POLO CHAPADA - MANAUS - AM/UNIVERSIDADE ESTÁCIO DE SÁ

Missão Prática | Nível 3 | Mundo 3

Curso: Desenvolvimento Full Stack
Disciplina Nível 3: RPG0016 - BackEnd sem banco não tem
Número da Turma: 2024.2
Semestre Letivo: Mundo-3
Aluno: Gilvan Júnior Nascimento Gonçalves
Matrícula: 202304560188
URL GIT: https://github.com/Kakarotox10/Mundo3-MissaoPratica-Nivel3.git

1º Título da Prática: BackEnd sem banco não tem

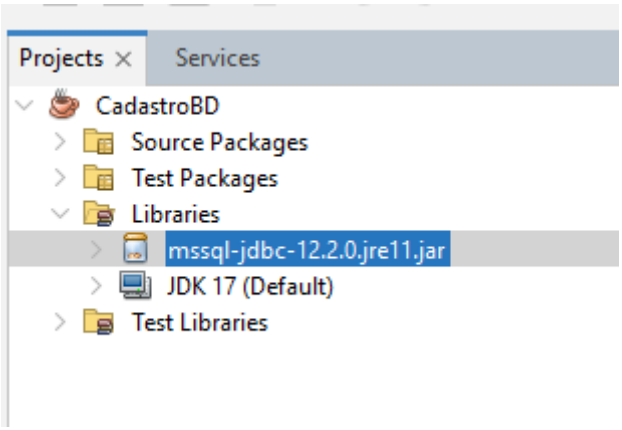
Criação de aplicativo Java, com acesso ao banco de dados SQL Server através do middleware JDBC.

2º Objetivo da Prática:

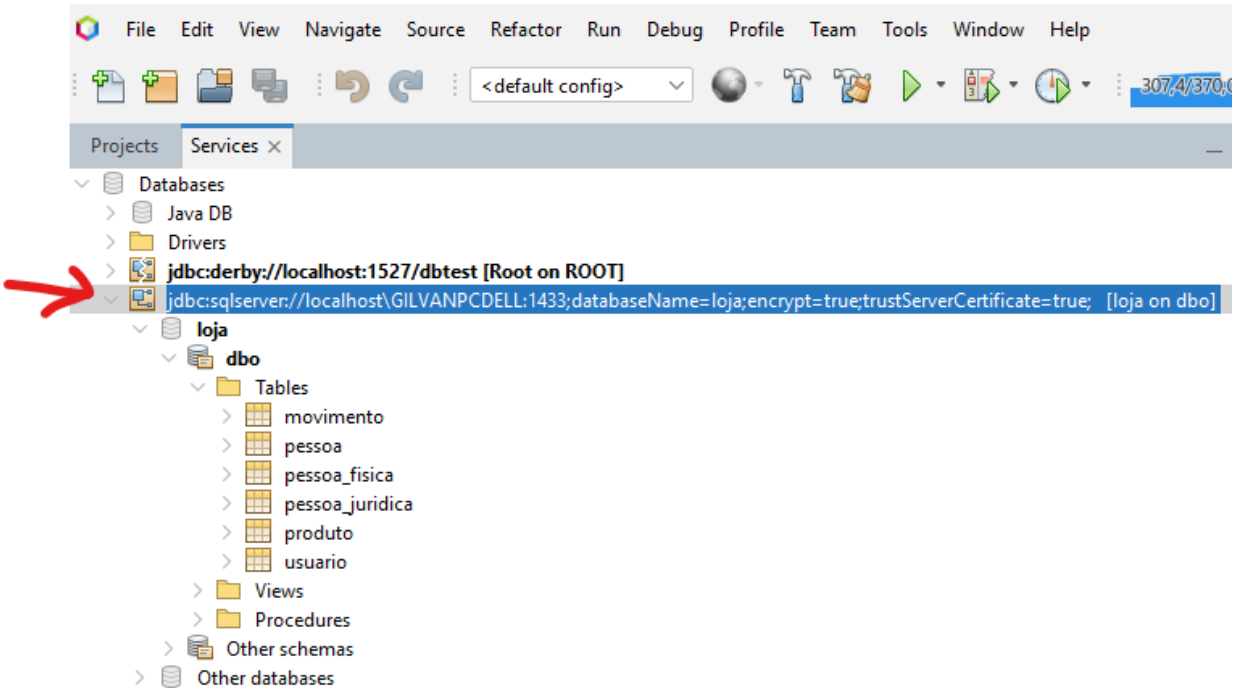
1. Implementar persistência com base no middleware JDBC.
2. Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
3. Implementar o mapeamento objeto-relacional em sistemas Java.
4. Criar sistemas cadastrais com persistência em banco relacional.
5. No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.

