CURSO DESENVOLVIMENTO FULL STACK
POLO JACAREPAGUÁ - RIO DE JANEIRO – RJ

POLO JACAREPAGUÁ – RIO DE JANEIRO- RJ

UNIVERSIDADE ESTÁCIO DE SÁ

Missão Prática | Nível 3 | Mundo 3

Curso: Desenvolvimento FullStack

Disciplina Nível 3: RPG0015 – BackEnd sem banco não tem

Número da Turma: 2024.2

Semestre Letivo: Mundo-3

Aluno: Gilberto Ferreira da Silva Junior

Matrícula: 2023 0701 4923

URL GITHUB:

https://github.com/GilbertoFSJunior/RPG0016-BackEndSemBancoNaoTem.git

1 º Título da Prática: BackEnd sem banco não tem

1 - Desenvolvimento de um Sistema de Cadastro de Pessoas com Conexão ao Banco de Dados utilizando JDBC e Padrão DAO.

2º Objetivo da Prática:

1- Implementar persistência com base no middleware JDBC.

2 - Utilizar o padrão DAO (Data Access Object) no manuseio de dados.

3 - Implementar o mapeamento objeto-relacional em sistemas Java.

4 - Criar sistemas cadastrais com persistência em banco relacional.

5 - No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

1 º Procedimento | Mapeamento Objeto-Relacional e DAO

Criando o projeto e configurando as bibliotecas que serão utilizadas.

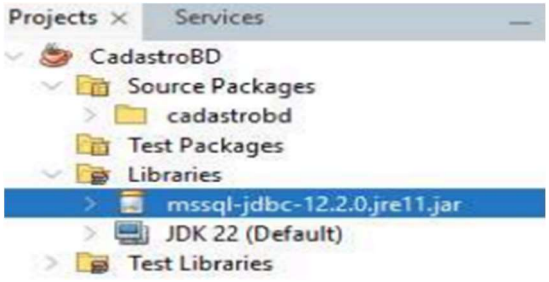

Imagem1: IDE NetBeans

Configurando o acesso ao banco de dados por meio da janela de Serviços do NetBeans.

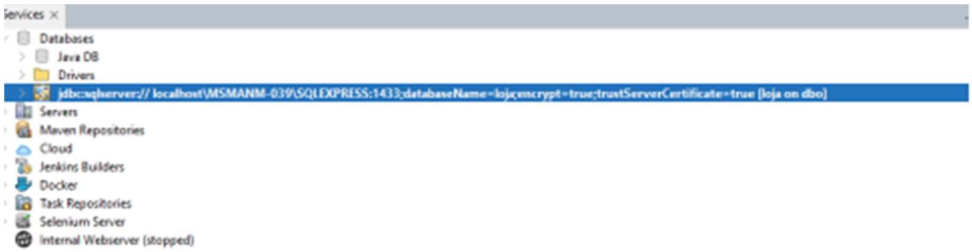

Imagem 2: IDE Netbeans

De volta ao projeto, faremos a criação do pacote, neste caso, **cadastrobd.model**, e dentro do mesmo criar também as classes abaixo.

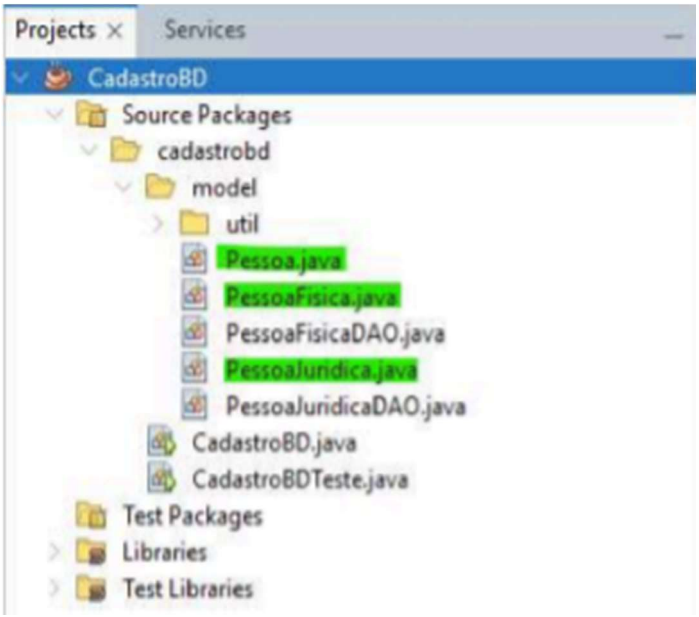

Imagen 3: Pacote cadastrobd e classes: Pessoa.java, PessoaFisica.java e PessoaJuridica.java

Criação do pacote **cadastr.model.util** para inclusão das classes (utilitárias) conforme abaixo.

