Campus: Polo Austin - Nova Iguaçu - RJ
Curso: Desenvolvimento Full Stack
Disciplina: Iniciando o Caminho pelo Java - **Turma:** 9001
Semestre letivo: 2024.2
Aluna: Vanessa Santana P de Souza

Título:
Criação das Entidades e Sistema de Persistência

Objetivo da prática:
O objetivo desta prática é implementar um sistema de gerenciamento de pessoas utilizando os conceitos de herança, interfaces e persistência de dados em Java. Os dados das pessoas serão armazenados e recuperados de arquivos binários, e as vantagens e desvantagens do uso de herança serão discutidas..

Códigos Solicitados

Classe Pessoa

```
import java.io.Serializable;

public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;

    private int id;
    private String nome;

    // Construtor padrão
    public Pessoa() {
    }

    // Construtor completo
    public Pessoa(int id, String nome) {
        this.id = id;
        this.nome = nome;
    }

    // Getters e Setters
    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;
    }

    // Método exibir para impressão dos dados
    public void exibir() {
        System.out.println("ID: " + id + ", Nome: " + nome);
    }

    @Override
    public String toString() {
        return "Pessoa{" +
                "id=" + id +
                ", nome='" + nome + '\'' +
                '}';
    }
}
```

Pessoa Física

```
package model;
import java.io.Serializable;

public class PessoaFisica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;

    private String cpf;
    private int idade;


    // Construtor padrão
    public PessoaFisica() {
        super();
    }

    // Construtor completo
```

```
    public PessoaFisica(int id, String nome, String cpf, int idade) {
        super(id, nome); // Chama o construtor da classe base
        this.cpf = cpf;
        this.idade = idade;
    }

    // Getters e Setters
    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;
    }
```

Pessoa Jurídica

```
package model;


import java.io.Serializable;

public class PessoaJuridica extends Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;

    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
        super();
    }

    // Construtor completo
    public PessoaJuridica(int id, String nome, String cnpj) {
        super(id, nome); // Chama o construtor da classe base
        this.cnpj = cnpj;
    }

    // Getters e Setters
```

```
    public String getCnpj() {
        return cnpj;
    }

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

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

    @Override
    public String toString() {
        return "PessoaJuridica{" +
                "cnpj='" + cnpj + '\'' +
                '}';
    }
}
```

Pessoa Física Repo

```
package model;

import java.io.*;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class PessoaFisicaRepo implements Serializable {
    private static final long serialVersionUID = 1L;


    // ArrayList de PessoaFisica
    private List<PessoaFisica> listaPessoas;

    // Construtor
    public PessoaFisicaRepo() {
        this.listaPessoas = new ArrayList<>();
    }
```

```
    // Método para inserir uma nova PessoaFisica
    public void inserir(PessoaFisica pessoa) {
        listaPessoas.add(pessoa);
    }

    // Método para alterar uma PessoaFisica existente
    public void alterar(PessoaFisica pessoa) throws IllegalArgumentException {
        if (!listaPessoas.contains(pessoa)) {
            throw new IllegalArgumentException("Pessoa não encontrada para alteração.");
        }

        int index = listaPessoas.indexOf(pessoa);
        listaPessoas.set(index, pessoa);
    }

    // Método para excluir uma PessoaFisica pelo ID
    public void excluir(int id) {
        listaPessoas.removeIf(p -> p.getId() == id);
    }

    // Método para obter uma PessoaFisica pelo ID
    public PessoaFisica obter(int id) {
        return listaPessoas.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    // Método para obter todas as PessoasFisicas
    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(listaPessoas);
    }
    // Método para persistir os dados no disco
    public void persistir(String nomeArquivo) {
        try (FileOutputStream fileOut = new FileOutputStream(nomeArquivo);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(listaPessoas);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (FileInputStream fileIn = new FileInputStream(nomeArquivo);
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            listaPessoas = (ArrayList<PessoaFisica>) in.readObject();
```

```
    }
  }



}
```

