Estácio

Pólo Tatui
Curso: Desenvolvimento Full Stack
Disciplina: Nível 1: Iniciando o Caminho Pelo Java
Integrantes da Prática: Clayton Prebelli Pires

Relatório de Prática - Cadastro de Pessoas com Persistência

1. Título da Prática

Desenvolvimento de um Sistema de Cadastro de Pessoas com Persistência em Arquivos Binários

2. Objetivo da Prática

O objetivo desta prática é implementar um sistema de cadastro de pessoas físicas e jurídicas, utilizando conceitos de herança, polimorfismo e persistência em arquivos binários em Java. O sistema também utiliza a interface Serializable para a serialização dos objetos e aborda a manipulação de dados por meio da API Stream do Java.

3. Códigos Solicitados

Aqui você deve incluir os códigos implementados durante a prática. Inclua todos os códigos solicitados no roteiro de aula, como as classes Pessoa, PessoaFisica, PessoaJuridica, PessoaFisicaRepo, PessoaJuridicaRepo e a classe principal Main. Os códigos devem ser apresentados em formato adequado para leitura.

Classe Pessoa

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
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;
  }

  // Getters e Setters
  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;
    }

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

Classe PessoaFisica

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package model;
```

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

    // Construtor padrão
    public PessoaFisica() {
    }

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

    // Getters e Setters
    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;
    }

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

Classe PessoaJuridica

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package model;

public class PessoaJuridica extends Pessoa {
    private String cnpj;

    // Construtor padrão
    public PessoaJuridica() {
    }

    // Construtor completo
    public PessoaJuridica(int id, String nome, String cnpj) {
```

```
        super(id, nome);
        this.cnpj = cnpj;
    }

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

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

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

Classe PessoaFisicaRrpo

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package model;
import java.io.*;
```

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

public class PessoaFisicaRepo {
  private List<PessoaFisica> listaPessoaFisica;

  // Construtor
  public PessoaFisicaRepo() {
    this.listaPessoaFisica = new ArrayList<>();
  }

  // Método inserir
  public void inserir(PessoaFisica pessoaFisica) {
    listaPessoaFisica.add(pessoaFisica);
  }

  // Método alterar
  public void alterar(PessoaFisica pessoaFisica) {
    for (int i = 0; i < listaPessoaFisica.size(); i++) {
      if (listaPessoaFisica.get(i).getId() == pessoaFisica.getId()) {
        listaPessoaFisica.set(i, pessoaFisica);
        return;
      }
    }
  }

  // Método excluir
  public void excluir(int id) {
```

```
        listaPessoaFisica.removeIf(pessoaFisica -> pessoaFisica.getId() == id);
    }

    // Método obter por ID
    public PessoaFisica obter(int id) {
        for (PessoaFisica pessoaFisica : listaPessoaFisica) {
            if (pessoaFisica.getId() == id) {
                return pessoaFisica;
            }
        }
        return null;
    }

    // Método obterTodos
    public List<PessoaFisica> obterTodos() {
        return new ArrayList<>(listaPessoaFisica);
    }

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

    // Método recuperar
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
```

```
      try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
          listaPessoaFisica = (List<PessoaFisica>) ois.readObject();
      }
   }
}
```

Classe PessoaJuridicaRepo

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt
to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit
this template
 */
package model;

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

public class PessoaJuridicaRepo {
   private List<PessoaJuridica> listaPessoaJuridica;

   // Construtor
   public PessoaJuridicaRepo() {
      this.listaPessoaJuridica = new ArrayList<>();
   }

   // Método inserir
```

```
public void inserir(PessoaJuridica pessoaJuridica) {
   listaPessoaJuridica.add(pessoaJuridica);
}

// Método alterar
public void alterar(PessoaJuridica pessoaJuridica) {
   for (int i = 0; i < listaPessoaJuridica.size(); i++) {
      if (listaPessoaJuridica.get(i).getId() == pessoaJuridica.getId()) {
         listaPessoaJuridica.set(i, pessoaJuridica);
         return;
      }
   }
}

// Método excluir
public void excluir(int id) {
   listaPessoaJuridica.removeIf(pessoaJuridica -> pessoaJuridica.getId() == id);
}

// Método obter por ID
public PessoaJuridica obter(int id) {
   for (PessoaJuridica pessoaJuridica : listaPessoaJuridica) {
      if (pessoaJuridica.getId() == id) {
         return pessoaJuridica;
      }
   }
   return null;
}
```

```
    // Método obterTodos
    public List<PessoaJuridica> obterTodos() {
        return new ArrayList<>(listaPessoaJuridica);
    }

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

    // Método recuperar
    public void recuperar(String nomeArquivo) throws IOException,
ClassNotFoundException {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(nomeArquivo))) {
            listaPessoaJuridica = (List<PessoaJuridica>) ois.readObject();
        }
    }
}
```

