa, PessoaFisica, PessoaJuridica;
* Criação dos gerenciadores: PessoaFisicaRepo, PessoaJuridicaRepo;
* Adição dos métodos: inserir, alterar, excluir, obter e obterTodos, além de pesistir e recuperar aos gerenciadores para armazenagem dos dados no disco;
* Alterar a classe principal (main) para testar os repositórios.
* Executar e verificar as funcionalidades implementadas e os arquivos gerados.

# Códigos e resultados obtidos:

* Resultado da execução dos códigos:
* Códigos:

Criando o pacote model, e suas entidades:

Classe Pessoa:

package model;

/\*\*

\*

\* @author gilvan

\*/

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 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: " + id + ", Nome: " + nome);

}

}

Classe PessoaFisica:

package model;

/\*\*

\*

\* @author gilvan

\*/

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;

}

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;

}

@Override

public void exibir() {

System.out.println("ID: " + getId() + ", Nome: " + getNome() + ", CPF: " + cpf + ", Idade: " + idade);

}

}

Classe PessoaJuridica:

package model;

/\*\*

\*

\* @author gilvan

\*/

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;

}

public String getCnpj() {

return cnpj;

}

public void setCnpj(String cnpj) {

this.cnpj = cnpj;

}

@Override

public void exibir() {

System.out.println("ID: " + getId() + ", Nome: " + getNome() + ", CNPJ: " + cnpj);

}

}

Agora criando os gerenciadores:

Classe PessoaFisicaRepo:

package model;

/\*\*

\*

\* @author gilvan

\*/

import java.io.\*;

import java.util.ArrayList;

public class PessoaFisicaRepo {

private ArrayList<PessoaFisica> 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);

break;

}

}

}

public void excluir(int id) {

for (int i = 0; i < pessoas.size(); i++) {

if (pessoas.get(i).getId() == id) {

pessoas.remove(i);

break;

}

}

}

public PessoaFisica obter(int id) {

for (PessoaFisica pessoa : pessoas) {

if (pessoa.getId() == id) {

return pessoa;

}

}

return null;

}

public ArrayList<PessoaFisica> obterTodos() {

return pessoas;

}

public void persistir(String nomeArquivo) throws IOException {

FileOutputStream fileOut = new FileOutputStream(nomeArquivo);

try (ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {

objectOut.writeObject(pessoas);

}

}

public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {

FileInputStream fileIn = new FileInputStream(nomeArquivo);

try (ObjectInputStream objectIn = new ObjectInputStream(fileIn)) {

pessoas = (ArrayList<PessoaFisica>) objectIn.readObject();

}

}

}

Classe PessoaJuridicaRepo:

package model;

/\*\*

\*

\* @author gilvan

\*/

import java.io.\*;

import java.util.ArrayList;

public class PessoaJuridicaRepo {

private ArrayList<PessoaJuridica> 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);

break;

}

}

}

public void excluir(int id) {

for (int i = 0; i < pessoas.size(); i++) {

if (pessoas.get(i).getId() == id) {

pessoas.remove(i);

break;

}

}

}

public PessoaJuridica obter(int id) {

for (PessoaJuridica pessoa : pessoas) {

if (pessoa.getId() == id) {

return pessoa;

}

}

return null;

}

public ArrayList<PessoaJuridica> obterTodos() {

return pessoas;

}

public void persistir(String nomeArquivo) throws IOException {

FileOutputStream fileOut = new FileOutputStream(nomeArquivo);

try (ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {

objectOut.writeObject(pessoas);

}

}

public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {

FileInputStream fileIn = new FileInputStream(nomeArquivo);

try (ObjectInputStream objectIn = new ObjectInputStream(fileIn)) {

pessoas = (ArrayList<PessoaJuridica>) objectIn.readObject();

}

}

}

Alterando o método main para testar os repositórios:

Classe Main\_01:

package model;

