Curso: Desenvolvimento Full Satck 2024.1   3° Semestre

Aluno: Rogério Benedito Geraldo

Polo: Nova Ourinhos

Tutora: Maria Manso

Disciplina: Iniciando o caminho pelo Java.

Link do Git: https://github.com/Rogeriobg/CadastroPOO

Códigos do 1° Procedimentos

Foi Criado um projeto do tipo Anti..Java no Netbeans com o nome de CadastroPOO.

Dentro de source packages foram criados dois packages o cadastropoo e o model, dentro do cadastropoo foi criado a classe Main.

Dentro do model foi criado as classes Pessoa, Pessoa Física, Pessoa Fisíca Repo, Pessoa Jurídica e Pessoa Jurídica Repo.

Códigos da Classe Main:

```
package cadastropoo;

import java.io.IOException;
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;
```

```
public class Main {
    public static void main(String[] args) {
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        repo1.inserir(new PessoaFisica(1, "João", "123.456.789-10", 30));
        repo1.inserir(new PessoaFisica(2, "Maria", "987.654.321-00", 25));
        repo1.inserir(new PessoaFisica(3, "Ricardo", "957.614.371-56", 45));

        try {
            repo1.persistir("pessoasFisicas.dat");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();
            repo2.recuperar("pessoasFisicas.dat");
            for (PessoaFisica pessoa : repo2.getPessoasFisicas()) {
                pessoa.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {

        }

        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        repo3.inserir(new PessoaJuridica(1, "Empresa A", "123456789"));
        repo3.inserir(new PessoaJuridica(2, "Empresa B", "987654321"));
```

```
        try {
            repo3.persistir("pessoasJuridicas.dat");

            PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();
            repo4.recuperar("pessoasJuridicas.dat");
            for (PessoaJuridica pessoa : repo4.getPessoasJuridicas()) {
                pessoa.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {

        }
    }
}
```

Códigos da Classe 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 void exibir() {
        System.out.println("ID: " + id);
        System.out.println("Nome: " + nome);
    }


}
```

Código da Classe PessoaFisica:

```
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;
    }

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


}
```

Códigos da Classe PessoaFisicaRepo:

```
package model;
```

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

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

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

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

    public void recuperar(String arquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream inputStream = new ObjectInputStream(new
FileInputStream(arquivo))) {
```

```
            pessoasFisicas = (List<PessoaFisica>) inputStream.readObject();
        }
    }

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

Código da Classe PessoaJuridica:

```
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;
```

```
    }

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


}
```

Código da Classe PessoaJuridicaRepo:

```
package model;


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

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

    public void inserir(PessoaJuridica pessoa) {
        pessoasJuridicas.add(pessoa);
```

```
    }

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

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

    public List<PessoaJuridica> getPessoasJuridicas() {
        return pessoasJuridicas;
    }
}
```

Resultado do Código Apresentado:

run:

ID: 1

Nome: João

CPF: 123.456.789-10

Idade: 30

ID: 2

Nome: Maria

CPF: 987.654.321-00

Idade: 25

ID: 3

Nome: Ricardo

CPF: 957.614.371-56

Idade: 45

ID: 1

Nome: Empresa A

CNPJ: 123456789

ID: 2

Nome: Empresa B

CNPJ: 987654321

BUILD SUCCESSFUL (total time: 0 seconds)

Conclussão:

Quais as vantagens e desvantagens do uso de herança?

A herança em programação é como passar características de uma geração para outra, tem como vantagens a economia de tempo e esforço ao reutilizar código, tem uma estrutura hierárquica clara e facilita a manutenção e extensão do software.

Como desvantagens, assim como nas famílias, pode haver complexidade excessiva e dependência excessiva entre classes, resultando em rigidez e dificuldades de manutenção. Assim, a herança é como uma herança familiar: traz benefícios significativos, mas deve ser gerenciada com cuidado para evitar complicações indesejadas.

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

Sem essa interface, os arquivos binários não teriam a habilidade de gravar e ler os objetos, resultando em um diálogo mudo entre o código e os dados.

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

A API Stream em Java é como um chef habilidoso que utiliza o paradigma funcional para preparar um banquete de dados. Com funções como map, filter e reduce, ela corta, filtra e combina elementos da coleção como ingredientes, resultando em pratos deliciosamente transformados e processados.

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

Quando trabalhamos com Java, adotamos o padrão de desenvolvimento chamado Serialização, onde objetos são transformados em uma sequência de bytes para serem armazenados em arquivos, criando uma espécie de fotografia dos dados que pode ser restaurada posteriormente.