Estácio

# Missão Prática: Nível 1: Mundo 3

Campus: Messejana
Curso: Desenvolvimento Full Stack
Disciplina: Iniciando o Caminho pelo Java
Semestre letivo: 2024.4
Aluno:Marília Regis

**Objetivos Gerais:**

- Desenvolver um sistema cadastral completo em Java, utilizando os princípios da programação orientada a objetos.
- Implementar a persistência de dados em arquivos binários para garantir a integridade e segurança da informação.
- Demonstrar o uso eficaz da herança e do polimorfismo para modelar entidades e suas relações.
- Aplicar o tratamento de exceções para garantir a robustez do sistema e evitar falhas inesperadas.

**Materiais necessários para a prática**

- JDK e IDE NetBeans.
- Equipamentos: Computador com JDK e NetBeans instalados.

## Roteiro da prática:

## 1º Procedimento | Criação das Entidades e Sistema de Persistência

### 1. Criação do Projeto no NetBeans

- **Novo Projeto:** Crie um novo projeto Java Application no NetBeans e nomeie-o como `nomeCadastroPOO`.
- **Pacote `model`:** Crie um novo pacote com o nome `model` para armazenar as classes de entidades e gerenciadores.

### 2. Criação das Classes de Entidade

- **Classe `Pessoa`:**

JAVA:

```
import java.io.Serializable;

public class Pessoa implements Serializable {
    private int id;
    private String nome;

    // Construtores, getters e setters

    public void exibir() {
        System.out.println("Id: " + id + ", Nome: " + nome);
    }
}
```

- **Classe `PessoaFisica`:**

**Java**:

```
public class PessoaJuridica extends Pessoa implements Serializable {
    private String cnpj;

    // Construtores, getters e setters

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

**3. Criação dos Gerenciadores**

- **Classe `PessoaFisicaRepo`:**

**Java:**

```
import java.io.*;

import java.util.ArrayList;

import java.util.List;

public class PessoaFisicaRepo {

    private List<PessoaFisica> pessoasFisicas = new ArrayList<>();

    // Métodos inserir, alterar, excluir, obter, obterTodos, persistir e recuperar

    // ... (implementação detalhada abaixo)

}
```

- Classe `PessoaJuridicaRepo`: **Similar à classe `PessoaFisicaRepo`**

**4. Implementação dos Métodos dos Gerenciadores**

- Métodos `persistir` e `recuperar`: **Utilizam as classes `ObjectOutputStream` e `ObjectInputStream` para serializar e desserilizar os objetos, respectivamente.**
- Demais métodos: **Realizam as operações de CRUD (Create, Read, Update, Delete) sobre a lista de objetos.**

Exemplo do método `persistir`:

Java:

```
public void persistir(String arquivo) throws IOException {

    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(arquivo))) {

        oos.writeObject(pessoasFisicas);

    }

}
```

**5. Classe Principal**

- Instanciar repositórios: Crie instâncias dos repositórios e adicione objetos.

Java:

```
PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();


PessoaFisica pessoa1 = new PessoaFisica("João da Silva", "12345678910", 30);

PessoaFisica pessoa2 = new PessoaFisica("Maria Souza", "98765432109", 25);


repoFisica.inserir(pessoa1);

repoFisica.inserir(pessoa2);


// Repita o processo para PessoaJuridicaRepo
```

- **Persistir dados: Chame o método `persistir` para salvar os dados em um arquivo.**

**Java**:

```
String arquivoFisica = "pessoas_fisicas.bin";

repoFisica.persistir(arquivoFisica);


// ... similar para PessoaJuridicaRepo
```

- **Recuperar dados: Chame o método `recuperar` para carregar os dados do arquivo e exibi-los.**

**Java:**

```
PessoaFisicaRepo repoFisicaRecuperado = new PessoaFisicaRepo();

repoFisicaRecuperado.recuperar(arquivoFisica);


// Exibir os dados:

for (PessoaFisica pessoa : repoFisicaRecuperado.obterTodos()) {

    pessoa.exibir();

