Disciplina: Nível 1: Iniciando o Caminho Pelo Java
Semestre: 1º - 2024
Aluno: DAVIDSON PEREIRA DE OLIVEIRA
Matrícula: 2022.11.54585-5
Repositório no GIT: dev-davidson/RPG0014---Iniciando-o-caminho-pelo-Java (github.com)

Relatório discente de acompanhamento

Título da Prática: RPG0014 - Iniciando o caminho pelo Java

Objetivo da Prática: Implementação de um cadastro de clientes em modo texto, com persistência em arquivos, baseado na tecnologia Java. 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.

Todos os códigos solicitados neste roteiro de aula

Pessoa.java

```java
import java.io.Serializable;

public class Pessoa implements Serializable {
    private static final long serialVersionUID = 1L;
    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;
    }

```

```
28.     public void setNome(String nome) {
29.         this.nome = nome;
30.     }
31.
32.     public void exibir() {
33.         System.out.println("ID: " + id);
34.         System.out.println("Nome: " + nome);
35.     }
36.}
```

PessoaFisica.java

```
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

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

public class PessoaFisicaRepo {
    private ArrayList<PessoaFisica> pessoas = new ArrayList<>();

    public void inserir(PessoaFisica pessoa) {
        pessoas.add(pessoa);
    }

    public void alterar(PessoaFisica pessoa) {

    }

    public void excluir(int id) {

    }

    public PessoaFisica obter(int id) {

        return null;
    }

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

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

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

PessoaJuridica.java

```java
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

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

public class PessoaJuridicaRepo {
    private ArrayList<Pessoa> pessoas = new ArrayList<>();

    public void inserir(PessoaJuridica pessoa) {
        pessoas.add(pessoa);
    }

    public void alterar(PessoaJuridica pessoa) {
        // Implemente conforme necessário
    }

    public void excluir(int id) {
        // Implemente conforme necessário
    }

    public Pessoa obter(int id) {
        // Implemente conforme necessário
```

```
        return null;
    }

    public ArrayList<Pessoa> obterTodos() {
        return pessoas;
    }

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

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

Análise e Conclusão:

- **Quais as vantagens e desvantagens do uso de herança?**

  Vantagens:

  Reutilização de Código: Permite a reutilização de membros e métodos de uma classe pai nas classes filhas.

  Polimorfismo: Facilita a criação de código mais genérico e flexível, onde objetos de classes filhas podem ser tratados como objetos da classe pai.

  Desvantagens:

  Acoplamento: Pode levar a um acoplamento excessivo entre classes, tornando o código mais difícil de entender e manter.

  Herança Múltipla Problema: Em linguagens que suportam herança múltipla, pode surgir o problema do diamante, onde uma classe é herdada por duas classes, que por sua vez são herdadas por uma terceira classe.

- **Por que a interface Serializable é necessária ao efetuar persistência em arquivos binários?**

  A interface Serializable é necessária para indicar que a classe pode ser serializada, ou seja, seus objetos podem ser convertidos em uma sequência de bytes. Isso é crucial ao persistir objetos em arquivos binários. A serialização permite que os objetos sejam armazenados em um formato que pode ser posteriormente desserializado, reconstruindo os objetos originais. Sem a implementação da interface Serializable, o Java lançará uma exceção NotSerializableException durante a tentativa de serialização.

- **Como o paradigma funcional é utilizado pela API stream no Java?**

  A API Stream do Java permite a utilização de programação funcional através do uso de expressões lambda, operações de ordem superior e pipeline de operações. As operações de Stream permitem realizar operações em conjuntos de dados de maneira mais concisa e expressiva. Algumas características funcionais incluem:

  - ✓ *Operações Map e Filter:* Transformação e filtragem de elementos do Stream.
  - ✓ *Operações Reduce:* Redução dos elementos do Stream a um resultado único.
  - ✓ *Expressões Lambda:* Uso de funções anônimas para definir operações a serem realizadas nos elementos do Stream.

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

  O padrão mais comum adotado na persistência de dados em arquivos em Java é o padrão de projeto DAO (Data Access Object). O DAO é responsável por fornecer uma interface para alguns tipos de fonte de dados, como um banco de dados ou, no caso, arquivos. Isso isola o código de acesso a dados da lógica de negócios, proporcionando maior modularidade e facilitando a manutenção. O DAO define métodos para inserir, atualizar, excluir, e recuperar dados, permitindo uma abstração eficaz da camada de persistência.