Classe Main

```
import model.PessoaFisica;
import model.PessoaFisicaRepo;
import model.PessoaJuridica;
import model.PessoaJuridicaRepo;
```

```
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        // Repositório de Pessoas Físicas
        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();

        // Adicionar duas pessoas físicas
        repo1.inserir(new PessoaFisica(1, "João Silva", "123.456.789-00", 30));
        repo1.inserir(new PessoaFisica(2, "Maria Souza", "987.654.321-00", 25));

        // Persistir as pessoas físicas no arquivo
        String arquivoPessoaFisica = "pessoasFisicas.dat";
        try {
            repo1.persistir(arquivoPessoaFisica);
        } catch (IOException e) {
            System.out.println("Erro ao persistir pessoas físicas: " + e.getMessage());
        }

        // Repositório para recuperar as pessoas físicas
        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        // Recuperar as pessoas físicas do arquivo
        try {
            repo2.recuperar(arquivoPessoaFisica);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar pessoas físicas: " + e.getMessage());
```

```
}

// Exibir todas as pessoas físicas recuperadas
System.out.println("Pessoas Físicas Recuperadas:");
for (PessoaFisica pessoaFisica : repo2.obterTodos()) {
    pessoaFisica.exibir();
}

// Repositório de Pessoas Jurídicas
PessoaJuridicaRepo repo3 = new PessoaJuridicaRepo();

// Adicionar duas pessoas jurídicas
repo3.inserir(new PessoaJuridica(1, "Empresa X", "00.000.000/0001-00"));
repo3.inserir(new PessoaJuridica(2, "Empresa Y", "11.111.111/0001-11"));

// Persistir as pessoas jurídicas no arquivo
String arquivoPessoaJuridica = "pessoasJuridicas.dat";
try {
    repo3.persistir(arquivoPessoaJuridica);
} catch (IOException e) {
    System.out.println("Erro ao persistir pessoas jurídicas: " + e.getMessage());
}

// Repositório para recuperar as pessoas jurídicas
PessoaJuridicaRepo repo4 = new PessoaJuridicaRepo();

// Recuperar as pessoas jurídicas do arquivo
try {
```

```
            repo4.recuperar(arquivoPessoaJuridica);
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Erro ao recuperar pessoas jurídicas: " + e.getMessage());
        }

        // Exibir todas as pessoas jurídicas recuperadas
        System.out.println("Pessoas Jurídicas Recuperadas:");
        for (PessoaJuridica pessoaJuridica : repo4.obterTodos()) {
            pessoaJuridica.exibir();
        }
    }
}
```

Resultados da Execução dos Códigos

```
run:
Pessoas F�sicas Recuperadas:
ID: 1, Nome: Jo�o Silva
CPF: 123.456.789-00, Idade: 30
ID: 2, Nome: Maria Souza
CPF: 987.654.321-00, Idade: 25
Pessoas Jur�dicas Recuperadas:
ID: 1, Nome: Empresa X
CNPJ: 00.000.000/0001-00
ID: 2, Nome: Empresa Y
CNPJ: 11.111.111/0001-11
BUILD SUCCESSFUL (total time: 0 seconds)
```

Vantagens e Desvantagens do Uso de Herança

- **Vantagens**:
    - Reutilização de código: permite reutilizar atributos e métodos da classe base nas classes derivadas.

    - Facilidade de manutenção: mudanças na classe base podem refletir nas classes derivadas.
    - Polimorfismo: permite o tratamento uniforme de objetos de classes derivadas.
- **Desvantagens**:
    - Acoplamento: herança pode criar uma relação forte entre classes, dificultando mudanças.
    - Complexidade: pode tornar o design do sistema mais complexo e menos flexível.
    - Fragilidade: mudanças na classe base podem afetar as classes derivadas de maneiras inesperadas.

Por que a Interface Serializable é Necessária ao Efetuar Persistência em Arquivos Binários?

A interface Serializable é necessária porque ela permite que um objeto seja convertido em um fluxo de bytes, o que é essencial para armazenar o estado do objeto em arquivos binários. Sem essa interface, o Java não permite serializar e desserializar objetos, impedindo a persistência dos dados em um formato binário.

