Estácio	**Universidade Estácio** **Polo São Lourenço da Mata** **Desenvolvimento Full Stack** **Semestre 2024.1**	Disciplina: **Iniciando o Caminho Pelo Java.** Aluno: **Manoel José** Matrícula: 202301361117 Turma: 2023.1

Criação das Entidades e Sistema de Persistência.

Objetivos da Prática:

1. Utilizar herança e polimorfismo na definição de entidades.
2. Utilizar persistência de objetos em arquivos binários.
3. Implementar uma interface cadastral em modo texto.
4. Utilizar o controle de exceções da plataforma Java.
5. 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.

Todos os códigos solicitados neste roteiro de aula:

1- Código da Classe Principal (CadastroPOO.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;
            }
        }
        throw new IllegalArgumentException("Pessoa não encontrada para alteração.");
    }

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

    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null; // Retorna null se não encontrar a pessoa com o id especificado
    }

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

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

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

Resultado da execução do código anterior:

```
run:
Pessoas f�sicas recuperadas:
ID: 1, Nome: Ana
CPF: 111.111.111-11, Idade: 25
```

```
ID: 2, Nome: Carlos
CPF: 222.222.222-22, Idade: 52

Pessoas jur�dicas recuperadas:
ID: 3, Nome: XPTO Sales
CNPJ: 12.345.678/0001-90
ID: 4, Nome: XPTO Solutions
CNPJ: 98.765.432/0001-21
BUILD SUCCESSFUL (total time: 0 seconds)
```

2- Código da Classe Pessoa (Pessoa.java.):

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

public class Pessoa implements Serializable {
    private int id;
    private String nome;

    // Construtor padrão
    public Pessoa() {
    }

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

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

    // Getters e Setters para id
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    // Getters e Setters para nome
    public String getNome() {
```

```
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    // Método main para teste
    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa();
        pessoa1.setId(1);
        pessoa1.setNome("João");
        pessoa1.exibir();

        // Utilizando os getters
        System.out.println("ID: " + pessoa1.getId());
        System.out.println("Nome: " + pessoa1.getNome());
    }
}
```

Resultado da execução do código anterior:

```
run:
ID: 1, Nome: Jo�o
ID: 1
Nome: Jo�o
BUILD SUCCESSFUL (total time: 0 seconds)
```

3- Código da Classe Pessoa Fisica (PessoaFisica.java):

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

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

    // Construtor
    public PessoaFisica() {
    }

    // Construtor completo
    public PessoaFisica(int id, String nome, String cpf, int idade) {
        super(id, nome);
        this.cpf = cpf;
        this.idade = idade;
```

```
    }

    // Método exibir
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da classe pai
        System.out.println("CPF: " + cpf + ", Idade: " + idade);
    }

    // Getters e Setters para cpf
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    // Getters e Setters para idade
    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }

    // Método main para teste
    public static void main(String[] args) {
        PessoaFisica pessoa1 = new PessoaFisica(1, "João", "123.456.789-10", 30);
        pessoa1.exibir();

        // Utilizando os getters
        System.out.println("CPF: " + pessoa1.getCpf());
        System.out.println("Idade: " + pessoa1.getIdade());
    }
}
```

Resultado da execução do código anterior:

```
run:
ID: 1, Nome: Jo�o
CPF: 123.456.789-10, Idade: 30
CPF: 123.456.789-10
Idade: 30
```

BUILD SUCCESSFUL (total time: 0 seconds)

4- Código da Classe Pessoa Jurídica (PessoaJuridica.java):

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

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

    // Construtor padrão
    public PessoaJuridica() {
    }

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

    // Método exibir polimórfico
    @Override
    public void exibir() {
        super.exibir(); // Chama o método exibir da classe pai
        System.out.println("CNPJ: " + cnpj);
    }

    // Getters e Setters para cnpj
    public String getCnpj() {
        return cnpj;
    }

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

    // Método main para teste
    public static void main(String[] args) {
        PessoaJuridica pessoa1 = new PessoaJuridica(1, "Empresa XYZ", "12.345.678/0001-90");
        pessoa1.exibir();

        // Utilizando o getter
        System.out.println("CNPJ: " + pessoa1.getCnpj());
    }
}
```

Resultado da execução do código anterior:

```
run:
ID: 1, Nome: Empresa XYZ
CNPJ: 12.345.678/0001-90
CNPJ: 12.345.678/0001-90
BUILD SUCCESSFUL (total time: 0 seconds)
```

5- Código da Classe Pessoa Física Repo (PessoaFisicaRepo.java):

```
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;

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;
            }
        }
        throw new IllegalArgumentException("Pessoa não encontrada para alteração.");
    }

