



**Campus:** Polo Ingleses

**Curso:** Desenvolvimento Full Stack

**Disciplina:** Nível 3 - Back-end Sem Banco Não Tem

**Turma:** 9001

**Semestre:** 23.3

**Aluna:** Maria Carolina Knudsen Boabaid

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

**Objetivo da prática:** Conectar o bando de dados 'Cadastro.db' com o nosso projeto JAVA, e testar se a conexão foi exitosa com uma carga inicial de dados inseridas na classe testes.

**Resultados da execução dos códigos:**

The screenshot shows an IDE with the following components:

- EXPLORADOR (Left):** Project structure showing folders like 'database', 'src', 'test', and 'target'. The file 'CadastroDbApplicationTests.java' is selected.
- Editor (Center):** Displays the code for 'CadastroDbApplicationTests.java'. The code includes a static void method 'testPessoaFisicaOperations()' that creates a 'PessoaFisica' object, saves it to the database, and prints details.
- TERMINAL (Bottom):** Shows the output of the test execution. It includes the command to run the test and the resulting output, which lists the details of the created person: 'Joao', 'Logradouro: Rua 11, Centro', 'Estado: PA', 'Telefone: 1111-1111', 'Email: joao@riacho.com', 'CPF: 111111111111'.

```
private static void testPessoaFisicaOperations() {
    PessoaFisicaDAO pessoaFisicaDAO = new PessoaFisicaDAO();

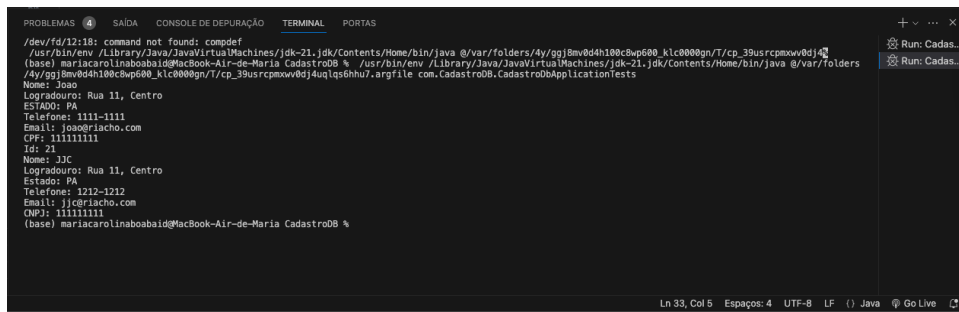
    PessoaFisica pessoaFisica = new PessoaFisica();
    pessoaFisica.criarNome(nome:"Joao");
    pessoaFisica.criarLogradouro(logradouro:"Rua 11, Centro");
    pessoaFisica.criarCidade(cidade:"Rio de Janeiro");
    pessoaFisica.criarEstado(estado:"PA");
    pessoaFisica.criarTelefone(telefone:"1111-1111");
    pessoaFisica.criarEmail(email:"joao@riacho.com");
    pessoaFisica.criarCpf(cpf:"111111111111");
    pessoaFisicaDAO.adicionar(pessoaFisica);

    if (pessoaFisica.pegarId() != null) {
        pessoaFisica.criarNome(nome:"Novo Nome");
        pessoaFisicaDAO.alterar(pessoaFisica);

        System.out.println("Pessoas Fisicas registradas");
        pessoaFisicaDAO.obterTodos().forEach(System.out::println);

        pessoaFisicaDAO.excluir(pessoaFisica.pegarId());
    } else {
        System.out.println("Falha ao persistir Pessoa Fisica no banco.");
    }
}
```

```
/dev/fd/12:18: command not found: compdef
/usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java @/var/folders/4y/gg18wv04h100cwp600_klc0000gn/T/cp_39usrpcmwv0djq4uqlq6hu7.argfile com.CadastroDB.CadastroDbApplicationTests
(base) mariacarinaboabaid@MacBook-Air-de-Maria CadastroDB % /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java @/var/folders/4y/gg18wv04h100cwp600_klc0000gn/T/cp_39usrpcmwv0djq4uqlq6hu7.argfile com.CadastroDB.CadastroDbApplicationTests
None: Joao
Logradouro: Rua 11, Centro
ESTADO: PA
Telefone: 1111-1111
Email: joao@riacho.com
CPF: 111111111111
Id: 21
None: JJC
Logradouro: Rua 11, Centro
Estado: PA
Telefone: 1212-1212
Email: jjc@riacho.com
CPF: 111111111111
(base) mariacarinaboabaid@MacBook-Air-de-Maria CadastroDB %
```



```
/dev/fd/12:18: command not found: compdef
/usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java @/var/folders/4y/ggjm0d4h100c8wp600_klc0000gn/T/cp_39usrpcmxwv0djq4
(base) mariacarinaboabaid@MacBook-Air-de-Maria-Cadastro08 % /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java @/var/folders
/4y/ggjm0d4h100c8wp600_klc0000gn/T/cp_39usrpcmxwv0djq4uqls6hu7.argfile com.Cadastro08.Cadastro08ApplicationTests
Nome: Joao
Logradouro: Rua 11, Centro
Estado: PA
Telefones: 1111-1111
Email: joao@riacho.com
CPF: 111111111
Id: 21
Nome: JJC
Logradouro: Rua 11, Centro
Estado: PA
Telefones: 1212-1212
Email: jjc@riacho.com
CPF: 111111111
(base) mariacarinaboabaid@MacBook-Air-de-Maria-Cadastro08 %
```

