RELATÓRIO DE PRÁTICA

UNIVERSIDADE ESTÁCIO DE SÁ

RELATÓRIO DE PRÁTICA DE DESENVOLVIMENTO FULL STACK

Recife 2024/2

EROS SANTOS DE VASCONCELOS

RELATÓRIO DE PRÁTICA DE DESEVOLVIMENTO FULL STACK

Este trabalho é pré-requisito para aprovação na disciplina Iniciando o Caminho Pelo Java, Turma 9001

(modalidade EAD), da instituição de Ensino Sá,

POLO IPUTINGA - RECIFE - PE 2024

OBJETIVO DA PRÁTICA:

O objetivo deste projeto é implementar um sistema de registro de clientes utilizando herança orientada a objetos, polimorfismo e uma interface Serializable para persistência de dados em binários. Também queremos implementar controles especiais e criar modos de registro e texto na plataforma Java.

DESENVOLVIMENTO PASSO A PASSO

1. Criação e Empacotamento

Iniciamos o exercício criando um projeto Java do tipo Ant Java Application denominado cadastroPOO no NetBeans IDE. Neste projeto criamos um pacote chamado model no qual definimos os principais objetos que representam recursos e seu armazenamento.

2. Definição dos componentes

O próximo passo é criar as classes Pessoa, PessoaFisica e PessoaJuridica, com as seguintes características:

Pessoa: Esta é a classe base do nosso sistema e representa todos. Os campos são:

id (inteiro): Identificador exclusivo do usuário. Capítulo Nome

(string): nome da pessoa.

Implementa o método display(), responsável por exibir dados individuais no controlador, além dos construtores (um padrão e um completo) e métodos getter na tabela.

Classe PessoaFisica: Esta classe é herdada de Pessoa e representa pessoa física, incluindo os campos:

cpf (string): CPF da pessoa física. Idade

(int): A idade desta pessoa.

Um método show() também foi escrito para exibir mais informações sobre a pessoa.

Classe PessoaJuridica: Também herdada de Pessoa, representa a pessoa jurídica, campos acrescidos:

cnpj (string): CNPJ da pessoa jurídica.

O método display() aqui foi modificado para incluir CNPJ.

Todas as classes implementam Serialized, que permite que objetos sejam salvos em arquivos binários

3. Criar um repositório

Depois dos diferentes grupos, criamos um repositório para gerir estas empresas. São criados dois repositórios:

PessoaFisicaRepo: Esta classe gerencia uma lista de objetos do tipo PessoaFisica. Ele usa um ArrayList privado e possui métodos:

insert(): Adiciona nova pessoa. Capítulo

Variante (): Modifique a variante existente.

delete(): Exclui uma pessoa física por id. find():

Encontre uma pessoa por ID. getAll(): Retorna

todos os usuários cadastrados.

continue(): Armazena dados da lista em arquivo binário.

return(): Lê dados de um arquivo binário e os retorna para uma lista.

PessoaJuridicaRepo: Funciona da mesma forma que um repositório único, mas também utiliza o tipo de objeto PessoaJuridica.

Ambos os buffers usam try-catch para tratar exceções ao ler ou gravar arquivos, como IOException ou ClassNotFoundException.

4. Teste na Main()

Após criar o recurso no repositório, testamos o método na Main(). O processo é o seguinte:

Cadastro:

Criamos um repositório (repo1) e adicionamos duas pessoas. Capítulo

Continuamos processando os dados do arquivo pessoasFisicas.dat.

Recuperação de Pessoas:

Criamos um novo repositório (repo2) e extraímos os dados de pessoasFisicas.dat.

Exibimos o objeto retornado no controle.

Nome Registrado:

Criamos um repositório (repo3) e adicionamos duas empresas. Capítulo

Continuamos processando os dados do arquivo pessoasJuridicas.dat.

Escritório Jurídico de Recuperação:

Criamos um novo repositório (repo4) e extraímos os dados de pessoasJuridicas.dat.

Mostramos que a empresa recuperou o controle.

Resultados

Utilizamos com sucesso um sistema para organizar pessoas físicas e jurídicas em binários. exibe informações do controlador, garantindo que os dados sejam armazenados e lidos corretamente.

Análise

1. **Vantagens e Desvantagens do Uso de Herança**

Vantagens: A herança permite a reutilização de código entre as classes PessoFisica e PessoaJuridica. Isso reduz a complexidade do código.

Desvantagens: A herança aumentará a complexidade do sistema, principalmente quando a classe principal crescer, dificultando sua manutenção.

2. **Por que as interfaces seriais são importantes?**

A interface Serializable é importante porque permite converter objetos Java em uma sequência de bytes para que você possa armazenar facilmente esses objetos em um arquivo binário. Sem ele, Java não pode processar dados e manter o estado dos objetos no disco.