}


// ... similar para PessoaJuridicaRepo
```

**Análise e Conclusão:**

**Herança: Vantagens e Desvantagens**

- **Vantagens:**
  - **Reutilização de código:** Permite criar classes mais específicas a partir de classes mais genéricas, evitando a duplicação de código.
  - **Hierarquia de classes:** Organiza o código de forma mais clara e intuitiva, facilitando a compreensão do sistema.
  - **Polimorfismo:** Permite que objetos de classes diferentes sejam tratados como se fossem de uma classe pai comum, aumentando a flexibilidade do código.
- **Desvantagens:**
  - **Acoplamento:** A herança cria um forte acoplamento entre classes, o que pode dificultar a manutenção e a reutilização de código em diferentes contextos.
  - **Rigidez:** Modificações em uma classe base podem afetar todas as suas subclasses, tornando o código mais frágil a mudanças.
  - **Complexidade:** A hierarquia de classes pode se tornar complexa e difícil de entender, especialmente em sistemas grandes.

**Interface Serializable e Persistência em Arquivos Binários**

A interface `Serializable` indica ao mecanismo de serialização que os objetos de uma classe podem ser convertidos em um fluxo de bytes e, posteriormente, restaurados a partir desse fluxo. **É fundamental para a persistência em arquivos binários porque:**

- **Conversão de objetos em bytes:** A serialização transforma os objetos em uma representação binária, permitindo que sejam armazenados em um arquivo.
- **Restauração de objetos:** A desserialização reconstrói os objetos a partir dos dados armazenados no arquivo.
- **Persistência:** Ao implementar `Serializable`, você permite que os objetos sejam persistidos em um estado consistente, podendo ser recuperados posteriormente.

**Paradigma Funcional e API Stream no Java**

A API Stream no Java introduz um estilo de programação funcional, permitindo realizar operações em coleções de dados de forma concisa e expressiva. **Algumas características do paradigma funcional exploradas pela API Stream:**

- **Imutabilidade:** As operações em streams geralmente não modificam a coleção original, gerando novos resultados.

- **Mapeamento:** Transforma cada elemento de uma coleção em outro elemento.
- **Filtragem:** Retorna uma nova coleção com os elementos que satisfazem um determinado critério.
- **Redução:** Combina todos os elementos de uma coleção em um único valor.
- **Laziness:** As operações em streams são executadas somente quando o resultado final é solicitado, otimizando o desempenho.

**Padrões de Desenvolvimento para Persistência em Arquivos em Java**

Não existe um padrão de desenvolvimento único para persistência em arquivos em Java. A escolha do padrão depende da complexidade do sistema, das necessidades específicas e das preferências do desenvolvedor. **Alguns padrões comuns incluem:**

- **DAO (Data Access Object):** Encapsula o acesso aos dados, separando a lógica de persistência da lógica de negócio.
- **Repository:** Similar ao DAO, mas com um foco maior em operações de leitura e escrita.
- **ActiveRecord:** Combina a representação de objetos com o mecanismo de persistência, simplificando o mapeamento objeto-relacional.

**Outros aspectos a considerar:**

- **Serialização:** Para objetos simples, a serialização padrão do Java pode ser suficiente. Para objetos mais complexos, frameworks como Jackson ou Gson podem oferecer funcionalidades adicionais.
- **Formatos de arquivo:** Além de arquivos binários, você pode utilizar outros formatos como JSON ou XML para armazenar dados.
- **Bibliotecas e frameworks:** Existem diversas bibliotecas e frameworks que facilitam a persistência de dados, como Hibernate, Spring Data JPA e MyBatis.

**Em resumo,** a escolha do padrão de desenvolvimento e das ferramentas adequadas depende das necessidades específicas do projeto. É importante avaliar os prós e contras de cada abordagem e escolher a que melhor se adapta ao seu contexto.

## 2º Procedimento | Criação do Cadastro em Modo Texto

### Implementando um Menu de Texto para o Cadastro

Vamos criar um menu de texto interativo para o seu sistema de cadastro, permitindo ao usuário realizar as operações de inclusão, alteração, exclusão, consulta e persistência de dados.

- **Estrutura do Código:**

**Java:**

```
import java.util.Scanner;

import java.io.IOException;


// ... (resto das suas classes)


public class Main {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        PessoaFisicaRepo repoFisica = new PessoaFisicaRepo();

        PessoaJuridicaRepo repoJuridica = new PessoaJuridicaRepo();


        int opcao;

        do {

            System.out.println("1 - Incluir\n2 - Alterar\n3 - Excluir\n4 - Obter por ID\n5 - Obter todos\n6 - Salvar dados\n7 - Recuperar dados\n0 - Sair");

            System.out.print("Opção: ");

            opcao = scanner.nextInt();
```

```
            switch (opcao) {
                case 1:
                    // ... (implementação da opção incluir)
                    break;
                case 2:
                    // ... (implementação da opção alterar)
                    break;
                // ... (implementação das demais opções)
                case 0:
                    System.out.println("Saindo...");
                    break;
                default:
                    System.out.println("Opção inválida.");
            }
        } while (opcao != 0);
    }
}
```

- Implementação Detalhada das Opções:

- Incluir: Perguntar se é pessoa física ou jurídica, solicitar os dados e adicionar ao repositório correspondente.
- Alterar: Perguntar o tipo, solicitar o ID, apresentar os dados atuais, solicitar os novos dados e atualizar o repositório.
- Excluir: Perguntar o tipo, solicitar o ID e remover do repositório.
- Obter por ID: Perguntar o tipo, solicitar o ID e exibir os dados da entidade encontrada.

- Obter todos: Perguntar o tipo e exibir os dados de todas as entidades do repositório.
- Salvar dados: Solicitar o prefixo, chamar os métodos `persistir` dos repositórios com os nomes de arquivo correspondentes.
- Recuperar dados: Solicitar o prefixo, chamar os métodos `recuperar` dos repositórios com os nomes de arquivo correspondentes.

Exemplo da Opção Incluir:

Java:

```
case 1:
    System.out.print("Pessoa Física (F) ou Jurídica (J)? ");
    char tipo = scanner.next().charAt(0);
    if (tipo == 'F') {
        // Criar uma nova PessoaFisica e adicionar ao repoFisica
    } else if (tipo == 'J') {
        // Criar uma nova PessoaJuridica e adicionar ao repoJuridica
    } else {
        System.out.println("Tipo inválido.");
    }
    break;
```

Tratamento de Exceções:

```
case 6:
    System.out.print("Digite o prefixo dos arquivos: ");
    String prefixo = scanner.next();
```

```
try {
    repoFisica.persistir(prefixo + ".fisica.bin");
    repoJuridica.persistir(prefixo + ".juridica.bin");
    System.out.println("Dados salvos com sucesso.");
} catch (IOException e) {
    System.out.println("Erro ao salvar os dados: " + e.getMessage());
}
break;
```

## Conclusão:

### Elementos Estáticos

**Elementos estáticos** em Java são aqueles que pertencem à classe e não a um objeto específico daquela classe. Isso significa que eles são compartilhados por todas as instâncias da classe e podem ser acessados diretamente pelo nome da classe, sem a necessidade de criar um objeto.

### Por que o método main é estático?

- **Ponto de entrada da aplicação:** O método `main` é o ponto de partida da execução de um programa Java. Ele precisa ser executado antes mesmo de qualquer objeto ser criado, por isso é declarado como `static`.
- **Acesso direto:** Ao ser estático, o método `main` pode acessar outros membros estáticos da classe, como variáveis e métodos, sem a necessidade de criar uma instância.
- **Convenção:** É a forma padrão de definir o método principal em Java, seguindo as convenções da linguagem.

### A Classe Scanner

A classe `Scanner` é uma ferramenta útil para ler dados de entrada do usuário, como o teclado. Ela permite que você capture diferentes tipos de dados, como inteiros, números de ponto flutuante, strings, etc. Com ela, você pode criar programas interativos que solicitam informações ao usuário e processam essas informações.

### Para que serve?

- **Leitura de dados do teclado:** É a principal função da classe Scanner. Ela fornece métodos como `nextInt()`, `nextDouble()`, `nextLine()`, entre outros, para ler diferentes tipos de dados.
- **Flexibilidade:** Permite personalizar a leitura de dados, como definir delimitadores, pular linhas em branco, etc.

### Classes de Repositório e a Organização do Código

**Classes de repositório** são responsáveis por gerenciar o acesso a dados, seja em um banco de dados, em um arquivo ou em qualquer outra fonte de dados. Ao utilizar classes de repositório, você obtém os seguintes benefícios:

- **Encapsulamento:** A lógica de acesso aos dados é encapsulada em uma única classe, o que torna o código mais organizado e fácil de manter.
- **Reutilização:** As classes de repositório podem ser reutilizadas em diferentes partes do seu programa, evitando a duplicação de código.
- **Testes:** É mais fácil escrever testes unitários para as classes de repositório, pois elas possuem uma interface bem definida.
- **Abstração:** As classes de repositório abstraem a forma como os dados são armazenados, permitindo que você altere a implementação sem afetar o resto do seu código.

**Em resumo:**

- **Elementos estáticos:** São compartilhados por todas as instâncias de uma classe e podem ser acessados diretamente.
- **Método main:** É o ponto de entrada da aplicação e é declarado como estático por convenção e necessidade.
- **Classe Scanner:** Permite ler dados de entrada do usuário de forma flexível.
- **Classes de repositório:** Organizam o acesso aos dados, facilitando a manutenção e a testabilidade do código.