**Criação das Entidades e Sistema de Persistência**

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.

Códigos Solicitados:

**Classe Pessoa:**

public class Pessoa implements Serializable{

private int id;

private String nome;

public int getId() {

return id;

}

public String getNome() {

return nome;

}

public void setId(int id) {

this.id = id;

}

public void setNome(String nome) {

this.nome = nome;

}

public void exibir(){

System.out.printf("Id: %d%n",this.id);

System.out.printf("Nome: %s%n",this.nome);

}

public Pessoa(String nome,int id){

this.nome=nome;

this.id=id;

}

}

**Classe Pessoa Física:**

public class PessoaFisica extends Pessoa implements Serializable {

private String cpf;

private int idade;

public String getCpf() {

return cpf;

}

public int getIdade() {

return idade;

}

public void setCpf(String cpf) {

this.cpf = cpf;

}

public void setIdade(int idade) {

this.idade = idade;

}

@Override

public void exibir(){

super.exibir();

System.out.printf("CPF: %s%n",this.cpf);

System.out.printf("Idade: %d%n",this.idade);

}

public PessoaFisica(String nome, int id, String cpf, int idade){

super(nome,id);

this.cpf = cpf;

this.idade = idade;

}

}

**Classe PessoaJuridica:**

public class PessoaJuridica extends Pessoa implements Serializable {

private String cnpj;

public String getCnpj() {

return cnpj;

}

public void setCnpj(String cnpj) {

this.cnpj = cnpj;

}

@Override

public void exibir(){

super.exibir();

System.out.printf("CNPJ:%s%n",this.cnpj);

}

public PessoaJuridica(String nome, int id, String cnpj){

super(nome,id);

this.cnpj=cnpj;

}

}

**Classe Pessoa Jurídica:**

public class PessoaJuridica extends Pessoa implements Serializable {

private String cnpj;

public String getCnpj() {

return cnpj;

}

public void setCnpj(String cnpj) {

this.cnpj = cnpj;

}

@Override

public void exibir(){

super.exibir();

System.out.printf("CNPJ:%s%n",this.cnpj);

}

public PessoaJuridica(String nome, int id, String cnpj){

super(nome,id);

this.cnpj=cnpj;

}

}

**Classe Pessoa Física Repositório:**

public class PessoaFisicaRepo {

private ArrayList<PessoaFisica> pessoasF= new ArrayList<>();

public void inserir(PessoaFisica ...pessoas){

this.pessoasF.addAll(Arrays.asList(pessoas));

}

public void alterar(PessoaFisica pessoa){

int id = pessoa.getId();

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

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

pessoasF.remove(i);

pessoasF.add(i,pessoa);

}

}

}

public void excluir(int id){

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

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

pessoasF.remove(i);

}

}

}

public void obter(int id){

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

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

pessoasF.get(i).exibir();

}

}

}

public void obterTodos(){

this.pessoasF.stream().map(e -> {

e.exibir();

return e;

}).forEachOrdered(\_item -> {

System.out.println("--------------------------");

});

}

public void persistir(String arquivo) throws IOException{

serializarF(pessoasF,arquivo);

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

}

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

pessoasF = deserializarF(arquivo);

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

}

public PessoaFisicaRepo(){

}

}

**Classe Pessoa Jurídica Repositório:**

public class PessoaJuridicaRepo {

private ArrayList<PessoaJuridica> pessoasJ = new ArrayList<>();

public void inserir(PessoaJuridica ...pessoas){

this.pessoasJ.addAll(Arrays.asList(pessoas));

}

public void alterar(PessoaJuridica pessoa){

int id = pessoa.getId();

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

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

pessoasJ.remove(i);

pessoasJ.add(i,pessoa);

}

}

}

public void excluir(int id){

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

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

pessoasJ.remove(i);

}

}

}

public void obter(int id){

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

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

pessoasJ.get(i).exibir();