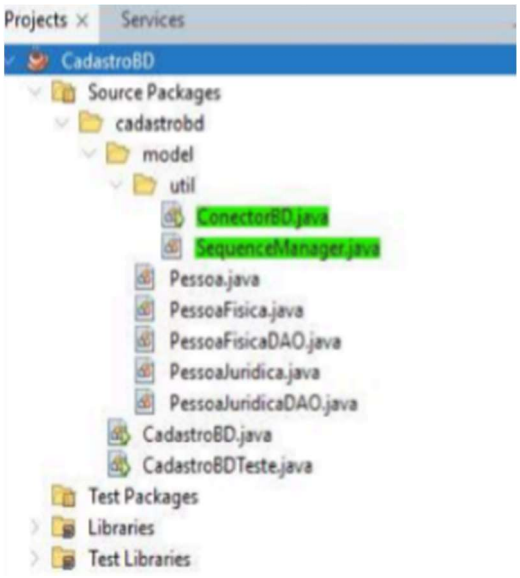

Imagem 4: pacote cadastro.model.util e classes utilitárias: ConectorBD.java e SequenceManager.java

Codificando as classes em padrão DAO, para o pacote **cadastro.model**.

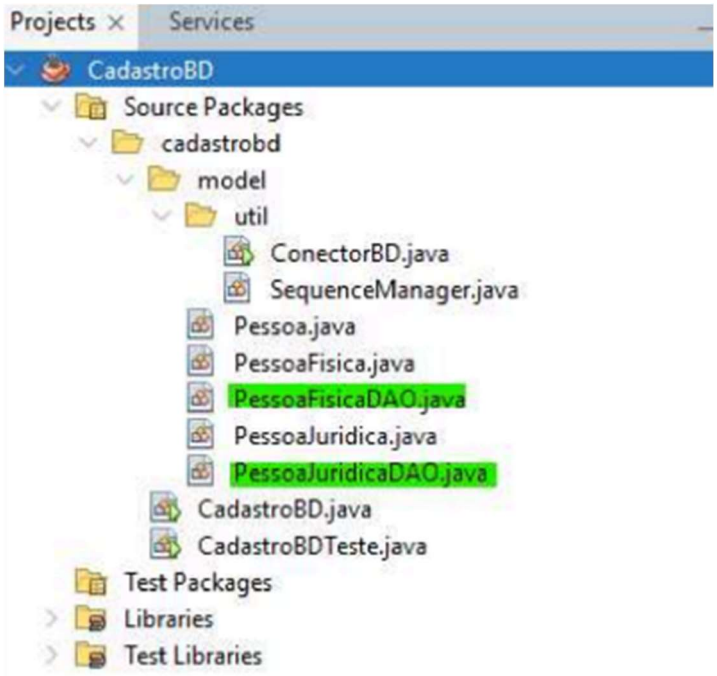

Imagem 5: classes padrão DAO, PessoaFisicaDAO.java e PessoaJuridicaDAO.java

Criação de uma classe principal (de onde o projeto inicia sua execução), nomeada **cadastroBDTeste**, que efetuará as operações do método Main.

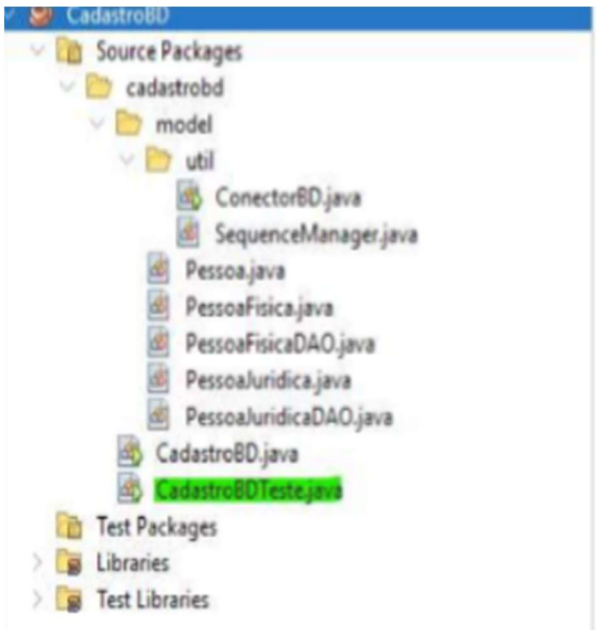

Imagem 6: Classe cadastroBDTeste, contendo o método main.

