Implementação de um Sistema Cadastral em Java Utilizando Herança, Polimorfismo e Persistência em Arquivos Binários

1º Procedimento | Criação das Entidades e Sistema de Persistência

Aluno: Caio Viana Castelo Branco

Matrícula: 202307465925

Disciplina: RPG0014 - Iniciando o caminho pelo Java

Curso: Desenvolvimento Full Stack

Turma: 9001

Semestre Letivo: 2025.1

Campus: Parangaba

1. Objetivo 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.

2. Códigos utilizados na prática

Classe principal: CadastroPOO.java

```
package cadastropoo;

import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;
import java.io.IOException;

public class CadastroPOO {
   public static void main(String[] args) {
      try {
         // Teste PF
         PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
         repo1.inserir(new PessoaFisica(1, "Ana", "11111111111", 25));
         repo1.inserir(new PessoaFisica(2, "Carlos", "22222222222", 52));
         repo1.persistir("pessoas_fisicas.dat");
         System.out.println("Dados de Pessoa Física Armazenados.");

         PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
         repo2.recuperar("pessoas_fisicas.dat");
         System.out.println("Dados de Pessoa Física Recuperados.");
         for (PessoaFisica pf : repo2.obterTodos()) {
            pf.exibir();
         }

         // Teste PJ
         PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
         repo3.inserir(new PessoaJuridica(3, "XPTO Sales", "33333333333333"));
         repo3.inserir(new PessoaJuridica(4, "XPTO Solutions", "44444444444444"));
         repo3.persistir("pessoas_juridicas.dat");
         System.out.println("Dados de Pessoa Jurídica Armazenados.");
```

```
        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        repo4.recuperar("pessoas_juridicas.dat");
        System.out.println("Dados de Pessoa Jurídica Recuperados.");
        for (PessoaJuridica pj : repo4.obterTodos()) {
            pj.exibir();
        }

    } catch (IOException e) {
        System.out.println("Erro de I/O: " + e.getMessage());
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        System.out.println("Classe não encontrada: " + e.getMessage());
        e.printStackTrace();
    } catch (Exception e) {
        System.out.println("Erro inesperado: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Classes de Modelo:

Pessoa.java

```
package model;

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 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;

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;
    }

    @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; }
}
```

PessoaJuridica.java

```
package model;

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;
   }

   @Override
   public void exibir() {
      super.exibir();
      System.out.println("CNPJ: " + cnpj);
   }

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

Repositórios:

PessoaFisicaRepo.java

```
package model;

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

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

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

   public void alterar(PessoaFisica pessoa) {
      for (int i = 0; i < pessoasFisicas.size(); i++) {
         if (pessoasFisicas.get(i).getId() == pessoa.getId()) {
            pessoasFisicas.set(i, pessoa);
            break;
         }
      }
   }

   public void excluir(int id) {
```

```
    pessoasFisicas.removeIf(p -> p.getId() == id);
  }

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

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

  public void persistir(String arquivo) throws IOException {
    try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(arquivo))) {
      out.writeObject(pessoasFisicas);
    }
  }

  public void recuperar(String arquivo) throws IOException,
ClassNotFoundException {
    try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(arquivo))) {
      pessoasFisicas = (ArrayList<PessoaFisica>) in.readObject();
    }
  }
}
```

PessoaJuridicaRepo.java

```
package model;

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

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

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

```
    public void alterar(PessoaJuridica pessoa) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            if (pessoasJuridicas.get(i).getId() == pessoa.getId()) {
                pessoasJuridicas.set(i, pessoa);
                break;
            }
        }
    }

    public void excluir(int id) {
        pessoasJuridicas.removeIf(p -> p.getId() == id);
    }

    public PessoaJuridica obter(int id) {
        for (PessoaJuridica p : pessoasJuridicas) {
            if (p.getId() == id) {
                return p;
            }
        }
        return null;
    }

    public ArrayList<PessoaJuridica> obterTodos() {
        return pessoasJuridicas;
    }

    public void persistir(String arquivo) throws IOException {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(arquivo))) {
            out.writeObject(pessoasJuridicas);
        }
    }

    public void recuperar(String arquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new
FileInputStream(arquivo))) {
            pessoasJuridicas = (ArrayList<PessoaJuridica>) in.readObject();
        }
    }
}
```

3. Resultados da execução dos códigos

```
run:
Dados de Pessoa F�sica Armazenados.
Dados de Pessoa F�sica Recuperados.
Id: 1
Nome: Ana
CPF: 11111111111
Idade: 25
Id: 2
Nome: Carlos
CPF: 22222222222
Idade: 52
Dados de Pessoa Jur�dica Armazenados.
Dados de Pessoa Jur�dica Recuperados.
Id: 3
Nome: XPTO Sales
CNPJ: 33333333333333
Id: 4
Nome: XPTO Solutions
CNPJ: 44444444444444
BUILD SUCCESSFUL (total time: 0 seconds)
```

Arquivos gerados:
pessoas_fisicas.dat
pessoas_juridicas.dat

4. Análise e Conclusão

- Quais as vantagens e desvantagens do uso de herança?

  A herança permitiu reutilizar código entre as classes PessoaFisica e PessoaJuridica, evitando duplicação, mas também mostrou que pode aumentar o acoplamento entre classes. O polimorfismo foi essencial para sobrescrever o método exibir(), adaptando-o a cada tipo de pessoa.

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

  Serializable foi necessária para transformar objetos em bytes e salvá-los em arquivos, garantindo que os dados permanecessem consistentes entre

execuções do programa. Sem ela, a serialização não funcionaria, lançando uma exceção.

- Como o paradigma funcional é utilizado pela API stream no Java?
  O paradigma funcional apareceu no uso da Stream API, como no método removeIf(), que simplificou a exclusão de itens com uma expressão lambda, tornando o código mais limpo.

- Quando trabalhamos com Java, qual padrão de desenvolvimento é adotado na persistência de dados em arquivos?
  O padrão de persistência adotado foi a serialização binária, usando ObjectOutputStream e ObjectInputStream. Essa abordagem é simples e eficaz para armazenar objetos, mas em sistemas mais complexos, outras opções como JSON ou bancos de dados podem ser mais flexíveis.