/\*\*

\*

\* @author gilvan

\*/

import java.io.\*;

public class Main\_01 {

public static void main(String[] args) {

try {

//Instanciando repo1

PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

//Adicionando duas pessoas fisicas

PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Ana", "111.111.111-11", 25);

PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Carlos", "222.222.222-22", 52);

repo1.inserir(pessoaFisica1);

repo1.inserir(pessoaFisica2);

//Persistindo os dados em repo1

repo1.persistir("pessoasFisicas.dat");

System.out.println("Dados de Pessoas Fisica Armazenados.");

//Instanciando repo2

PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

//Recuperando os dados em repo2

repo2.recuperar("pessoasFisicas.dat");

//Exibindo os dados recuperados das pessoas fisicas

System.out.println("Dados de Pessoas Fisica Recuperados.");

for (PessoaFisica pessoa : repo2.obterTodos()) {

pessoa.exibir();

}

//Instanciando repo3

PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

//Adicionando duas pessoas jurídicas

PessoaJuridica pessoaJuridica1 = new PessoaJuridica(3, "XPTO Sales", "33.333.333/3333-33");

PessoaJuridica pessoaJuridica2 = new PessoaJuridica(4, "XPTO Solutions", "44.444.444/4444-44");

repo3.inserir(pessoaJuridica1);

repo3.inserir(pessoaJuridica2);

//Persistindo os dados em repo3

repo3.persistir("pessoasJuridicas.dat");

System.out.println("Dados de Pessoas Juridica Armazenados.");

//Instanciando repo4

PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

//Recuperando os dados em repo4

repo4.recuperar("pessoasJuridicas.dat");

//Exibindo os dados recuperados das pessoas juridicas

System.out.println("Dados de Pessoas Juridica Recuperados.");

for (PessoaJuridica pessoa : repo4.obterTodos()) {

pessoa.exibir();

}

} catch (IOException | ClassNotFoundException e) {

}

}

}

**Conclusão:**

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

Reutilização de código, para assim evitar redundâncias;

Polimorfismo, para melhor manipulação de objetos de diferentes tipos; Organização da estrutura de forma hierárquica.

* + Desvantagens:

Hierarquia complexa, dependendo da complexidade do código pode dificultar o entendimento;

Acoplamento entre as classes, tornando mais difícil modificar e até compreender;

Alterações na classe base podem afetar todas as classes derivadas, causando efeitos colaterais indesejados.

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

Para que permita que objetos Java sejam transformados em sequências de bytes, facilitando a sua gravação e leitura em arquivos binários de forma eficiente e consistente.

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

Permite operações de processamento de dados de forma funcional, expressões lambda e pipelines de dados, ou seja, permite escrever códigos mais conciso, legível e eficiente para as operações solicitadas. Tornando assim o código mais declarativo, facilitando o desenvolvimento e manutenção do mesmo.

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

O padrão comum para persistência de dados é o de projeto DAO (Data Access Object), que separa a lógica de acesso a dados da lógica de negócios, proporcionando uma abstração limpa e modular para a manipulação dos dados armazenados.

# 2⁰ Procedimento - Criação do Cadastro em Modo Texto

**Objetivo da prática:**

Nesta etapa o objetivo é criar um sistema simples de cadastro em texto, onde o usuário pode realizar operações como adicionar, alterar, excluir e visualizar dados de entidades (pessoas físicas ou jurídicas). Essas operações serão feitas através de um menu de opções, onde o usuário digitará números para escolher o que deseja fazer. Além disso, o sistema permite salvar e recuperar os dados em arquivos.

# Prática:

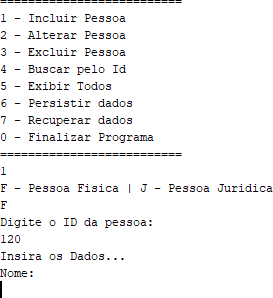
* + Alterar a classe principal (main) para implementação do cadastro em modo texto;
  + Criar e apresentar um meu com opções do programa para o usuário: 1 - Incluir, 2

- Alterar, 3 - Excluir, 4 - Exibir pelo Id, 5 - Exibir Todos, 6 - Salvar Dados, 7 - Recuperar Dados e 0 - Finalizar a Execução;

* + Criar estrutura (código) para as opções do menu;
  + Executar e verificar as funcionalidades implementadas e os arquivos gerados.

