Missão Prática | Nível 1 | Mundo 3

Eduardo Eugênio de Araujo e Silva Domingues 202302579043

1172 POLO CASA CAIADA - OLINDA - PE
Iniciando o caminho pelo Java – RPG0014 – 2024.1

Objetivo da Prática

Teve como objetivo ensinar conceitos e práticas fundamentais de programação orientada a objetos e manipulação de listas.

- Manipulação de listas: Como percorrer e manipular elementos em uma lista usando estruturas de repetição como o loop for. Isso é essencial para entender como lidar com conjuntos de dados em um programa.
- Objetos: Manipulação de objetos para entender como criar e usar objetos em programas Java, incluindo como acessar seus atributos e métodos.
- Comparações de objetos: A comparação dos IDs para ensinar conceitos de comparação de objetos e como verificar a igualdade de objetos em Java.
- Métodos de acesso e modificação: A utilização de métodos como get e set é uma introdução aos conceitos de métodos de acesso e modificação, fundamentais na programação orientada a objetos.
- Boas práticas: Encapsulamento e modularidade, dividindo o código em funções e métodos menores e mais gerenciáveis.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

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);
    }
}
```

PessoaFisica.java

```
package model;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
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() {
        super.exibir();
        System.out.println("CPF: " + cpf);
        System.out.println("Idade: " + idade);
    }
}
```

PessoaFisicaRepo.java

```
package model;

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

public class PessoaFisicaRepo {
    private List<PessoaFisica> pessoasFisicas;

    public PessoaFisicaRepo() {
        this.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) {
    for (int i = 0; i < pessoasFisicas.size(); i++) {
        if (pessoasFisicas.get(i).getId() == id) {
            pessoasFisicas.remove(i);
            return;
        }
    }
}

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

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

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

    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoasFisicas = (List<PessoaFisica>) in.readObject();
        }
    }
}
```

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() {
        super.exibir();
        System.out.println("CNPJ: " + cnpj);
    }
}
```

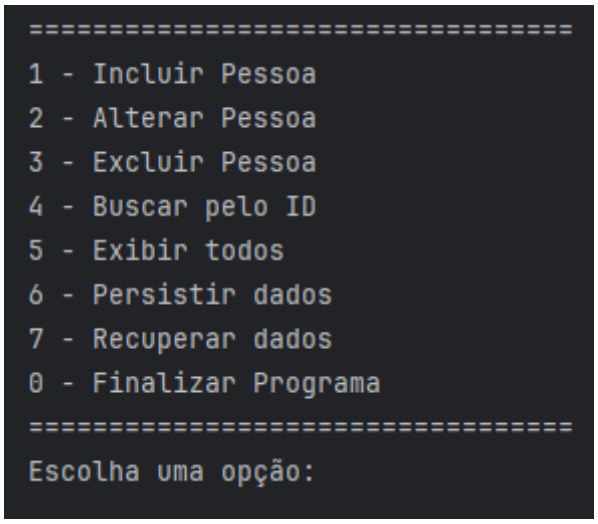
PessoaJuridicaRepo.java

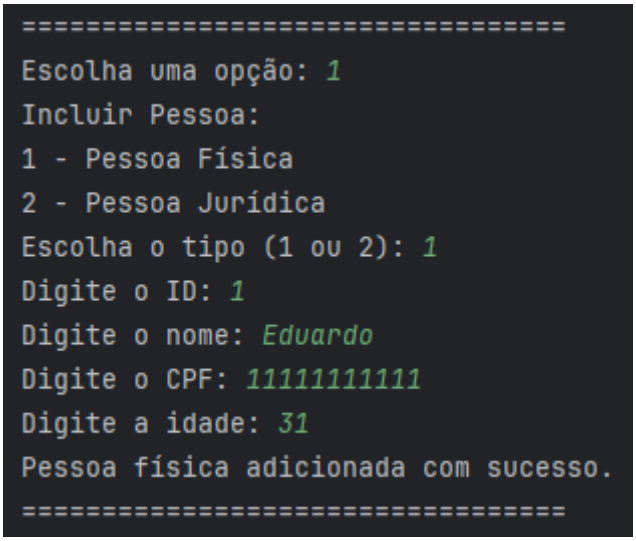
```
package model;

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

```
import java.util.List;

public class PessoaJuridicaRepo {
    private List<PessoaJuridica> pessoasJuridicas;

    public PessoaJuridicaRepo() {
        this.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) {
        for (int i = 0; i < pessoasJuridicas.size(); i++) {
            if (pessoasJuridicas.get(i).getId() == id) {
```

```
            pessoasJuridicas.remove(i);
            return;
        }
    }
}

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

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

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

```
    @SuppressWarnings("unchecked")
    public void recuperar(String nomeArquivo) throws IOException, ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(nomeArquivo))) {
            pessoasJuridicas = (List<PessoaJuridica>) in.readObject();
        }
    }
}
```





```
==================================
Escolha uma opção: 5
Exibir Todos:
1 - Pessoa Física
2 - Pessoa Jurídica
Escolha o tipo (1 ou 2): 1
Pessoas Físicas:
ID: 1
Nome: Eduardo
CPF: 11111111111
Idade: 31
==================================
```

```
==================================
Escolha uma opção: 6
Digite o prefixo dos arquivos: eduardo
Dados salvos com sucesso.
==================================
```

```
==================================
Escolha uma opção: 7
Digite o prefixo dos arquivos: eduardo
Dados recuperados com sucesso.
==================================
```

Conclusão:

a) **Qual a importância dos componentes de middleware, como o JDBC?**

São importantes porque facilitam a comunicação entre aplicativos Java e bancos de dados, permitindo que os desenvolvedores se concentrem na lógica do aplicativo, sem se preocuparem com detalhes de conexão com o banco de dados.

b) Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

É melhor que o Statement porque pré-compila consultas, o que pode melhorar o desempenho e proteger contra injeção de SQL ao separar os parâmetros do comando SQL principal.

c) Como o padrão DAO melhora a manutenibilidade do software?

Ajuda a manter o código organizado separando a lógica de acesso a dados em classes específicas, o que facilita a modificação do código relacionado a dados sem afetar outras partes do sistema e promove a reutilização de código.

d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

A herança é implementada usando tabelas de subtipos, onde cada subtipo tem sua própria tabela com uma relação com a tabela principal. Isso reflete a hierarquia de herança e mantém a integridade dos dados.

2º Procedimento | Alimentando a Base

Conclusão:

a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência em arquivo armazena dados localmente, enquanto a persistência em banco de dados permite armazenar e recuperar dados de forma estruturada e mais eficiente, facilitando a consulta e a manipulação.

b) Como o uso de operador *lambda* simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

O uso de operador lambda simplificou a impressão de valores ao permitir uma sintaxe mais concisa e expressiva para a passagem de comportamentos como argumentos de métodos, tornando o código mais legível.

c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como *static*?

Porque o método main é estático e pode ser chamado sem criar uma instância da classe. Métodos estáticos pertencem à classe em vez de instâncias específicas, permitindo o uso sem necessidade de instanciar um objeto.

Observe que os tópicos acima seguem exatamente o que está na Atividade Prática exigida.

Conclusão

O projeto apresenta uma aplicação Java que manipula uma lista de pessoas jurídicas, destacando-se pela simplicidade e foco na funcionalidade principal de atualização de entidades. A função alterar oferece uma abordagem direta e eficaz para modificar informações específicas com base no ID da pessoa jurídica.

A estrutura do projeto é clara e organizada, facilitando a compreensão do código. A utilização de conceitos básicos de manipulação de listas e objetos é uma excelente introdução para programação orientada a objetos.