## Análise e Conclusão:

### a) Qual a importância dos componentes de middleware, como o JDBC?

Os componentes de middleware, como o JDBC, são essenciais para o desenvolvimento de aplicações. Eles atuam como uma camada intermediária entre o código da aplicação e os recursos subjacentes, simplificando a interação entre ambos e promovendo a modularidade, portabilidade e eficiência no desenvolvimento de software.

Os componentes de middleware são softwares que fornecem serviços compartilhados a aplicações. No caso do JDBC, ele fornece uma interface uniforme para o acesso a dados em bancos de dados. Isso simplifica a complexidade associada ao acesso e manipulação de dados, permitindo que os desenvolvedores foquem na lógica da aplicação.

Além disso, o JDBC contribui para a portabilidade das aplicações. Isso significa que o mesmo código Java pode ser usado com diferentes bancos de dados, contanto que haja suporte JDBC para esses sistemas. Isso promove flexibilidade e facilita a manutenção do código ao longo do tempo.

### b) Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?

O Statement e o PreparedStatement são duas interfaces da API JDBC utilizadas para executar consultas SQL. A principal diferença entre elas está na forma como lidam com parâmetros.

O Statement é utilizado para executar consultas SQL simples, mas não lida eficientemente com parâmetros. Isso pode resultar em vulnerabilidades de segurança, como injeção de SQL. O PreparedStatement é mais seguro e eficiente quando se trabalha com consultas parametrizadas. Ele permite a definição de parâmetros de forma segura, evitando a injeção de

SQL, e melhora o desempenho por meio do uso de consultas preparadas, que podem ser reutilizadas com diferentes valores de parâmetros.

**c) Como o padrão DAO melhora a manutenibilidade do software?**

O padrão DAO (Data Access Object) é um padrão de projeto de software que fornece uma camada de abstração entre a lógica de negócios da aplicação e o acesso aos dados. Isso permite que a lógica de negócios da aplicação seja isolada das mudanças no armazenamento de dados, tornando o software mais manutenível.

O DAO encapsula as operações de acesso a dados em objetos específicos. Isso significa que a lógica de negócios da aplicação não precisa lidar diretamente com a API do banco de dados. Em vez disso, a aplicação pode simplesmente chamar os métodos do DAO para acessar os dados.