Como o Paradigma Funcional é Utilizado pela API Stream no Java?

A API Stream em Java permite a manipulação de coleções de dados de maneira funcional. Ela utiliza conceitos como **lambda expressions**, **map**, **filter**, **reduce** e outros para permitir operações sobre coleções de forma concisa e expressiva, promovendo um estilo de programação mais funcional, que enfatiza a imutabilidade e evita efeitos colaterais.

Quando Trabalhamos com Java, Qual Padrão de Desenvolvimento é Adotado na Persistência de Dados em Arquivos?

Ao trabalhar com persistência de dados em arquivos em Java, o padrão de desenvolvimento comumente adotado é o **Data Access Object (DAO)**. Este padrão separa a lógica de negócios da lógica de acesso a dados, promovendo uma arquitetura mais organizada e facilitando a manutenção e evolução do código.

Repositório do Projeto

https://github.com/ClaytonPrebelli/CadastroPOO

O que são elementos estáticos e qual o motivo para o método main adotar esse modificador?

Elementos estáticos em Java são membros de uma classe (atributos ou métodos) que pertencem à classe em si, em vez de pertencer a instâncias específicas dessa classe. Esses membros são declarados usando a palavra-chave static.

- **Atributos estáticos**: São compartilhados entre todas as instâncias da classe. Se um atributo é modificado em uma instância, a mudança é refletida em todas as outras instâncias da mesma classe.
- **Métodos estáticos**: Podem ser chamados sem precisar instanciar a classe. Eles só podem acessar outros membros estáticos da classe.

Motivo para o método main ser estático:

- O método main é o ponto de entrada de um programa Java. Ele precisa ser acessível pela JVM (Java Virtual Machine) sem que seja necessário criar uma instância da classe que o contém.
- Como o main é o primeiro método a ser executado em um programa Java, ele deve ser estático para que a JVM possa chamá-lo diretamente, sem precisar de um objeto da classe.

Para que serve a classe Scanner?

A classe Scanner faz parte do pacote java.util e é usada para ler a entrada de dados. Ela pode ser utilizada para receber entradas de diferentes fontes, como o teclado (entrada padrão), arquivos, ou strings.

- **Uso comum**: A classe Scanner é frequentemente usada para ler entradas do usuário a partir do console (teclado), o que facilita a criação de programas interativos.
- **Métodos**: Ela fornece métodos como nextInt(), nextLine(), nextDouble(), etc., para ler diferentes tipos de dados (inteiros, strings, números de ponto flutuante) a partir da entrada.

Exemplo:

```
Scanner scanner = new Scanner(System.in);

System.out.print("Digite seu nome: ");

String nome = scanner.nextLine();

System.out.println("Olá, " + nome + "!");
```

Nesse exemplo, Scanner é usado para ler uma linha de texto digitada pelo usuário.

Como o uso de classes de repositório impactou na organização do código?

O uso de classes de repositório, como PessoaFisicaRepo e PessoaJuridicaRepo, ajuda a organizar e separar a lógica de negócio da lógica de persistência de dados, seguindo boas práticas de programação orientada a objetos.

Impactos na organização do código:

- **Separação de responsabilidades**: As classes de repositório encapsulam toda a lógica relacionada ao armazenamento, modificação, recuperação e exclusão de objetos. Isso mantém o código relacionado à manipulação de dados isolado da lógica de aplicação, facilitando a manutenção e evolução do código.
- **Reutilização**: As classes de repositório podem ser reutilizadas em diferentes partes do programa ou em diferentes projetos, sem a necessidade de duplicar código.
- **Facilidade de manutenção**: Se houver necessidade de alterar a maneira como os dados são armazenados (por exemplo, mudando de um armazenamento em memória para um banco de dados), essas mudanças podem ser feitas nas classes de repositório sem afetar o restante do código.
- **Melhoria da legibilidade**: Dividindo a lógica do programa em classes específicas (repositórios, entidades, etc.), o código torna-se mais modular e fácil de entender.

Resumo

- **Elementos estáticos** são membros da classe que pertencem à classe e não a uma instância. O método main é estático para ser chamado pela JVM sem a necessidade de criar uma instância.
- A classe **Scanner** é usada para ler a entrada do usuário a partir do console ou outras fontes.
- As **classes de repositório** melhoram a organização do código ao separar a lógica de manipulação de dados da lógica da aplicação, promovendo reutilização, facilidade de manutenção e melhor legibilidade.