POLO SETOR O – BRASÍLIA – DF
DESENVOLVIMENTO FULL STACK
NIVEL 1: INICIANDO O CAMINHO PELO JAVA
2024.2 FULL STACK
MAYCON MOURA

Título da Prática

Criação das Entidades e Sistema de Persistência

Objetivo da Prática

Demonstrar o uso de herança e polimorfismo na criação de classes para gerenciamento de dados de pessoas físicas e jurídicas, implementando persistência em arquivos binários utilizando Java.

Códigos

CadastroPOO.java

```
package cadastropoo;

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

public class CadastroPOO {

    public static void main(String[] args) {
        String nomeArquivoPessoaFisica = "pessoas_fisicas.dat";
        String nomeArquivoPessoaJuridica = "pessoas_juridicas.dat";

        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
```

```
        PessoaFisica pessoa1 = new PessoaFisica(1, "João", "123.456.789-00", 30);
        PessoaFisica pessoa2 = new PessoaFisica(2, "Maria", "987.654.321-00", 25);

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

        try {
            repo1.persistir(nomeArquivoPessoaFisica);
            System.out.println("Dados das pessoas físicas persistidos com sucesso.");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar(nomeArquivoPessoaFisica);

            List<PessoaFisica> pessoasFisicasRecuperadas = repo2.obterTodos();
            System.out.println("Dados das pessoas físicas recuperados:");
            for (PessoaFisica pessoa : pessoasFisicasRecuperadas) {
                pessoa.exibir();
            }

        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao persistir/recuperar dados das pessoas físicas: " +
e.getMessage());
        }

        System.out.println();

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

        PessoaJuridica empresa1 = new PessoaJuridica(1, "Empresa ABC", "123456780001-
01");
        PessoaJuridica empresa2 = new PessoaJuridica(2, "Empresa XXX", "987654320001-
02");

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

        try {
            repo3.persistir(nomeArquivoPessoaJuridica);
            System.out.println("Dados das pessoas jurídicas persistidos com sucesso.");

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar(nomeArquivoPessoaJuridica);

            List<PessoaJuridica> pessoasJuridicasRecuperadas = repo4.obterTodos();
            System.out.println("Dados das pessoas jurídicas recuperados:");
            for (PessoaJuridica empresa : pessoasJuridicasRecuperadas) {
                empresa.exibir();
```

```
            }

        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao persistir/recuperar dados das pessoas jurídicas: " +
e.getMessage());
        }
    }
}
```

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);
        System.out.println("Nome: " + nome);
    }
}
```

PessoaFisica

```
package model;

import java.io.Serializable;

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

    public PessoaFisica() {
        super();
    }

    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

```
package model;

import java.io.Serializable;
```

```
public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;

    public PessoaJuridica() {
        super();
    }

    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.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

public class PessoaFisicaRepo {

    private List<PessoaFisica> listaPessoasFisicas;

    public PessoaFisicaRepo() {
        this.listaPessoasFisicas = new ArrayList<>();
    }
```

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

    public void alterar(PessoaFisica pessoa) {
        for (int i = 0; i < listaPessoasFisicas.size(); i++) {
            if (listaPessoasFisicas.get(i).getId() == pessoa.getId()) {
                listaPessoasFisicas.set(i, pessoa);
                return;
            }
        }
        throw new IllegalArgumentException("Pessoa não encontrada para alteração");
    }

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

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

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

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

    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException
{
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasFisicas = (List<PessoaFisica>) ois.readObject();
        }
    }
}
```

PessoaJuridicaRepo

```
package model;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;

public class PessoaJuridicaRepo {
    private List<PessoaJuridica> listaPessoasJuridicas;

    public PessoaJuridicaRepo() {
        this.listaPessoasJuridicas = new ArrayList<>();
    }

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

    public void alterar(PessoaJuridica pessoa) {
        for (int i = 0; i < listaPessoasJuridicas.size(); i++) {
            if (listaPessoasJuridicas.get(i).getId() == pessoa.getId()) {
                listaPessoasJuridicas.set(i, pessoa);
                return;
            }
        }
        throw new IllegalArgumentException("Pessoa não encontrada para alteração");
    }

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

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

    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(listaPessoasJuridicas);
```

```
    }

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

    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException
{
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoasJuridicas = (List<PessoaJuridica>) ois.readObject();
        }
    }
}
```

Resultado da Execução

```
ID: 1
Nome: Jo□o
CPF: 123.456.789-00
Idade: 30
ID: 2
Nome: Maria
CPF: 987.654.321-00
Idade: 25

Dados das pessoas jur□dicas persistidos com sucesso.
Dados das pessoas jur□dicas recuperados:
ID: 1
Nome: Empresa ABC
CNPJ: 123456780001-01
ID: 2
Nome: Empresa XXX
CNPJ: 987654320001-02
```

Análise e Conclusão

Quais as vantagens e desvantagens do uso de herança?

Vantagens:

- **Reutilização de código:** Classes podem herdar comportamentos e atributos de classes, promovendo a reutilização de código.

- **Polimorfismo:** O tratamento de objetos de classes diferentes é feito de maneira uniforme através de referências de tipo pela classe que herda as atribuições.
- **Organização e estruturação:** A organização dos códigos é melhorada, facilitando a interpretação e reutilização.

Desvantagens:

- **Herança múltipla não suportada:** Em java só é possível a herança de apenas uma classe pai.

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

A interface Serializable em Java marca as classes para dizer que seus objetos podem ser transformados em bytes. Isso é útil porque permite salvar esses objetos em arquivos ou enviá-los pela internet. Quando implementamos Serializable, a Java Virtual Machine (JVM) faz todo o trabalho pesado de converter os objetos em uma sequência de bytes e depois reconvertê-los quando necessário.

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

> A API Stream em Java traz o paradigma funcional para o primeiro plano ao permitir que realizemos operações de alto nível em coleções de dados de maneira declarativa e expressiva. Em vez de usarmos loops tradicionais para manipular coleções, podemos utilizar métodos da API Stream para filtrar, mapear e reduzir dados de forma mais concisa e legível.

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

A forma mais comum de salvar dados em arquivos no Java é usando a Serialização de Objetos. Isso significa que marcamos nossas classes com a interface Serializable para que seus objetos possam ser transformados em bytes e gravados em arquivos binários. Com a serialização, fica fácil guardar e recuperar objetos Java, já que ela mantém a estrutura dos objetos intacta durante essas operações. É uma maneira eficiente e simples de trabalhar com persistência de dados.