# Códigos e resultados obtidos:

* + Resultado inicial (exemplo):



* + Códigos:

Alterar o método main para implementação do cadastro em modo texto:

Classe Main\_02:

package model;

/\*\*

\*

\* @author gilvan

\*/

import java.io.\*;

import java.util.Scanner;

public class Main\_02 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

PessoaFisicaRepo repoPessoaFisica = new PessoaFisicaRepo();

PessoaJuridicaRepo repoPessoaJuridica = new PessoaJuridicaRepo();

boolean continuar = true;

while (continuar) {

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 - 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("==========================");

int opcao = scanner.nextInt();

switch (opcao) {

case 1 -> incluir(scanner, repoPessoaFisica, repoPessoaJuridica);

case 2 -> alterar(scanner, repoPessoaFisica, repoPessoaJuridica);

case 3 -> excluir(scanner, repoPessoaFisica, repoPessoaJuridica);

case 4 -> exibirPorId(scanner, repoPessoaFisica, repoPessoaJuridica);

case 5 -> exibirTodos(scanner, repoPessoaFisica, repoPessoaJuridica);

case 6 -> salvarDados(scanner, repoPessoaFisica, repoPessoaJuridica);

case 7 -> recuperarDados(scanner, repoPessoaFisica, repoPessoaJuridica);

case 0 -> continuar = false;

default -> System.out.println("Opcao invalida. Tente novamente.");

}

}

}

private static void incluir(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String tipo = scanner.next();

System.out.println("Digite o ID da pessoa:");

int id = scanner.nextInt();

System.out.println("Insira os Dados...");

if (tipo.equalsIgnoreCase("F")) {

System.out.println("Nome:");

String nome = scanner.next();

System.out.println("CPF:");

String cpf = scanner.next();

System.out.println("Idade:");

int idade = scanner.nextInt();

PessoaFisica pessoaFisica = new PessoaFisica(id, nome, cpf, idade);

repoPessoaFisica.inserir(pessoaFisica);

pessoaFisica.exibir();

} else if (tipo.equalsIgnoreCase("J")) {

System.out.println("Nome:");

String nome = scanner.next();

System.out.println("CNPJ:");

String cnpj = scanner.next();

PessoaJuridica pessoaJuridica = new PessoaJuridica(id, nome, cnpj);

repoPessoaJuridica.inserir(pessoaJuridica);

pessoaJuridica.exibir();

} else {

System.out.println("Opcao invalida.");

}

}

private static void alterar(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String tipo = scanner.next();

System.out.println("Digite o ID da pessoa:");

int id = scanner.nextInt();

switch (tipo.toUpperCase()) {

case "F" -> {

PessoaFisica pessoaFisica = repoPessoaFisica.obter(id);

if (pessoaFisica != null) {

System.out.println("Dados atuais:");

pessoaFisica.exibir();

scanner.nextLine();

System.out.println("Digite o novo nome:");

String nome = scanner.nextLine();

System.out.println("Digite o novo CPF:");

String cpf = scanner.nextLine();

System.out.println("Digite a nova idade:");

int idade = scanner.nextInt();

pessoaFisica.setNome(nome);

pessoaFisica.setCpf(cpf);

pessoaFisica.setIdade(idade);

repoPessoaFisica.alterar(pessoaFisica);

pessoaFisica.exibir();

} else {

System.out.println("Pessoa fisica nao encontrada.");

}

}

case "J" -> {

PessoaJuridica pessoaJuridica = repoPessoaJuridica.obter(id);

if (pessoaJuridica != null) {

System.out.println("Dados atuais:");

pessoaJuridica.exibir();

scanner.nextLine();