Pessoa Jurídica Repo

```
package model;

import java.io.*;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

public class PessoaJuridicaRepo implements Serializable {
   private static final long serialVersionUID = 1L;
   private List<PessoaJuridica> listaPessoas;

   // Construtor
   public PessoaJuridicaRepo() {
      this.listaPessoas = new ArrayList<>();
   }

   // Método para inserir uma nova PessoaJuridica
   public void inserir(PessoaJuridica pessoa) {
      listaPessoas.add(pessoa);
   }

   // Método para alterar uma PessoaJuridica existente
   public boolean alterar(PessoaJuridica pessoa) {
      Optional<PessoaJuridica> pessoaExistente = listaPessoas.stream()
            .filter(p -> p.getId() == pessoa.getId())
            .findFirst();

      if (pessoaExistente.isPresent()) {
         int index = listaPessoas.indexOf(pessoaExistente.get());
         listaPessoas.set(index, pessoa);
         return true;
      }

      return false;
   }
```

```
    // Método para excluir uma PessoaJuridica pelo ID
    public void excluir(int id) {
        listaPessoas.removeIf(p -> p.getId() == id);
    }

    // Método para obter uma PessoaJuridica pelo ID
    public PessoaJuridica obter(int id) {
        return listaPessoas.stream()
                .filter(p -> p.getId() == id)
                .findFirst()
                .orElse(null);
    }

    // Método para obter todas as PessoasJuridicas
    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(listaPessoas);
    }
    // Método para persistir os dados no disco
    public void persistir(String nomeArquivo) {
        try (FileOutputStream fileOut = new FileOutputStream(nomeArquivo);
             ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            out.writeObject(listaPessoas);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Método para recuperar os dados do disco
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) {
        try (FileInputStream fileIn = new FileInputStream(nomeArquivo);
             ObjectInputStream in = new ObjectInputStream(fileIn)) {
            listaPessoas = (ArrayList<PessoaJuridica>) in.readObject();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Main

```
package model;

import model.PessoaFisica;
import model.PessoaFisicaRepo;

import java.io.IOException;
```

```java
import java.util.List;


public class Main {
    public static void main(String[] args) throws IOException, ClassNotFoundException {

        // Instanciar o repositório de pessoas físicas
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        // Adicionar duas pessoas físicas
        PessoaFisica pessoa1 = new PessoaFisica(1, "João", "123.456.789-10", 30);
        PessoaFisica pessoa2 = new PessoaFisica(2, "Maria", "987.654.321-00", 25);
        repo1.inserir(pessoa1);
        repo1.inserir(pessoa2);

        // Invocar o método de persistência em repo1
        repo1.persistir("pessoas_fisicas.dat");

        // Invocar o método de recuperação em repo2
        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
        repo2.recuperar("pessoas_fisicas.dat");

        System.out.println();
        System.out.println("------------------------------");
        // Exibir os dados de todas as pessoas físicas recuperadas
        List<PessoaFisica> pessoasRecuperadas = repo2.obterTodos();
        System.out.println("Dados de pessoas físicas armazenados.");
        System.out.println("Dados de pessoas físicas Recuperados.");
        System.out.println();
        for (PessoaFisica pessoa : pessoasRecuperadas) {
            System.out.println("Id: " + pessoa.getId());
            System.out.println("nome: " + pessoa.getNome());
            System.out.println("CPF: " + pessoa.getCpf());
            System.out.println("Idade: " + pessoa.getIdade());
        }

        // Instanciar o repositório de pessoas jurídicas repo3
        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        // Adicionar duas pessoas jurídicas
        PessoaJuridica empresa1 = new PessoaJuridica(1, "Empresa Luwa Tech",
"12345678901234");
        PessoaJuridica empresa2 = new PessoaJuridica(2, "Empresa Apple",
"98765432109876");

        repo3.inserir(empresa1);
        repo3.inserir(empresa2);
```

```
        // Invocar o método de persistência em repo3
        repo3.persistir("pessoas.dat");

        // Instanciar outro repositório de pessoas jurídicas repo4
        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
        repo4.recuperar("pessoas.dat");

        // Exibir os dados de todas as pessoas jurídicas recuperadas
        List<PessoaJuridica> pessoasJuridicasRecuperadas = repo3.obterTodos();

