

Nome do Aluno: Matheus Oliveira de Jesus

Matrícula: 2023.10.10087-8

Disciplina: Iniciando o Caminho Pelo Java

Turma: RPG0014

Semestre Letivo: 1º/2024

Este relatório tem como objetivo apresentar códigos e resultados referente aos exercícios solicitados em: Missão Prática | Nível 1 | Mundo 3.

Respostas: 1º Procedimento | Criação das Entidades e Sistema de Persistência

1 – Título da Prática: Iniciando o caminho pelo Java

2 – 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.

3 – Códigos gerados:

1) 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);

    }

}
```

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

2) GeradorID.java

```
package model;

public class GeradorID {

    private static int currentId = 1;

    public static int getNextId() {

        return currentId++;

    }

}
```

3) PessoaFisica.java

```
package model;

public class PessoaFisica extends Pessoa {

    private String cpf;

    private int idade;

    public PessoaFisica() {

        super(GeradorID.getNextId(), "");

    }

    public PessoaFisica(String nome, String cpf, int idade) {

        super(GeradorID.getNextId(), 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);
    }
}
```

4) PessoaJuridica.java

```
package model;

public class PessoaJuridica extends Pessoa {

    private String cnpj;

    public PessoaJuridica() {

        super(GeradorID.getNextId(), "");

    }

    public PessoaJuridica(int id, String nome, String cnpj) {

        super(GeradorID.getNextId(), 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);

    }

}
```

5) PessoaFisicaRepo.java

```
package model;

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

public class PessoaFisicaRepo {

    private ArrayList<PessoaFisica> pessoasFisicas = new ArrayList<>();

    public void inserir(PessoaFisica pessoaFisica) {

        pessoasFisicas.add(pessoaFisica);
    }


    public void alterar(PessoaFisica pessoaFisica) {

        for (int i = 0; i < pessoasFisicas.size(); i++) {

            if (pessoasFisicas.get(i).getId() == pessoaFisica.getId()) {

                pessoasFisicas.set(i, pessoaFisica);

                return;

            }

        }

    }


    public void excluir(int id) {

        Iterator<PessoaFisica> iterator = pessoasFisicas.iterator();

        while (iterator.hasNext()) {

            if (iterator.next().getId() == id) {

                iterator.remove();

                return;

            }

        }

    }

}
```

```
}
```

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

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

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

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

6) PessoaJuridicaRepo.java

```
package model;

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

public class PessoaJuridicaRepo {

    private ArrayList<PessoaJuridica> pessoasJuridicas = new ArrayList<>();

    public void inserir(PessoaJuridica pessoaJuridica) {

        pessoasJuridicas.add(pessoaJuridica);

    }

    public void alterar(PessoaJuridica pessoaJuridica) {

        for (int i = 0; i < pessoasJuridicas.size(); i++) {

            if (pessoasJuridicas.get(i).getId() == pessoaJuridica.getId()) {

                pessoasJuridicas.set(i, pessoaJuridica);

                return;

            }

        }

    }

    public void excluir(int id) {

        Iterator<PessoaJuridica> iterator = pessoasJuridicas.iterator();

        while (iterator.hasNext()) {

            if (iterator.next().getId() == id) {

                iterator.remove();

                return;

            }

        }

    }

}
```



```

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

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

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

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

```

7) Main.java

```
package model;

import java.io.IOException;

public class Main {

    public static void main(String[] args) {

        try {

            PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

            PessoaFisica pessoa1 = new PessoaFisica("Ana", "11111111111", 25);

            PessoaFisica pessoa2 = new PessoaFisica("Carlos", "22222222222", 52);

            repo1.inserir(pessoa1);

            repo1.inserir(pessoa2);


            String arquivoPF = "pessoas_fisicas.dat";

            repo1.persistir(arquivoPF);


            PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

            repo2.recuperar(arquivoPF);


            System.out.println("Dados de Pessoa Física Armazenados.");

            System.out.println("Dados de Pessoa Física Recuperados.");

            for (PessoaFisica pf : repo2.obterTodos()) {

                pf.exibir();

            }


            PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
```

```

        PessoaJuridica pj1 = new PessoaJuridica(1, "XPTO
Sales","3333333333333333");

        PessoaJuridica pj2 = new PessoaJuridica(2, "XPTO Soluções",
"4444444444444444");

        repo3.inserir(pj1);

        repo3.inserir(pj2);


        String arquivoPJ = "pessoas_juridicas.dat";

        repo3.persistir(arquivoPJ);


