Missão Prática | Nível 1 | Mundo 3

Desenvolvimento Full Stack

RPG0014 - Iniciando o caminho pelo Java

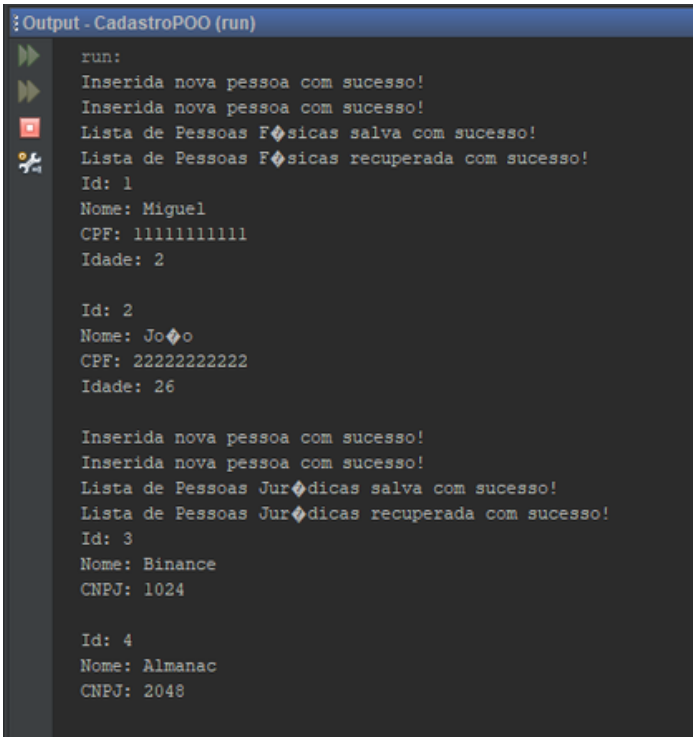
Aluno: Miguel Burali Artioli Firmino

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,
6. utilizando os recursos da programação orientada a objetos e a persistência em arquivos binários.

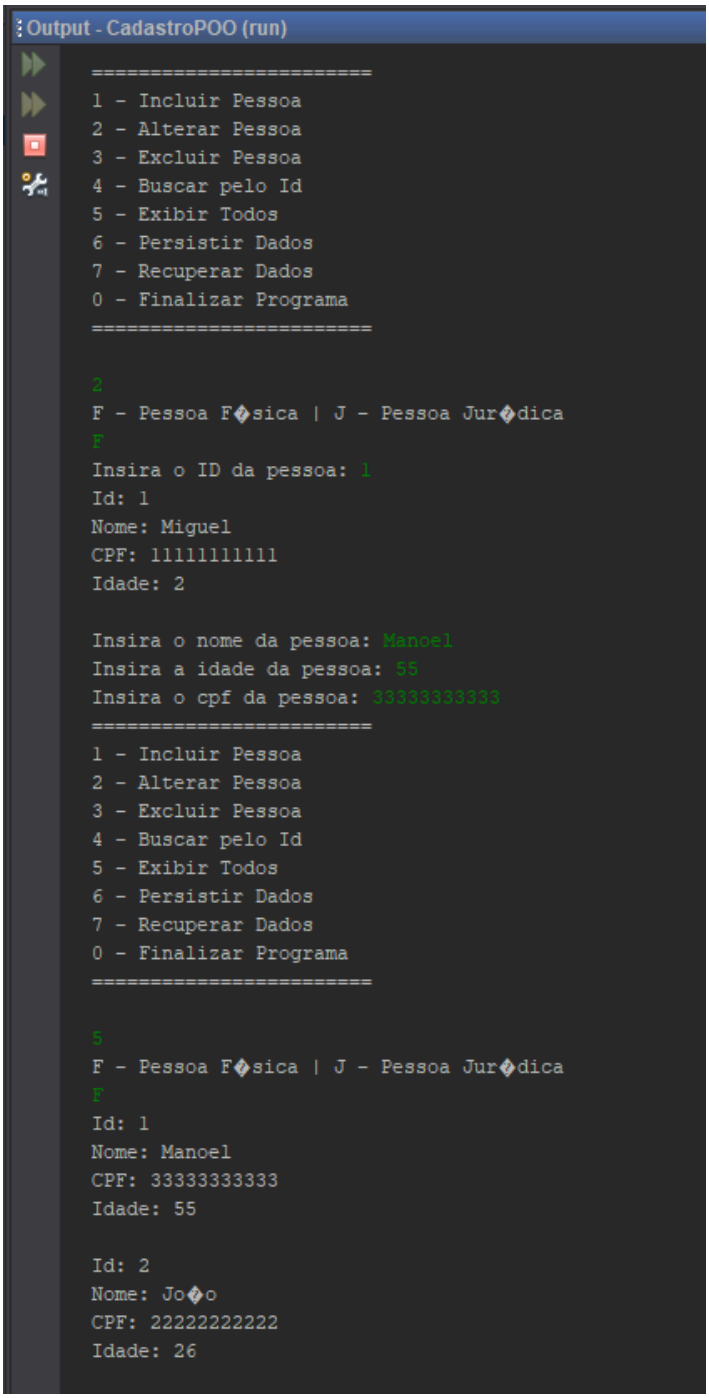
Resultados:

