POLO CENTRO II - MONTENEGRO - RS
NÍVEL 1: INICIANDO O CAMINHO PELO JAVA
TURMA 2023.3 - MUNDO 3
ALUNO: LEANDRO LUIZ LOTTERMANN

Relatório discente de acompanhamento

Objetivos da prática:

- 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.
- No final do projeto, o aluno terá implementado um sistema cadastral em Java,
- utilizando os recursos da programação orientada a objetos e a persistência em
- arquivos binários.

1º Procedimento | Criação das Entidades e Sistema de Persistência

A primeira etapa consistiu em criar as entidades e realizar a persistência dos dados e testar. As instâncias criadas a partir das classes Pessoa, PessoaFisica (extends de Pessoa) e PessoaJuridica (extends de Pessoa) foram armazenadas em uma instancia das classes PessoaJuridicaRepo e PessoaFisicaRepo a qual, pelo método inserir armazenava o objeto em um ArrayList.

```
public class PessoaJuridicaRepo implements Serializable {
    private ArrayList<PessoaJuridica> listaPessoasJuridicas = new ArrayList<>();

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

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

    public void inserir (PessoaFisica pessoaFisica) { this.listaPessoasFisicas.add(pessoaFisica); }
```

Para a realização do teste de persistência, foi instanciado duas classes de repositório de cada categoria (Física e Jurídica). Na primeira eram adicionados dois objetos criados em códico e depois foi chamado o método persistir invocando o nome do arquivo para ser salvo. Então na segunda instância que até agora não tinha nenhum objeto armazenado era executado o método “recuperar” passando o mesmo nome de arquivo. Então o repositório carregava o os objetos armazenados pela outra instância.

```java
public class Main {
    public static void main(String[] args) {
        int contadorIds = 1;

        PessoaFisica leandro = new PessoaFisica(contadorIds,  nome: "Leandro",  cpf: "123.123.123",  idade: 12);
        contadorIds++;
        PessoaFisica claudio = new PessoaFisica(contadorIds,  nome: "Claudio",  cpf: "123.121233.123",  idade: 12);
        contadorIds++;


        PessoaFisicaRepo repo1 = new PessoaFisicaRepo();
        repo1.inserir(leandro);
        repo1.inserir(claudio);


        try {
            repo1.persistir( nomeArquivo: "pessoas.fisica.bin");
        } catch (FileNotFoundException e ) {
            System.out.println(e);
        }



        PessoaFisicaRepo repo2 = new PessoaFisicaRepo();

        try {
            repo2.recuperar( nomeArquivo: "pessoas.fisica.bin");
            repo2.obterTodos().forEach(pessoaFisica -> {
                System.out.println("Id: " + pessoaFisica.getId());
                System.out.println("Nome: " + pessoaFisica.getNome());
                System.out.println("CPF: " + pessoaFisica.getCpf());
                System.out.println("Idade: " + pessoaFisica.getIdade());
                System.out.println("-----------------------");
            });
        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
```

O exemplo mostra apenas a de Pessoa Fisica, mas o mesmo foi feito com a Pessoa Juridica. Veja o resultado no terminal:

```
"C:\Program Files\Java\jdk-17\bin\java.exe"
Dados de Pessoa Física Armazenados.
Dados de Pessoa Física Recuperados.
Id: 1
Nome: Leandro
CPF: 123.123.123
Idade: 12
-----------------------
Id: 2
Nome: Claudio
CPF: 123.121233.123
Idade: 12
-----------------------
Dados de Pessoa Jurídica Armazenados.
Dados de Pessoa Jurídica Recuperados.
Id: 3
Nome: Casas Bahia
CNPJ: 11111111111
-----------------------
Id: 4
Nome: magalu
CNPJ: 11111111111
-----------------------

Process finished with exit code 0
```

Análise e Conclusão:

a. A maior vantagem no uso de herança é o reaproveitamento de códico facilitando muito a manutenção por conta da visibilidade e organização.
b. A interface Serializable é necessária pois indica à JDK que aquela classe pode ser serializada.
c. A API Stream no Java é uma parte fundamental da biblioteca Java.util.stream que introduziu um paradigma funcional para trabalhar com coleções de dados de forma mais concisa e expressiva.
d. Em Java, ao trabalhar com a persistência de dados em arquivos, um padrão comum é o uso do padrão de projeto "DAO" (Data Access Object) .

2º Procedimento | Criação do Cadastro em Modo Texto

Esse procedimento consistiu em criar uma interface para o usuário realizar as ações no programa. No início é apresentado o seguinte menu:

```
===================================
1 - Incluir Pessoa
2 - Alterar Pessoa
3 - Excluir Pessoa
4 - Buscar pelo Id
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
===================================
```

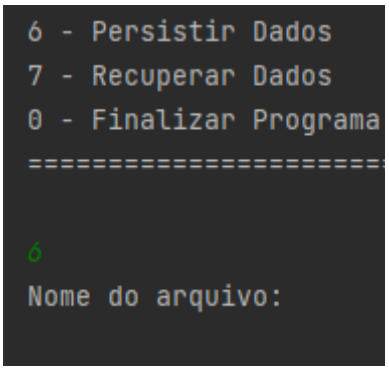
O menu foi feito com um “Do while”, que tem como condição de parada a resposta 0. As ações das opções são realizadas por um switch case.

Nas opções 1, 2 , 3, 4 e 5, é solicitado a escolha entre Pessoa Física e Jurídica.

```
5 - Exibir Todos
6 - Persistir Dados
7 - Recuperar Dados
0 - Finalizar Programa
===================================

5
F - Pessoa Física | J - Pessoa Juridica
```

As opções de persistir e recuperar dados pedem um nome do arquivo que é usado como prefixo da seguinte forma: [prefixo].fisica.bin e [prefixo].juridica.bin.



Todas as funcionalidades foram testadas e estão funcionando de acordo com sua utilidade.

Relatório discente de acompanhamento

a. Elementos estáticos podem ser usados sem a necessidade de instanciar a classe proprietária. Por isso o método main usa esse modificador, pois sem ele nada seria executado.
b. O Scanner serve para observar algo na aplicação e a partir de seus métodos podemos capturar os dados.
c. O uso de classe facilitou muito a organização do códico. O fato de se aproximar da realidade onde cada objeto da vida real possui elementos e realiza certas ações faz com que facilite o pensamento lógico da aplicação.