3. Como usar paradigma de serviço e API de streaming em Java?

A API Java Streaming permite que funções como filtragem, mapeamento e redução sejam aplicadas ao processamento de dados. Segue um modelo funcional, fornece mais informações e apoia a gestão eficiente das coleções.

4. Métodos de desenvolvimento de persistência de arquivos

O método aceito para persistência de dados em arquivos é usar o sistema como uma maneira simples de armazenar e recuperar objetos complexos em arquivos. Esse arranjo é comum em pequenos projetos e sistemas locais onde a persistência do banco de dados não é necessária.

Conclusão

Neste exercício podem ser implementados os conceitos de manipulação de objetos, estabilidade de dados e gerenciamento de exceções em Java, fortalecendo assim o conhecimento e a disciplina adquiridos. O sistema desenvolvido é um exemplo prático e útil de manutenção de registros de clientes em arquivos binários.

CÓDIGOS UTILIZADOS

MAIN:

```
package CadastrPOO;
```

```
import model.PessoaFisica; import
model.PessoaFisicaRepo; import
model.PessoaJuridica; import
model.PessoaJuridicaRepo;

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


public class Main {     public static void
main(String[] args) {

      PessoaFisicaRepo repo1 = new PessoaFisicaRepo();


      repo1.inserir(new PessoaFisica(1, "João Silva", "123.456.789-
00", 30));
      repo1.inserir(new PessoaFisica(2, "Maria Souza",
"987.654.321-00", 25));


try {
         repo1.persistir("pessoasFisicas.dat");
      } catch (IOException e) {
         System.out.println("Erro ao persistir dados: " +
e.getMessage());
```

```
        }

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        try {
            repo2.recuperar("pessoasFisicas.dat");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar dados: " +
e.getMessage());
        }

        List<PessoaFisica> pessoasFisicas = repo2.obterTodos();
System.out.println("Pessoas Físicas recuperadas:");        for
(PessoaFisica pf : pessoasFisicas) {
            pf.exibir();
        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        repo3.inserir(new       PessoaJuridica(1,       "Empresa       X",
"12.345.678/0001-99"));
        repo3.inserir(new       PessoaJuridica(2,      "Empresa Y",
"98.765.432/0001-88"));

try {
```

```
            repo3.persistir("pessoasJuridicas.dat");
        } catch (IOException e) {
            System.out.println("Erro ao persistir dados: " +
e.getMessage());
        }

        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        try {
            repo4.recuperar("pessoasJuridicas.dat");
        } catch (IOException | ClassNotFoundException e) {
System.out.println("Erro ao recuperar dados: " + e.getMessage());
        }

        List<PessoaJuridica> pessoasJuridicas = repo4.obterTodos();
System.out.println("Pessoas Jurídicas recuperadas:");          for
(PessoaJuridica pj : pessoasJuridicas) {
            pj.exibir();
        }
    }
}
```

PESSOA:

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

}
PESSOAFISICA: package
model;

public class PessoaFisica extends Pessoa {
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 + ", Idade: " + idade);
    }
}


    }
```

PESSOAJURIDICA:

```
 package model;


public class PessoaJuridica extends Pessoa {
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:

```
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(int id, PessoaFisica novaPessoa) {
PessoaFisica pessoa = obter(id);        if (pessoa != null)
{            pessoa.setNome(novaPessoa.getNome());
pessoa.setCpf(novaPessoa.getCpf());
pessoa.setIdade(novaPessoa.getIdade());
      }
   }

   public void excluir(int id) {
PessoaFisica pessoa = obter(id);        if
(pessoa != null) {
pessoasFisicas.remove(pessoa);
      }
   }
```

```
    public PessoaFisica obter(int id) {
        return pessoasFisicas.stream().filter(p -> p.getId()
== id).findFirst().orElse(null);
    }

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

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

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

```
}
```

PESSOAJURIDICAREPO:

```
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(int id, PessoaJuridica novaPessoa) {
PessoaJuridica pessoa = obter(id);        if (pessoa != null)
{           pessoa.setNome(novaPessoa.getNome());
pessoa.setCnpj(novaPessoa.getCnpj());
      }
   }


   public void excluir(int id) {
```

```
        PessoaJuridica pessoa = obter(id);
if (pessoa != null) {
pessoasJuridicas.remove(pessoa);
        }
    }

    public PessoaJuridica obter(int id) {
        return pessoasJuridicas.stream().filter(p      -> p.getId() ==
id).findFirst().orElse(null);
    }

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

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

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

```
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(arquivo))) {
            pessoasJuridicas = (ArrayList<PessoaJuridica>)
ois.readObject();
        }
    }
}
```

RESULTADO DA EXECUÇÃO DO CÓDIGO: