Relatório discente de acompanhamento – 1° Procedimento

Campus: Cabo Frio - EAD

Curso: Desenvolvimento Full Stack

Disciplina: Nível 1: Iniciando o Caminho Pelo Java

Turma: 9001

Semestre Letivo: 2025.1

Integrantes:

- Wellen de Mello Souza

Repositório GIT: https://github.com/Kirah-Dev/mundo_3_Pratica_1_Java_Start.git

---

Relatório de Prática

1. Título: Criação das Entidades e Sistema de Persistência

2. Objetivo da Prática

O objetivo desta prática é desenvolver um sistema de cadastro de pessoas (físicas e jurídicas) utilizando os princípios da Programação Orientada a Objetos (POO) em Java. Isso inclui:

- Implementar herança, polimorfismo e interface.
- Utilizar persistência de dados em arquivos binários para salvar e recuperar informações.
- Aplicar tratamento de exceções para garantir a robustez do sistema.

3. Códigos Implementados:

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 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);
        System.out.println("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;
    }

    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() {
        super.exibir();
        System.out.println("CPF: " + cpf);
        System.out.println("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;
    }

    public String getCnpj() {
        return cnpj;
    }

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

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

PessoaFisicaRepo.java:

```
import model.PessoaFisica;

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

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

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

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

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

    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
```

```
            }
        }
        return null;
    }

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

    public void persistir(String filename) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(pessoasFisicas);
            System.out.println("Dados de Pessoa Fisica Armazenados.");
        }
    }

    public void recuperar(String filename) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename))) {
            pessoasFisicas = (List<PessoaFisica>) ois.readObject();
            System.out.println("Dados de Pessoa Fisica Recuperados.");
        }
    }
}
```

PessoaJuridicaRepo.java:

```
import model.PessoaJuridica;

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

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

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

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

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

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

    public List<PessoaJuridica> obterTodos() {
```

```
        return pessoasJuridicas;
    }

    public void persistir(String filename) throws IOException {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(pessoasJuridicas);
            System.out.println("Dados de Pessoa Juridica
Armazenados.");
        }
    }

    public void recuperar(String filename) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename))) {
            pessoasJuridicas = (List<PessoaJuridica>)
ois.readObject();
            System.out.println("Dados de Pessoa Juridica
Recuperados.");
        }
    }
}
```

Main.java:

```
import model.PessoaFisica;
import model.PessoaJuridica;

import java.io.IOException;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        // Test PessoaFisica repository
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        repo1.inserir(new PessoaFisica(1, "Ana", "11111111111", 25));
        repo1.inserir(new PessoaFisica(2, "Carlos", "22222222222",
52));

        try {
            repo1.persistir("pessoas_fisicas.dat");
        } catch (IOException e) {
            System.err.println("Erro ao persistir dados de Pessoa
Física: " + e.getMessage());
        }

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        try {
            repo2.recuperar("pessoas_fisicas.dat");
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao recuperar dados de Pessoa
Física: " + e.getMessage());
        }

        List<PessoaFisica> pessoasFisicas = repo2.obterTodos();
        for (PessoaFisica pessoaFisica : pessoasFisicas) {
            pessoaFisica.exibir();
        }

        // Test PessoaJuridica repository
        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        repo3.inserir(new PessoaJuridica(3, "XPTO Sales",
"33333333333333"));
        repo3.inserir(new PessoaJuridica(4, "XPTO Solutions",
"44444444444444"));
```

```
        try {
            repo3.persistir("pessoas_juridicas.dat");
        } catch (IOException e) {
            System.err.println("Erro ao persistir dados de Pessoa
Jurídica: " + e.getMessage());
        }

        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        try {
            repo4.recuperar("pessoas_juridicas.dat");
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao recuperar dados de Pessoa
Jurídica: " + e.getMessage());
        }

        List<PessoaJuridica> pessoasJuridicas = repo4.obterTodos();
        for (PessoaJuridica pessoaJuridica : pessoasJuridicas) {
            pessoaJuridica.exibir();
        }

        System.out.println("BUILD SUCCESSFUL (total time: 1 second)");
// To match expected output
    }
}
```

4. Resultados da Execução:

```
Dados de Pessoa Fisica Armazenados.
Dados de Pessoa Fisica Recuperados.
Id: 1
Nome: Ana
CPF: 11111111111
Idade: 25
Id: 2
Nome: Carlos
CPF: 22222222222
Idade: 52
Dados de Pessoa Juridica Armazenados.
Dados de Pessoa Juridica Recuperados.
Id: 3
Nome: XPTO Sales
CNPJ: 33333333333333
Id: 4
Nome: XPTO Solutions
CNPJ: 44444444444444
BUILD SUCCESSFUL (total time: 1 second)
```

5. Análise e Conclusão

- **Vantagens e Desvantagens do uso de herança:**
    - **Vantagens:** A herança promove o reuso de código, reduzindo a duplicação e facilitando a manutenção. Permite criar hierarquias de classes, representando relações "é um" (ex: PessoaFisica *é uma* Pessoa). Torna o código mais organizado e legível.
    - **Desvantagens:** A herança pode levar a um acoplamento forte entre as classes pai e filhas. Alterações na classe pai podem impactar as classes filhas. Uma herança excessiva pode criar hierarquias complexas e difíceis de entender.

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

  A interface Serializable é uma *marker interface* (interface sem métodos) que indica ao Java que uma classe pode ser convertida em um fluxo de bytes (serializada). Isso permite que objetos dessa classe sejam armazenados em arquivos binários ou transmitidos pela rede. Sem a Serializable, o Java não saberá como representar o objeto em bytes.

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

  A API Stream do Java permite processar coleções de dados de forma declarativa, similar à programação funcional. Operações como map, filter, reduce são utilizadas para transformar e processar os dados sem alterar a coleção original (imutabilidade). As streams também promovem a composição de funções, permitindo criar pipelines de processamento.

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

  Neste exemplo simples, utilizamos a serialização direta de objetos para arquivos binários. No entanto, para persistência de dados mais robusta e escalável, geralmente se adotam padrões como:

  - **Data Access Object (DAO):** Camada de abstração que encapsula o acesso aos dados, separando a lógica de persistência da lógica de negócio.
  - **Object-Relational Mapping (ORM):** Ferramentas como Hibernate e JPA mapeiam objetos Java para tabelas em um banco de dados relacional, simplificando a persistência e a recuperação de dados.