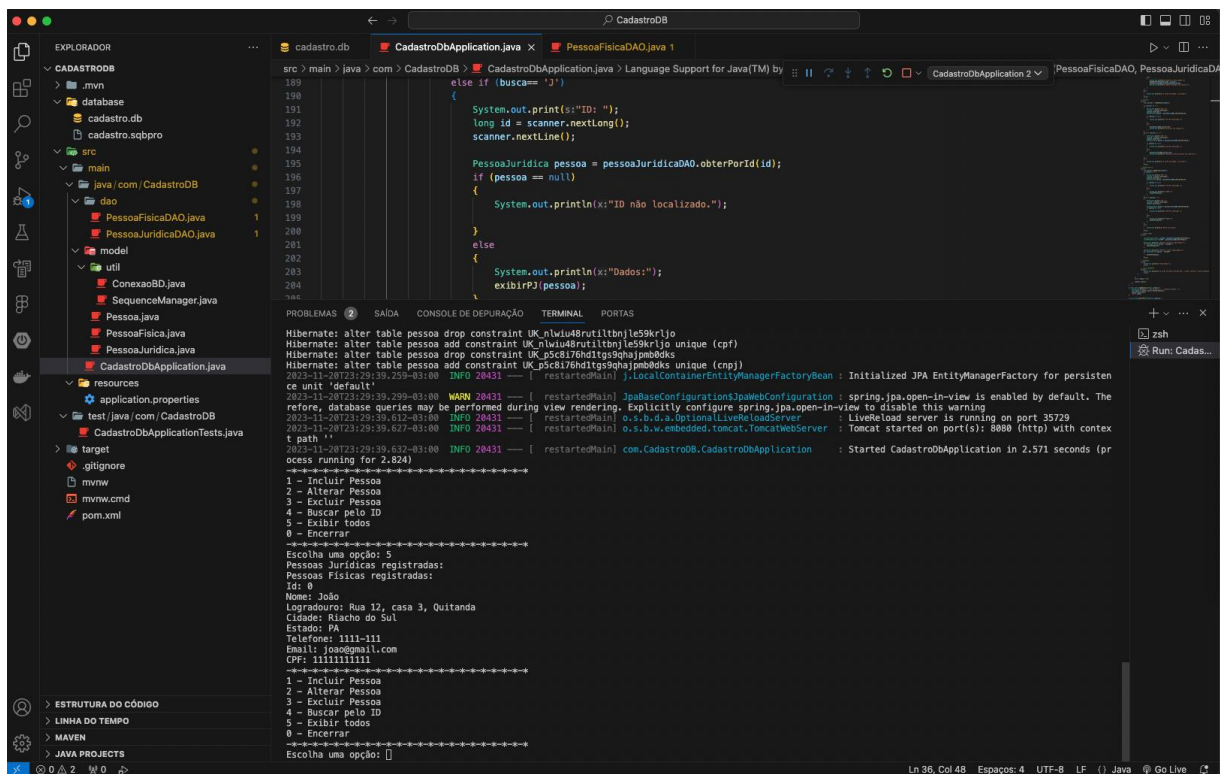
**d) Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?**

Quando lidamos com um modelo estritamente relacional no banco de dados e utilizamos herança em nossa modelagem, ela é frequentemente refletida através de tabelas. Cada classe na hierarquia de herança é representada por uma tabela separada, e as relações entre essas tabelas espelham a estrutura de herança no modelo de dados.

## **2º Procedimento | Alimentando a Base**

**Objetivo da prática:** O objetivo é gerar os métodos CRUD no JAVA com a conexão de banco de dados.

**Resultados da execução dos códigos:**



## Análise e Conclusão:

### a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência em arquivo e a persistência em banco de dados são duas abordagens distintas para armazenar e recuperar dados em um sistema. A principal diferença reside no local onde as informações são mantidas e na estrutura utilizada para organizá-las.

Na persistência em arquivo, os dados são armazenados em arquivos do sistema de arquivos. Cada entidade ou conjunto de dados pode ser representado por um arquivo separado, e o acesso aos dados é feito diretamente nos arquivos. Essa abordagem é simples e adequada para volumes pequenos de dados, mas pode tornar-se menos eficiente e mais suscetível a problemas de concorrência em escalas maiores.

Por outro lado, na persistência em banco de dados, os dados são organizados em tabelas dentro de um sistema de gerenciamento de banco de dados (SGBD). O acesso aos dados é realizado por meio de consultas SQL, e a estrutura de banco de dados oferece uma maneira mais organizada e eficiente de gerenciar grandes conjuntos de informações. Além disso, os

SGBDs proporcionam funcionalidades como transações, controle de concorrência e integridade referencial.

**b) Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?**

O uso de operadores lambda nas versões mais recentes do Java simplificou significativamente a tarefa de imprimir os valores contidos nas entidades. Anteriormente, para realizar essa operação, era necessário escrever um bloco de código mais extenso utilizando as versões tradicionais do Java. Com a introdução de operadores lambda, a sintaxe tornou-se mais concisa, permitindo expressar a lógica de impressão de forma mais direta e compacta.

Esses operadores possibilitam a definição de funções anônimas de maneira mais sucinta, eliminando a necessidade de criar classes ou métodos adicionais apenas para tarefas simples como a impressão de valores. Ao adotar operadores lambda, o código torna-se mais legível e eficiente, proporcionando uma abordagem mais moderna e expressiva para operações como a impressão de dados contidos nas entidades em Java.

**c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static?**

Métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como static devido à natureza estática do método main em Java. O método main é o ponto de entrada da execução do programa e pertence à classe, não a instâncias específicas da classe. Portanto, para que métodos possam ser chamados diretamente a partir do método main, sem a criação de uma instância da classe, eles devem ser declarados como static.

A marcação como static indica que o método pertence à classe em si e não requer uma instância específica para ser invocado. Isso é essencial para garantir que o método main possa ser chamado sem a necessidade de criar um objeto da classe principal.