Universidade Estácio

Bernardo Oliveira Ramos

Repo: https://github.com/bernardobor/MissaoPratica_Nivel1_Mundo-3

Título da Prática:

Missão Prática | Nível 1 | Mundo 3

Objetivo da Prática:

O objetivo desta prática é implementar um sistema de cadastro de clientes em modo texto, utilizando a linguagem Java, e persistir os dados em arquivos binários. O sistema deverá demonstrar o uso de conceitos de **herança**, **polimorfismo**, e **interface Serializable** para a serialização de objetos, além de permitir a manipulação dos dados (inclusão, alteração, exclusão, exibição e recuperação) de maneira organizada e eficiente.

Resultados da Execução dos Códigos:

Ao executar o código, o menu de opções é exibido. As funcionalidades de inclusão, alteração, exclusão, exibição, salvamento e recuperação foram testadas com sucesso. Abaixo está um exemplo de execução:

css

Copiar código

```
Menu:

1 - Incluir

2 - Alterar

3 - Excluir

4 - Exibir por ID

5 - Exibir todos

6 - Salvar dados

7 - Recuperar dados

0 - Sair
```

Análise e Conclusão:

Análise e Conclusão

Vantagens e desvantagens do uso de herança

- **Vantagens:**
  - **Reutilização de código:** permite que as classes filhas herdem comportamentos e atributos das classes-pai, evitando duplicação de código.
  - **Criação de hierarquias de classes:** facilita a organização e compreensão do sistema, agrupando classes com comportamentos comuns.
  - **Facilidade de entendimento:** sistemas com herança bem implementada tendem a ser mais organizados e fáceis de manter.
- **Desvantagens:**
  - **Forte acoplamento:** classes que herdam de outras podem ficar fortemente dependentes de suas classes-pai, dificultando a manutenção e modificações futuras.
  - **Complexidade:** compreender toda a cadeia de herança pode se tornar desafiador, especialmente em sistemas grandes, aumentando a dificuldade de manutenção e evolução do código.

Por que a interface Serializable é necessária para a persistência em arquivos binários?

A interface `Serializable` é necessária porque permite que o Java **converta (serializar)** um objeto em um formato que pode ser gravado em um arquivo binário e, posteriormente, **recuperar (desserializar)** o objeto para seu estado original. Sem essa interface, o Java não permitiria que objetos complexos fossem gravados diretamente em arquivos binários, limitando a persistência de dados.

Como o paradigma funcional é a base para a API Stream do Java?

A API de **Streams** do Java é baseada no paradigma funcional e permite o uso de operações funcionais como:

- **Expressões lambda**
- **Operações de filtragem**
- **Mapeamento**
- **Redução**

Essas operações possibilitam a manipulação de coleções de maneira declarativa, tornando o código mais **legível**, **eficiente** e fácil de manter.

Qual o padrão de desenvolvimento utilizado para a persistência de dados em arquivos?

O padrão utilizado é o **Data Access Object (DAO)**. Nesse padrão, as operações de persistência e recuperação de dados são encapsuladas em classes específicas, conhecidas como repositórios. Isso **isola a lógica de acesso aos dados**, separando-a do restante do sistema, o que facilita a manutenção e a escalabilidade do código.

Elementos Estáticos e o Método `main`

- **O que são elementos estáticos?**
  - Em Java, os elementos **estáticos** são associados à própria **classe**, e não às instâncias individuais da classe.
  - Eles podem ser acessados diretamente pela classe, sem a necessidade de criar um objeto.
  - Esses membros estáticos, que podem ser métodos ou atributos, são **compartilhados por todas as instâncias** da classe.
- **Por que o método `main` é estático?**
  - O método `main` é o ponto de entrada da aplicação, sendo chamado pela **JVM** antes da criação de qualquer objeto da classe.
  - Como o `main` precisa ser invocado sem a criação de um objeto da classe principal, ele deve ser declarado como **estático**, possibilitando que seja chamado diretamente pela JVM para iniciar o programa.

Utilidade da Classe `Scanner`

- **O que é a classe `Scanner`?**

    - A classe **`Scanner`** é importada da biblioteca `java.util` e usada para fazer **leitura de entrada de dados**, com a entrada padrão do **teclado** sendo a principal utilização.
- **Usos do `Scanner` no projeto:**
    - No contexto do projeto, o `Scanner` é utilizado para capturar a **entrada do usuário** via console, como dados de clientes (nome, CPF, CNPJ, idade) e as **escolhas do menu**.
    - Ele facilita a manipulação de entradas, realizando a **conversão automática** dos dados fornecidos pelo usuário para os tipos adequados (como inteiros, strings, etc.).

Impacto das Classes de Repositório na Organização do Código

- **Separação de responsabilidades:**
    - O uso de **classes de repositório** trouxe uma clara separação de responsabilidades, transferindo toda a lógica de manipulação de dados para **classes dedicadas** como `PessoaFisicaRepo` e `PessoaJuridicaRepo`.
- **Modularidade e coesão:**
    - Isso tornou o código mais **modular**, **coeso**, e de **fácil manutenção**, com cada repositório sendo responsável por operações específicas de armazenamento, alteração, recuperação e exclusão de dados.
- **Encapsulamento:**
    - O encapsulamento das operações em repositórios facilita futuras modificações, permitindo alterações nos mecanismos de persistência sem afetar a lógica do programa principal.