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

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

```
        }
        return null; // Retorna null se não encontrar a pessoa com o id especificado
    }

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

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

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

    public static void main(String[] args) {
        // Teste para PessoaFisicaRepo
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        PessoaFisica pessoaFisica1 = new PessoaFisica(1, "Ana", "111.111.111-11", 25);
        PessoaFisica pessoaFisica2 = new PessoaFisica(2, "Carlos", "222.222.222-22", 52);

        repo1.inserir(pessoaFisica1);
        repo1.inserir(pessoaFisica2);

        try {
            repo1.persistir("pessoasFisicas.dat");
        } catch (IOException e) {
            System.err.println("Erro ao persistir pessoas físicas: " + e.getMessage());
        }

        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        try {
            repo2.recuperar("pessoasFisicas.dat");
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao recuperar pessoas físicas: " + e.getMessage());
        }
```

```
        System.out.println("Pessoas físicas recuperadas:");
        for (PessoaFisica pessoa : repo2.obterTodos()) {
            pessoa.exibir();
        }

        // Teste para PessoaJuridicaRepo
        PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();
        PessoaJuridica pessoaJuridica1 = new PessoaJuridica(3, "XPTO Sales",
"12.345.678/0001-90");
        PessoaJuridica pessoaJuridica2 = new PessoaJuridica(4, "XPTO Solutions",
"98.765.432/0001-21");

        repo3.inserir(pessoaJuridica1);
        repo3.inserir(pessoaJuridica2);

        try {
            repo3.persistir("pessoasJuridicas.dat");
        } catch (IOException e) {
            System.err.println("Erro ao persistir pessoas jurídicas: " + e.getMessage());
        }

        PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

        try {
            repo4.recuperar("pessoasJuridicas.dat");
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Erro ao recuperar pessoas jurídicas: " + e.getMessage());
        }

        System.out.println("\nPessoas jurídicas recuperadas:");
        for (PessoaJuridica pessoa : repo4.obterTodos()) {
            pessoa.exibir();
        }
    }

}
```

Resultado da execução do código anterior:

```
run:
Pessoas f�sicas recuperadas:
ID: 1, Nome: Ana
CPF: 111.111.111-11, Idade: 25
ID: 2, Nome: Carlos
```

CPF: 222.222.222-22, Idade: 52

Pessoas jur&#65533;dicas recuperadas:
ID: 3, Nome: XPTO Sales
CNPJ: 12.345.678/0001-90
ID: 4, Nome: XPTO Solutions
CNPJ: 98.765.432/0001-21
BUILD SUCCESSFUL (total time: 0 seconds)

6- Código da Classe Pessoa Jurídica Repo (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;
            }
        }
        throw new IllegalArgumentException("Pessoa não encontrada para alteração.");
    }

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

    public PessoaJuridica obter(int id) {
        for (PessoaJuridica pessoa : pessoasJuridicas) {
            if (pessoa.getId() == id) {
                return pessoa;
            }
        }
        return null; // Retorna null se não encontrar a pessoa com o id especificado
    }
```

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

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

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

Resultado da execução do código anterior:

```
run:
Pessoas Jur�dicas:
ID: 1, Nome: Empresa A
CNPJ: 12.345.678/0001-90
ID: 2, Nome: Empresa B
CNPJ: 98.765.432/0001-21
Dados de pessoas jur�dicas persistidos com sucesso.
Dados de pessoas jur�dicas recuperados com sucesso.

Pessoas Jur�dicas ap�s recupera��o:
ID: 1, Nome: Empresa A
CNPJ: 12.345.678/0001-90
ID: 2, Nome: Empresa B
CNPJ: 98.765.432/0001-21
BUILD SUCCESSFUL (total time: 0 seconds)
```

Análise e Conclusão:

A) **Quais as vantagens e desvantagens do uso de Herança?**

R: Vantagens:

- Reutilização de código: Classes filhas podem herdar atributos e métodos das classes pai, evitando a repetição de código.

- Polimorfismo: Permite tratar objetos de classes diferentes de maneira uniforme, facilitando a implementação de interfaces genéricas.
- Estruturação hierárquica: Ajuda a organizar e estruturar o código de forma mais intuitiva, refletindo relações entre objetos do mundo real.

Desvantagens:

- Acoplamento forte: Pode resultar em um acoplamento forte entre classes pai e filhas, tornando o código mais difícil de manter e modificar.
- Herança múltipla: Em linguagens que suportam herança múltipla, pode levar a problemas de ambiguidade e complexidade.
- Fragilidade: Alterações na classe pai podem afetar todas as classes filhas, o que pode causar problemas de compatibilidade e quebra de encapsulamento.

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

R: A interface Serializable é necessária ao efetuar persistência em arquivos binários porque ela indica que os objetos de uma classe podem ser convertidos em uma sequência de bytes, permitindo que sejam salvos em arquivos ou transmitidos através da rede. Isso é essencial para a serialização e deserialização de objetos, o que é necessário para armazenar objetos em arquivos binários de forma que possam ser recuperados posteriormente.

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

R: A API stream no Java utiliza o paradigma funcional para operar em coleções de dados de forma mais eficiente e concisa. Ela permite o encadeamento de operações de alto nível, como map(), filter(), reduce(), entre outras, em uma sequência de elementos. Isso facilita a escrita de código mais declarativo e expressivo, tornando o processamento de coleções mais legível e fácil de entender.

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

R: No desenvolvimento em Java, um padrão comum na persistência de dados em arquivos é o uso de classes de repositório (também conhecidas como DAO - Data Access Object). Essas classes são responsáveis por abstrair o acesso aos dados, fornecendo métodos para inserir, atualizar, excluir e recuperar objetos de um determinado tipo a partir de arquivos ou de um banco de dados. O uso de classes de repositório ajuda a manter uma separação clara entre a lógica de negócios e a lógica de persistência, facilitando a manutenção e a evolução do código.