A e E – Inserindo uma Pessoa Física/Pessoa Jurídica e persistindo as informações no banco de dados.

B e F – Alterando os dados da pessoa física/ pessoa jurídica no banco de dados.

C e G – Consultando as pessoas físicas/pessoas jurídicas do banco de dados e listando no console.

D e H – Exclusão de Pessoa Física/Pessoa Jurídica incluída anteriormente no Banco de Dados.

a) Importância dos Componentes de Middleware como o JDBC

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham uma função essencial na arquitetura de sistemas de software, especialmente em aplicações empresariais e web. Abaixo os principais motivos que ressaltam essa importância:

- **Abstração**: Simplifica o desenvolvimento ao ocultar a complexidade da infraestrutura subjacente, permitindo que os desenvolvedores foquem na lógica de negócio.
- **Reusabilidade**: Oferece componentes pré-desenvolvidos que podem ser reutilizados em diferentes projetos, acelerando o ritmo de desenvolvimento.
- **Interoperabilidade**: Facilita a integração entre diferentes sistemas e tecnologias, promovendo a heterogeneidade.
- **Escalabilidade**: Auxilia na expansão das aplicações de forma fácil e eficiente.
- **Gerenciamento**: fornece ferramentas para monitorar e gerenciar o desempenho da aplicação.

Benefícios Específicos do JDBC:

- **Portabilidade**: Aplicações Java com JDBC podem ser executadas em múltiplas plataformas e bancos de dados sem grandes alterações.
- **Produtividade**: Aumenta a eficiência dos desenvolvedores ao eliminar a necessidade de escrever SQL específico para cada banco de dados.
- **Segurança**: Disponibiliza mecanismos de proteção de dados, como autenticação e autorização.
- **Transações**: Possibilidade de transações atômicas, garantindo a integridade dos dados.

Em Resumo: O JDBC e outros componentes de middleware são cruciais para criar aplicações escaláveis, seguras e de fácil manutenção, garantindo comunicação eficiente com sistemas de banco de dados.

---

b) Diferença no Uso de Statement e PreparedStatement para Manipulação de Dados

As interfaces **Statement e PreparedStatement** do JDBC são utilizadas para executar comandos SQL em um banco de dados, mas possuem diferenças importantes:

Declaração

- **Criação**: Um novo objeto **Statement** é criado e enviado ao banco de dados para cada execução de comando SQL.

- **Parâmetros**: Parâmetros são concatenados diretamente na string SQL, podendo gerar riscos de segurança (injeção de SQL) e problemas de desempenho.
- **Compilação**: O comando SQL é compilado toda vez que é executado.

Declaração preparada

- **Criação**: Um único objeto **PreparedStatement** é criado para um comando SQL específico, podendo ser reutilizado com diferentes valores de parâmetros.
- **Parâmetros**: Os parâmetros são tratados como valores, melhorando a segurança contra injeção de SQL.
- **Compilação**: O comando SQL é compilado apenas uma vez, otimizando o desempenho, especialmente em consultas repetitivas.

Resumo:

- **Statement** é adequado para consultas SQL simples e situações onde segurança e desempenho não são as maiores preocupações.
- **PreparedStatement** é recomendado para consultas mais complexas e situações onde segurança e desempenho são cruciais. A reutilização de comandos e a gestão de parâmetros são facilitadas, além de melhorar a legibilidade do código.

---

c) Melhorias na Manutenibilidade do Software com o Padrão DAO

O padrão **DAO (Data Access Object)** é uma abordagem excepcional para aumentar a manutenibilidade do software, especialmente em sistemas que ocorrem de acesso a bancos de dados.