Seguem abaixo os resultados do procedimento 1, foram salvos as pessoas físicas “Miguel” e “João” em repo1, registrados os dados em dados.bin, e recuperados os dados em repo2. O mesmo foi feito com as pessoas jurídicas “Binance” e “Almanac” nos repo3 e repo4.



Resultados:

Seguem abaixo os resultados do procedimento 2, o usuário escolhe alterar a pessoa de Id 1 (Miguel), insere os dados, depois, solicita que sejam exibidos os dados de todas as pessoas físicas, e as mudanças são confirmadas.



Missão Prática | Nível 1 | Mundo 3

Análise e Conclusão

a. Quais as vantagens e desvantagens do uso de herança?

O uso de herança em Java facilita a manutenção do código por meio dos métodos e propriedades definidos na superclasse, permitindo a reutilização desse código. A relação de herança entre objetos também aumenta a flexibilidade do design por meio do polimorfismo, pois as características da superclase podem ser expressos por quaisquer objetos das subclasses. Essas características como um todo promovem maior organização e legibilidade do código.

b. Por que a interface Serializable é necessária ao efetuar persistência em arquivos binários?

A interface Serializable possibilita ao Java a conversão de objetos em sequências de bytes, processo chamado serialização. As classes ObjectOutputStream e ObjectInputStream necessitam que os objetos implementem a interface Serializable.

c. Como o paradigma funcional é utilizado pela API stream no Java?

A API stream pode receber funções como .map(), .filter() para moldar o resultado desejado, essas funções recebem instruções declarativas como "x -> x + 2". Os métodos resultam em novos dados, sem alterar o objeto inicial.

Segue exemplo:

```
List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5, 6);

List<Integer> resultado = numeros.stream()
    .filter(n -> n % 2 == 0)         // remove os números ímpares
    .map(n -> n * 2)                 // dobra o valor dos números
    .collect(Collectors.toList());   // coleta em nova lista

System.out.println(resultado); // o resultado é [4, 8, 12]
System.out.println(numeros);   // os números permanecem iguais
                               // ([1, 2, 3, 4, 5, 6])
```

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

O padrão de desenvolvimento mais utilizado em Java quanto à persistência de dados é o Padrão DAO (Data Access Object), que se baseia na separação entre a lógica de acesso a dados e a lógica de negócio em diferentes classes. Esse padrão visa manter o código mais limpo, organizado e modular.

Missão Prática | Nível 1 | Mundo 3

Análise e Conclusão

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

Os elementos estáticos são inerentes às suas classes, e independem dos objetos da classe, portanto podem ser acessados sem a inicialização de um objeto próprio.
O método main adota esse modificador pois ele é o ponto de início da aplicação, e não pode depender da instanciação de objetos, já que o Java busca-o primeiro para começar o programa.

b. Para que serve a classe Scanner?

A classe Scanner serve para ler (escanear) a entrada de dados vindos do usuário, com ela é possível:

- Ler números inteiros (nextInt())
- Ler números decimais (nextDouble())
- Ler palavras (next())
- Ler linhas inteiras (nextLine())

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

O uso das classes “repo” foi muito útil para separar a lógica do comportamento individual de “pessoas” da lógica das listas de “pessoas”, isso promoveu mais legibilidade ao código bem como facilidade para implementar novas funções.