UNIVERSIDADE ESTÁCIO DE SÁ

Relatório de Atividade Prática

IDELMAR FERNANDO DE SOUZA

SANTA ROSA
2024

## 1. Atividade Prática – ProjetoPOO

Objetivo da Prática

Desenvolver um pequeno projeto com orientação a objeto com a linguagem Java, para inserir cadastro de pessoa física e pessoa jurídica, com métodos de inserir, alterar, excluir e recuperar os dados existentes;

## 2. Códigos utilizados

## Classe CadastroPOO:

```
public class CadastroPOO {

   public static void main(String[] args) {

      //PESSOA FISICA
      PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
      PessoaFisica pessoa1 = new PessoaFisica(1, "Marcos", "25896314700", 21);
      PessoaFisica pessoa2 = new PessoaFisica(2, "Gustavo", "75395147050", 34);

      repo1.inserir(pessoa1);
      repo1.inserir(pessoa2);

      String pessoas_fisicas = "pessoas_fisicas.dat";
      try {
         repo1.persistir(pessoas_fisicas);
         System.out.println("Dados de Pessoa Fisica Armazenados!" + pessoas_fisicas);
      } catch (IOException e) {
         System.out.println("Erro ao armazenar os dados: " + e.getMessage());
      }

      PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

      try {
         repo2.recuperar(pessoas_fisicas);
         System.out.println("Dados de Pessoa Fisica Recuperados!" + pessoas_fisicas);
      } catch (IOException | ClassNotFoundException e) {
         System.out.println("Erro ao recuperar os dados: " + e.getMessage());
      }

      List<PessoaFisica> cadastrosFis = repo2.obterTodos();
      for (PessoaFisica pessoaFis : cadastrosFis) {
         System.out.println("ID: " + pessoaFis.getId());
         System.out.println("Nome: " + pessoaFis.getNome());
         System.out.println("CPF: " + pessoaFis.getCpf());
         System.out.println("Idade: " + pessoaFis.getIdade());
         System.out.println();
      }

      //PESSOA JURIDICA
      PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
      PessoaJuridica pessoajuridica1 = new PessoaJuridica(1, "Lavanderia Super",
"258963147000188");
      PessoaJuridica pessoajuridica2 = new PessoaJuridica(2, "Padaria PaoBao",
"7539514705000155");

      repo3.inserir(pessoajuridica1);
      repo3.inserir(pessoajuridica2);

      String pessoa_juridica = "pessoa_juridica.dat";
      try {
         repo3.persistir(pessoa_juridica);
         System.out.println("Dados de Pessoa Juridica Armazenados!" + pessoa_juridica);
      } catch (IOException e) {
         System.out.println("Erro ao armazenar os dados: " + e.getMessage());
      }

      PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
```

```
        try {
            repo4.recuperar(pessoa_juridica);
            System.out.println("Dados de Pessoa Juridica Recuperados!" + pessoa_juridica);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar os dados: " + e.getMessage());
        }

        List<PessoaJuridica> cadastrosJur = repo4.obterTodos();
        for (PessoaJuridica pessoaJur : cadastrosJur) {
            System.out.println("ID: " + pessoaJur.getId());
            System.out.println("Nome: " + pessoaJur.getNome());
            System.out.println("CNPJ: " + pessoaJur.getCnpj());
            System.out.println();
        }
    }
}
```

Resultado da execução:
Define repo1 sendo um novo repositório de pessoa física, para receber os objetos passados no caso pessoa1 e pessoa2; define um arquivo pessoas_fisicas.dat para armazenar os dados;
Define repo2 para recuperar uma lista dos dados armazenados.
As definições de repo3 e repo4 fazem o mesmo papel que repo1 e repo2, porém suas funções remetem as pessoas jurídicas;

**Classe Pessoa:**

```
public class Pessoa implements Serializable{
    private int id;
    private String nome;

    //construtor
    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 String getNome() {
        return nome;
    }
}
```

Define o objeto principal com os seus atributos primários ( id, nome);
Define também getID e getNome para retornar id e nome;

## Classe PessoaFisica:

```
public class PessoaFisica extends Pessoa {
    private String cpf;
    private int idade;

    public PessoaFisica() {
        super(0, "");
    }

    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 int getIdade() {
        return idade;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    @Override
    public void exibir() {
        System.out.println("CPF: " + cpf);
        System.out.println("idade: " + idade);
    }
}
```

Define o primeiro objeto herdado de Pessoa, adiciona os atributos cpf e idade e também os métodos getCpf,getIdade e setCpf;