System.out.println("Digite o novo nome:");

String nome = scanner.nextLine();

System.out.println("Digite o novo CNPJ:");

String cnpj = scanner.nextLine();

pessoaJuridica.setNome(nome);

pessoaJuridica.setCnpj(cnpj);

repoPessoaJuridica.alterar(pessoaJuridica);

pessoaJuridica.exibir();

} else {

System.out.println("Pessoa juridica nao encontrada.");

}

}

default -> System.out.println("Opcao invalida.");

}

}

private static void excluir(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String tipo = scanner.next();

System.out.println("Digite o ID da pessoa:");

int id = scanner.nextInt();

switch (tipo.toUpperCase()) {

case "F" -> repoPessoaFisica.excluir(id);

case "J" -> repoPessoaJuridica.excluir(id);

default -> System.out.println("Opcao invalida.");

}

}

private static void exibirPorId(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String tipo = scanner.next();

System.out.println("Digite o ID da pessoa:");

int id = scanner.nextInt();

switch (tipo.toUpperCase()) {

case "F" -> {

PessoaFisica pessoaFisica = repoPessoaFisica.obter(id);

if (pessoaFisica != null) {

pessoaFisica.exibir();

} else {

System.out.println("Pessoa fisica nao encontrada.");

}

}

case "J" -> {

PessoaJuridica pessoaJuridica = repoPessoaJuridica.obter(id);

if (pessoaJuridica != null) {

pessoaJuridica.exibir();

} else {

System.out.println("Pessoa juridica nao encontrada.");

}

}

default -> System.out.println("Opcao invalida.");

}

}

private static void exibirTodos(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String tipo = scanner.next();

switch (tipo.toUpperCase()) {

case "F" -> {

System.out.println("Pessoas Fisicas:");

for (PessoaFisica pessoa : repoPessoaFisica.obterTodos()) {

pessoa.exibir();

}

}

case "J" -> {

System.out.println("Pessoas Juridicas:");

for (PessoaJuridica pessoa : repoPessoaJuridica.obterTodos()) {

pessoa.exibir();

}

}

default -> System.out.println("Opcao invalida.");

}

}

private static void salvarDados(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

try {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String prefixo = scanner.next();

repoPessoaFisica.persistir(prefixo + ".fisica.bin");

repoPessoaJuridica.persistir(prefixo + ".juridica.bin");

System.out.println("Dados salvos com sucesso.");

} catch (IOException e) {

System.out.println("Erro ao salvar os dados: " + e.getMessage());

}

}

private static void recuperarDados(Scanner scanner, PessoaFisicaRepo repoPessoaFisica, PessoaJuridicaRepo repoPessoaJuridica) {

try {

System.out.println("F - Pessoa Fisica | J - Pessoa Juridica");

String prefixo = scanner.next();

repoPessoaFisica.recuperar(prefixo + ".fisica.bin");

repoPessoaJuridica.recuperar(prefixo + ".juridica.bin");

System.out.println("Dados recuperados com sucesso.");

} catch (IOException | ClassNotFoundException e) {

System.out.println("Erro ao recuperar os dados: " + e.getMessage());

}

}

}

# Conclusão:

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

Elementos estáticos são aqueles que pertencem à classe em vez de instâncias individuais da classe. No contexto do método ‘main’, ele é estátco para ser acessado sem criar uma instância da classe, facilitando a execução do

programa.

1. Para que serve a classe Scanner?

É utilizada para obter entrada do usuário a partir do teclado, permitindo ler diferentes tipos de dados de entrada, como inteiros, strings, etc. tornando-se fundamental para interações entre usuário e programa.

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

O uso de classes promove uma organização modular e coesa para o código, separando as responsabilidades de gerenciamentos de tipos específicos de entidades, torando o código mais legível, de fácil manutenção e atualização, seguindo os princípios de encapsulamento e coesão, deixando a reutilização de código mais ágil, pois as operações relacionadas as entidades específicas estão contidas em suas próprias classes de repositório.