}

}

}

public void obterTodos(){

this.pessoasJ.stream().map(e -> {

e.exibir();

return e;

}).forEachOrdered(\_item -> {

System.out.println("--------------------------");

});

}

public void persistir(String arquivo) throws IOException{

serializarJ(pessoasJ,arquivo);

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

}

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

pessoasJ = deserializarJ(arquivo);

System.out.println("Dados de Pessoa Juridica Recuperado");

}

public PessoaJuridicaRepo(){

}

}

**Métodos de persistência:**

public class Serializacao {

public static void serializarF(ArrayList pessoas, String arquivo) throws IOException{

Path path = Paths.get(arquivo);

try(ObjectOutputStream oos= new ObjectOutputStream(Files.newOutputStream(path))){

oos.writeObject(pessoas);

}catch(IOException e){

System.out.println(e);

}

}

public static void serializarJ(ArrayList pessoas, String arquivo) throws IOException{

Path path = Paths.get(arquivo);

try(ObjectOutputStream oos= new ObjectOutputStream(Files.newOutputStream(path))){

oos.writeObject(pessoas);

}catch(IOException e){

System.out.println(e);

}

}

public static ArrayList deserializarF(String arquivo) throws IOException, ClassNotFoundException{

Path path = Paths.get(arquivo);

ArrayList<PessoaFisica> pessoasF= new ArrayList<>();

try(ObjectInputStream ois= new ObjectInputStream(Files.newInputStream(path))){

pessoasF= (ArrayList) ois.readObject();

}catch(IOException | ClassNotFoundException e){

System.out.println(e);

}

return pessoasF;

}

public static ArrayList deserializarJ(String arquivo) throws IOException, ClassNotFoundException{

Path path = Paths.get(arquivo);

ArrayList<PessoaJuridica> pessoasJ= new ArrayList<>();

try(ObjectInputStream ois= new ObjectInputStream(Files.newInputStream(path))){

pessoasJ= (ArrayList) ois.readObject();

}catch(IOException | ClassNotFoundException e){

System.out.println(e);

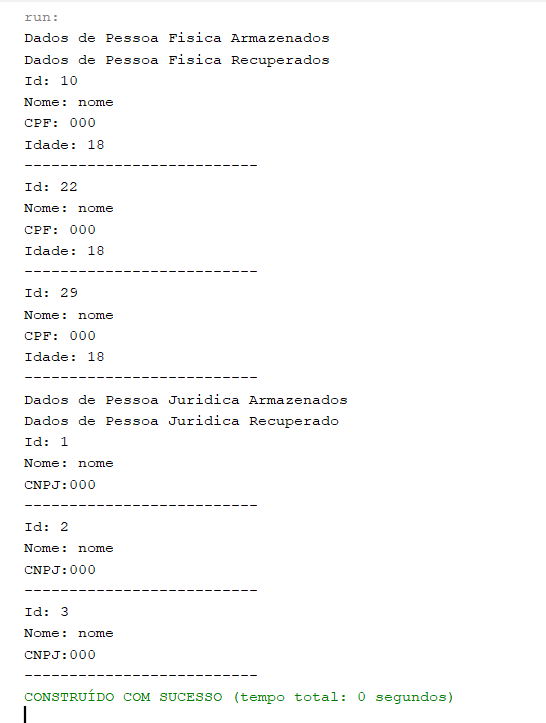
}

return pessoasJ;

}

}

**Resultados da execução:**



**Análise e Conclusão**

* Quais as vantagens e desvantagens do uso de herança?

Com a herança é possível reutilizar a classe “pai” por diversas vezes, oque nos permite reduzir bastante os códigos que seriam repetidos na classe “filho”.

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

A interface serializable serve como marcação para que o computador entenda que aquele código é serializavel.

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

Acrescenta diversas ferramentas para aplicação.

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

O método outputstream, onde pegamos os dados e os direcionamos a um arquivo onde sera guardado em forma de uma cadeia de bytes.