1. **Separação de Responsabilidades**:
   - **Lógica de Negócio x Acesso a Dados**: O DAO encapsula todas as operações de acesso ao banco em uma camada isolada, permitindo que a lógica de negócio não dependa de como os dados são armazenados.
   - **Facilita Testes**: Com o acesso a dados isolados, a lógica de negócios pode ser testada de forma independente, usando mocks ou stubs para simular o comportamento do banco de dados.
2. **Abstração**:
   - **Independência do Banco de Dados**: O DAO cria uma camada de abstração, possibilitando mudanças no banco de dados sem impacto na lógica de negócios.
   - **Facilidade de Mudanças**: Alterações na estrutura do banco de dados podem ou no armazenamento dos dados ser realizadas no DAO, sem afetar o restante da aplicação.
3. **Reutilização**:
   - **Componentes reutilizáveis**: DAOs podem ser reutilizados em várias partes do sistema, reduzindo a duplicação de código e aumentando a consistência.
4. **Facilidade de Manutenção**:
   - **Localização de Erros**: Uma lógica de acesso a dados isolados facilita a identificação e correção de erros.
   - **Menor Impacto de Mudanças**: Alterações na estrutura do banco impactam menos a aplicação como um todo.

Em Resumo : O padrão DAO facilita a manutenção ao promover uma estrutura organizada para acesso a dados, separação de responsabilidades, testabilidade, reutilização e facilidade de manutenção.

---

d) Modelagem de Herança em Bancos de Dados Relacionais

Em bancos de dados relacionais, o conceito de herança não possui um mapeamento direto devido à estrutura tabular, que não é hierárquica como classes orientadas a objetos.

Desafios:

- **Modelagem de Dados**: Uma herança capturada de forma natural e relações de tipo-subtipo, que podem ser complexas de modelar em bancos relacionais.
- **Consultas**: Consultas envolvidas em sessões podem se tornar complexas e menos eficientes.

Estratégias para Modelagem de Herança:

1. **Tabela por Subtipo**: Cada subtipo possui uma tabela própria, com chave estrangeira para a tabela da superclasse.
    - **Vantagens**: Facilidade e desempenho em consultas específicas de subtipos.
2. **Tabela Única com Coluna Discriminadora**: Uma tabela armazena todos os objetos, com uma coluna decrescente o tipo.
    - **Vantagens**: Simplicidade e desempenho em consultas genéricas.
    - **Desvantagens**: Muitos valores nulos se os subtipos apresentam diferenças marcantes.
3. **Tabela Principal + Tabelas Filhas**: Tabela principal com atributos comuns, e tabelas filhas com atributos específicos.
    - **Vantagens**: Boa normalização e flexibilidade para novos subtipos.
    - **Desvantagens**: Consultas complexas devido a junções.
4. **Herança Total**: Cada subtipo possui uma tabela com todos os atributos, inclusive os herdados.
    - **Vantagens**: Simplicidade nas consultas.
    - **Desvantagens**: redundância de dados.
5. **Mapeamento Objeto-Relacional (ORM)**: Frameworks ORM, como Hibernate, abstraem o mapeamento, tornando a modelagem de herança mais natural.
    - **Vantagens**: Desenvolvimento simplificado.
    - **Desvantagens**: Possíveis perdas de desempenho.

Escolha da Melhor Abordagem: A escolha depende de fatores como frequência de consultas, número de subtipos, requisitos de desempenho e complexidade da classificação.

Resumo: Modelar herança em bancos de dados relacionais é desafio e exige planejamento. Frameworks ORM podem simplificar, mas é importante avaliar as implicações de cada abordagem.

2º Procedimento: Alimentando a Base de Dados

1. **Alteração do Método main** :
   - Apresentar ao usuário as opções de operação no programa:
     - 1: Incluir
     - 2: Alterar
     - 3: Excluir
     - 4: Exibir por ID
     - 5: Exibir todos
     - 0: Finalizar a execução
2. **Detalhes das Opções**:
   - **Incluir**: Escolher entre Pessoa Física ou Jurídica, inserir dados via teclado e adicionar ao banco de dados.
   - **Alterar**: Escolher o tipo de pessoa, fornecer ID e atualizar dados no banco de dados.
   - **Excluir**: Escolher o tipo de pessoa, fornecer ID e remover o banco de dados.
   - **Obter**: Escolher o tipo de pessoa, fornecer ID e exibir dados do banco.

Concluindo: Este relatório sintetiza a importância e os componentes de middleware, o padrão DAO e a modelagem de aplicação herdada, destacando as principais práticas para implementação e manutenção de sistemas em uma arquitetura Java conectada a banco de dados.