Universidade Estácio
Campus Vargem Grande Paulista
Curso: Desenvolvimento Full Stack
Disciplina: Iniciando o Caminho Pelo Java
Turma: 2024.3
3° semestre letivo
Marcia da Silva e Souza

Título da Prática: Iniciando o caminho pelo java
Implementação de um cadastro de clientes em modo texto, com persistência em
arquivos, baseado na tecnologia Java.

Objetivo da prática:
Utilizar herança e polimorfismo na definição de entidades.
Utilizar persistência de objetos em arquivos binários.
Implementar uma interface cadastral em modo texto.
Utilizar o controle de exceções da plataforma Java.

No final do projeto, o aluno terá implementado um sistema cadastral em Java,
utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

Códigos utilizados na atividade prática 1:

main.java:

```
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) {
        try {
            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

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

            repo1.persistir("pessoasFisicas.dat");

            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

            repo2.recuperar("pessoasFisicas.dat");

            for (PessoaFisica pf : repo2.obterTodos()) {
                pf.exibir();
            }

            PessoaJuridicaRepo repo3 = new
PessoaJuridicaRepo();

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

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

```
            PessoaJuridicaRepo repo4 = new
PessoaJuridicaRepo();

             repo4.recuperar("pessoasJuridicas.dat");

            for (PessoaJuridica pj : repo4.obterTodos()) {
                pj.exibir();
            }
        } catch (IOException | ClassNotFoundException e) {
        }
    }
}
```

Pessoa.java:

```
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.java:

```
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() {
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
        System.out.println("Idade: " + idade);
        System.out.println("CPF: " + cpf);

    }
}
```

PessoaFisicaRepo.java:

```
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(PessoaFisica pessoa) {
        for (int i = 0; i < pessoasFisicas.size(); i++) {
            if (pessoasFisicas.get(i).getId() == pessoa.getId()) {
                pessoasFisicas.set(i, pessoa);
                return;
            }
        }
    }

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

    public PessoaFisica obter(int id) {
        for (PessoaFisica p : pessoasFisicas) {
            if (p.getId() == id) {
                return p;
            }
        }
        return null;
    }
```

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

    public void persistir(String nomeArquivo) throws IOException
{
        try (ObjectOutputStream oos = new
ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoasFisicas);
            System.out.println("Dados de pessoa fisica
armazenados.");
        }
    }

    public void recuperar(String nomeArquivo) throws
IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoasFisicas = (ArrayList<PessoaFisica>)
ois.readObject();
            System.out.println("Dados de pessoa fisica
recuperados.");
        }
    }
}
```

PessoaJuridica.java:

```
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() {
        System.out.println("ID: " + getId());
        System.out.println("Nome: " + getNome());
        System.out.println("CNPJ: " + cnpj);
    }
}
```

PessoaJuridicaRepo.java:

```
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(PessoaJuridica pessoa) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            if (pessoasJuridicas.get(i).getId() == pessoa.getId()) {
                pessoasJuridicas.set(i, pessoa);
                return;
            }
        }
    }

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

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

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

```
    public void persistir(String nomeArquivo) throws IOException
{
        try (ObjectOutputStream oos = new
ObjectOutputStream(new FileOutputStream(nomeArquivo))) {
            oos.writeObject(pessoasJuridicas);
            System.out.println("Dados de pessoa juridica
armazenados.");
        }
    }

    public void recuperar(String nomeArquivo) throws
IOException, ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            pessoasJuridicas = (ArrayList<PessoaJuridica>)
ois.readObject();
            System.out.println("Dados de pessoa juridica
recuperados.");
        }
    }
}
```

Resultado dos códigos apresentados:

```
run:
Dados de pessoa fisica armazenados.
Dados de pessoa fisica recuperados.
ID: 1
Nome: Jo�o Silva
Idade: 30
CPF: 123.456.789-00
ID: 2
Nome: Maria Oliveira
```

Idade: 25
CPF: 987.654.321-00
Dados de pessoa juridica armazenados.
Dados de pessoa juridica recuperados.
ID: 1
Nome: Empresa ABC
CNPJ: 12.345.678/0001-99
ID: 2
Nome: Empresa XYZ
CNPJ: 98.765.432/0001-88
BUILD SUCCESSFUL (total time: 0 seconds)

Análise e Conclusão:
Quais as vantagens e desvantagens do uso de herança?
Vantagens: Reutilização de códigos,permite criar sistemas mais estruturados,flexíveis e fáceis de manter.

Desvantagens: mudar uma subclasse pode afetar todas as subclasses,pode causar acoplamento entre classes e subclasses e pode ter um baixo acoplamento entre as classes.

Por que a interface Serializable é necessária ao efetuar persistência em arquivos binários?
Ela permite que objetos java possam ser transformados em uma sequência de bytes,que podem ser transmitidos pelas redes ou armazenados em arquivo.

Como o paradigma funcional é utilizado pela API stream no Java?
Para permitir que o usuário processe sequências de elementos de forma declarativa

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

O padrão mais comum utilizado em java Data access object (DAO)