## Repositório Pessoa Física:

```
public class PessoaFisicaRepo {

    private List<PessoaFisica> listaPessoas;

    public PessoaFisicaRepo() {
        listaPessoas = new ArrayList<>();
    }

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

    public void alterar(int id, PessoaFisica pessoaAtualizada) {
```

```
        for (int i = 0; i < listaPessoas.size(); i++) {
            if (listaPessoas.get(i).getId() == id) {
                listaPessoas.set(i, pessoaAtualizada);
                break;
            }
        }
    }

    public void excluir(int id) {
        Iterator<PessoaFisica> iterator = listaPessoas.iterator();
        while (iterator.hasNext()) {
            PessoaFisica pessoa = iterator.next();
            if (pessoa.getId() == id) {
                iterator.remove();
                break;
            }
        }
    }

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

    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(listaPessoas);
    }

    public void persistir(String arquivo) throws IOException {
        try (FileOutputStream fileOut = new FileOutputStream(arquivo);
                ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {
            objectOut.writeObject(listaPessoas);

        } catch (IOException e) {
            throw e;
        }
    }

    public void recuperar(String arquivo) throws IOException, ClassNotFoundException {
        try (FileInputStream fileIn = new FileInputStream(arquivo);
                ObjectInputStream objectIn = new ObjectInputStream(fileIn)) {
            listaPessoas = (List<PessoaFisica>) objectIn.readObject();

        } catch (IOException | ClassNotFoundException e) {
            throw e;
        }
    }
}
```

Adiciona um vetor com os dados da pessoa física, para manipulação dos dados possui os métodos inserir, alterar, excluir, obter e obter todos e os métodos persistir e recuperar para gravar e retornar os dados;

## Classe Pessoa Jurídica:

```
public class PessoaJuridica extends Pessoa {

   private String cnpj;

   public PessoaJuridica() {
      super(0, "");
   }

   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() {
      System.out.println("Cnpj: " + cnpj);
   }
}
```

Segunda herança da classe principal, adiciona atributo cnpj, e métodos getCnpj,setCnpj e exibir (cnpj);

## Repositório Pessoa Jurídica:

```
public class PessoaJuridicaRepo {

   private List<PessoaJuridica> listaPessoaJuridica;

   public PessoaJuridicaRepo() {
      listaPessoaJuridica = new ArrayList<>();
   }

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

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

```
    }

    public void excluir(int id) {
        Iterator<PessoaJuridica> iterator = listaPessoaJuridica.iterator();
        while (iterator.hasNext()) {
            PessoaJuridica pessoa = iterator.next();
            if (pessoa.getId() == id) {
                iterator.remove();
                break;
            }
        }
    }

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

    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(listaPessoaJuridica);
    }

    public void persistir(String arquivo) throws IOException {
        try (FileOutputStream fileOut = new FileOutputStream(arquivo);
                ObjectOutputStream objectOut = new ObjectOutputStream(fileOut)) {
            objectOut.writeObject(listaPessoaJuridica);

        } catch (IOException e) {
            throw e;
        }
    }

    public void recuperar(String arquivo) throws IOException, ClassNotFoundException {
        try (FileInputStream fileIn = new FileInputStream(arquivo);
                ObjectInputStream objectIn = new ObjectInputStream(fileIn)) {
            listaPessoaJuridica = (List<PessoaJuridica>) objectIn.readObject();

        } catch (IOException | ClassNotFoundException e) {
            throw e;
        }
    }
}
```

Adiciona um vetor com os dados da pessoa jurídica, possui os métodos inserir, alterar, excluir, obter e obter todos e os métodos persistir e recuperar para gravar e retornar os dados;

## 3. Análise e Conclusão:

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

As vantagens do uso de herança é o reaproveitamento de código, hierarquia de classes, melhor visibilidade de estrutura facilitando a manutenção, torna de fácil entendimento a criação de novos objetos herdados e ao mesmo tempo protegendo o código principal do projeto, caso aconteça alguma incoerência fica limitada ao filho;

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

A interface é necessária para fazer a conversão dos dados em bytes para que posso ser armazenados;
Como o paradigma funcional é utilizado pela API stream no Java utilizando o steam o código fica mais enxuto melhorando a legibilidade e manutenção.

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

A API Steam simplifica a escrita do codigo dimiinuindo linhas, melhorando a legibilidade e facilitando a manutenção.

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

O padrão mais adotado é o DAO (Data Acess object) usado juntamento com a serialização é usado para encapsular a persistencia de dados.