        System.out.println();
        System.out.println("------------------------------");
        System.out.println("Dados de pessoas jurídicas armazenados.");
        System.out.println("Dados de pessoas jurídicas Recuperados.");
        System.out.println();
        for (PessoaJuridica pessoaJuridica : pessoasJuridicasRecuperadas) {
            System.out.println("Nome:" + pessoaJuridica.getNome());
            System.out.println("CNPJ:" + pessoaJuridica.getCnpj());
        }
        System.out.println();
    }
}
```

Resultado da execução dos códigos.

C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Educational Edition 2022.2.2\lib\idea_rt.jar=59430:C:\Program Files\JetBrains\IntelliJ IDEA Educational Edition 2022.2.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath "C:\Programas Vanessa Estudo\CadastroPOO1\out\production\CadastroPOO1" model.Main

---
Dados de pessoas físicas armazenados.
Dados de pessoas físicas Recuperados.

Id: 1
nome: João
CPF: 123.456.789-10
Idade: 30
Id: 2
nome: Maria
CPF: 987.654.321-00
Idade: 25

Dados de pessoas jurídicas armazenados.
Dados de pessoas jurídicas Recuperados.

Nome:Empresa Luwa Tech
CNPJ:12345678901234
Nome:Empresa Apple
CNPJ:98765432109876

Process finished with exit code 0

Análise e conclusão:

Quais as vantagens e desvantagens do uso de herança?

Vantagens:

Reutilização de Código:

Reduz duplicação de código ao compartilhar métodos e propriedades comuns entre classes.
Economiza tempo ao evitar reescrita de código.

Facilidade de Manutenção:

Alterações na classe base são automaticamente refletidas nas subclasses.
Mantém a consistência das funcionalidades compartilhadas.

Extensibilidade:

Adere ao princípio aberto/ fechado, permitindo extensões sem modificar código existente.
Permite especialização de funcionalidades em subclasses.

Organização e Estruturação do Código:

Criar uma hierarquia lógica e estruturada.
Facilita a abstração com classe que definem um modelo comum.

Polimorfismo

Permite tratamento genérico de objetos de subclasses como se fossem da classe base.
Aumenta a flexibilidade e reutilização do código.

Encapsulamento:

Controla acesso a dados e métodos, promovendo proteção e integridade.

Desvantagens da Herança:

Acoplamento excessivo:

Subclasses dependem fortemente da classe base, dificultando alterações na estrutura.

Complexidade:

Pode levar a uma hierarquia de classes complexa e difícil de entender e manter.

Rígida Estrutura:

Modificações na classe base podem ter impactos indesejados nas subclasses.
Difpicil adaptar a herança a novas necessidades que não foram previstas inicialmente.

Limitações de Flexibilidade:

Herança simples (uma classe base) pode ser limitante em comparação com composição, que permite mais flexibilidade ao combinar comportamentos de múltiplas fontes.

Problemas de Substituibilidade:

Subclasses podem não ser substituíveis pela classe se não seguirem corretamente o princípio de substituição de Liskov.

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

É necessária por esses motivos, **Identificação:** Marca a classe como apta a ser serializada, permitindo que a JVM saiba que o objeto pode ser convertido em bytes.

Controle: Garante que apenas objetos de classes que implementam Serializable possam ser serializados, prevenindo erros.

Preservação do Estado: Permite armazenar e recuperar o estado completo do objetivo (valores dos campos) de forma consistente.

Compatibilidade: Ajuda a manter a compatibilidade entre diferentes versões da classe através do uso de **'serialVersionUID'.**
Implementar 'Serializable' é essencial para garantir que um objeto possa ser corretamente convertido em formato binário para armazenamento e posteriormente recuperado.

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

O Paradigma funcional é utilizado pela API Stream no Java das seguintes maneiras:

Imutabilidade: As operações em streams não modificam a fonte de dados original; em vez disso retornam novos streams.

Funções de Alta Ordem: Streams permitem o suso de funções como argumentos para métodos como ' **map**', ' **filter**', e ' **reduce**'. Isso promove a utilização de expresso~es lambda e referências a métodos .

Operaçãoes Declarativas: A API permite expressar operações de processamento de dados de maneira declarativa, especificando o que fazer, e não como fazer. Por exemplo, **'stream.filte(x-> x >10).map(x -> x * 2).collect(Collector.toList())'.**

Lazy Evaluation: Muitas operações em streams são avaliadas de forma preguiçosa, ou seja são executadas apenas quando necessário, como quando uma operação terminal **(ex.: 'collect', 'forEach')** é invocada.

Composição de Funções: Streams facilitam a composição de várias operações em uma sequência de transformações, promovendo a legibilidade e a concisão do código.

Essas caracteristicas fazem com que a API Stream aproveite o paradigma funcional para fornecer uma maneira mais expressiva e eficiente de manipular coleções de dados.

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

É o Data Access Object (DAO).

Abstração: Separe a lógica de acesso a dados da lógica de negócios.

Operações CRUD: Fornece métodos claros para Create, Read, Updade e Delete.

Manutenção: Facilita a manutenção e a modificação do acesso a dados sem impactar outras partes do sistema.

Reutilização: Promove a reutilização de código através da centralização doas operações de acesso a dados.

O **DAO** encapsula a interação com a fonte de dados (como arquivos), permitindo que a aplicação se concentre na lógica de negócios, enquanto o DAO lida com a persistência.