        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

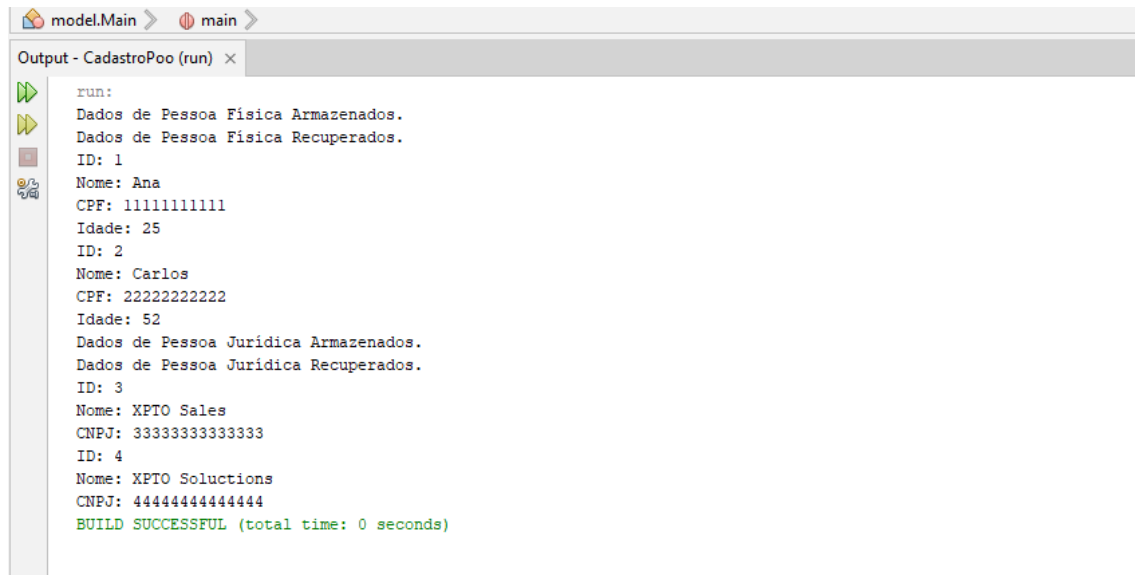

        repo4.recuperar(arquivoPJ);


        System.out.println("Dados de Pessoa Jurídica Armazenados.");
        System.out.println("Dados de Pessoa Jurídica Recuperados.");
        for (PessoaJuridica pj : repo4.obterTodos()) {
            pj.exibir();
        }

    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}
}
}

```

4 – Resultado dos códigos executados:



```
run:
Dados de Pessoa Fisica Armazenados.
Dados de Pessoa Fisica Recuperados.
ID: 1
Nome: Ana
CPF: 11111111111
Idade: 25
ID: 2
Nome: Carlos
CPF: 22222222222
Idade: 52
Dados de Pessoa Juridica Armazenados.
Dados de Pessoa Juridica Recuperados.
ID: 3
Nome: XPTO Sales
CNPJ: 33333333333333
ID: 4
Nome: XPTO Solutions
CNPJ: 44444444444444
BUILD SUCCESSFUL (total time: 0 seconds)
```

5 – Análise e Conclusão:

a. Quais as vantagens e desvantagens do uso de herança?

Vantagens da herança:

- Reutilização de código: Classes filhas herdam métodos e atributos da classe pai, evitando duplicação de código.
- Facilidade de manutenção: Alterações no código da classe pai afetam todas as subclasses, simplificando a manutenção.
- Organização: Ajuda a estruturar o código, criando uma hierarquia que reflete a relação entre as classes.
- Polimorfismo: Permite tratar objetos de diferentes classes de forma unificada, tornando o código mais flexível.

Desvantagens da herança:

- Acoplamento forte: As subclasses dependem muito da classe pai, o que pode tornar difícil alterar ou expandir o código.
- Rigidez: A herança fixa a estrutura das classes, o que pode limitar a flexibilidade de modificar o comportamento de subclasses.

- Risco de complexidade: Muitas camadas de herança podem tornar o código difícil de entender e manter.
- Sobrecarga: Alterações na classe pai podem impactar negativamente todas as subclasses, aumentando o risco de bugs.

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

A interface `Serializable` é usada para que um objeto possa ser "guardado" em um arquivo. Quando um programa quer salvar os dados de um objeto em um arquivo binário, ele precisa transformar esse objeto em uma sequência de dados simples, como zeros e uns. A `Serializable` permite essa transformação, garantindo que o objeto possa ser salvo e recuperado depois, do jeito que estava. Sem ela, o Java não saberia como "traduzir" o objeto para guardar no arquivo.

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

A API `Stream` do Java usa o paradigma funcional para processar dados de forma mais simples e eficiente. Com ela, você pode realizar operações como filtrar, mapear e reduzir coleções de dados sem precisar escrever loops manuais. O paradigma funcional permite usar funções como parâmetros e trabalhar com dados de forma declarativa, ou seja, você foca no "o que" quer fazer, e não no "como".

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

Quando trabalhamos com persistência de dados em arquivos no Java, o padrão de desenvolvimento mais comum adotado é o DAO (`Data Access Object`). Esse padrão separa a lógica de acesso a dados da lógica de negócios do sistema.

O DAO cria uma camada dedicada para realizar operações de leitura, escrita, atualização e exclusão de dados nos arquivos, mantendo o código organizado e facilitando a manutenção. Ele permite que as mudanças no modo de persistência não afetem o restante da aplicação.