1º Procedimento | Mapeamento Objeto-Relacional e DAO

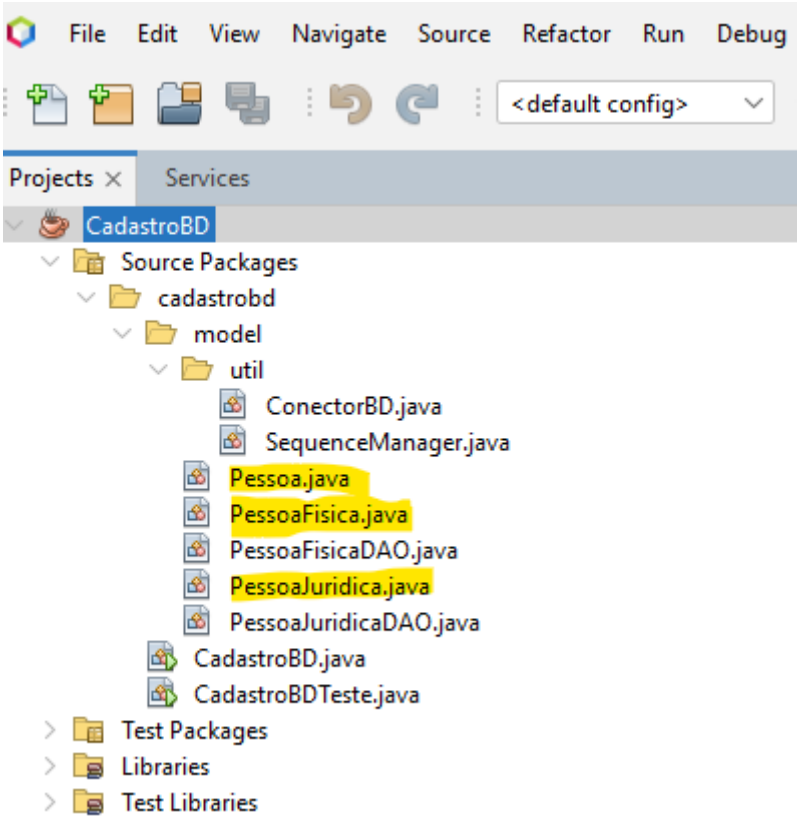
1. Criar o projeto e configurar as bibliotecas necessárias:



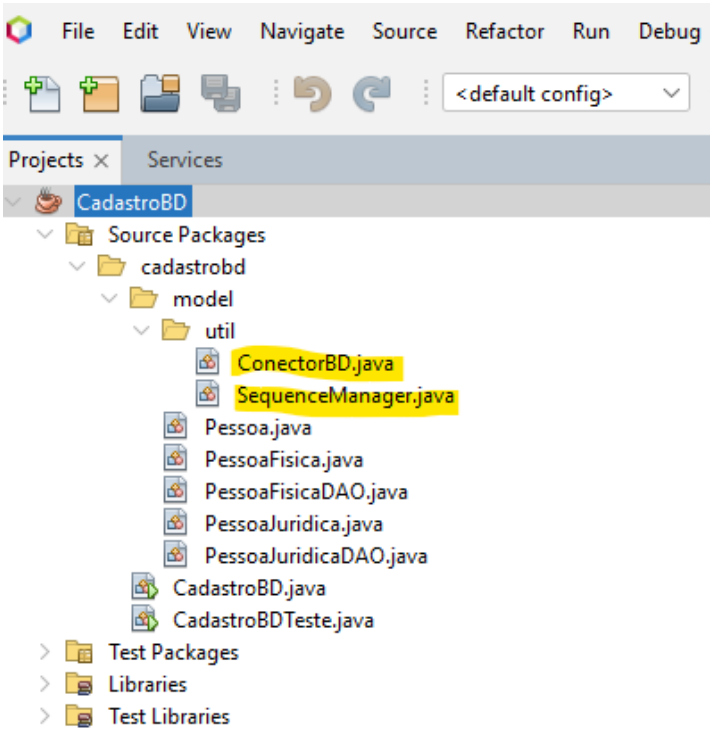
2. Configurar o acesso ao banco pela aba de serviços do NetBeans.



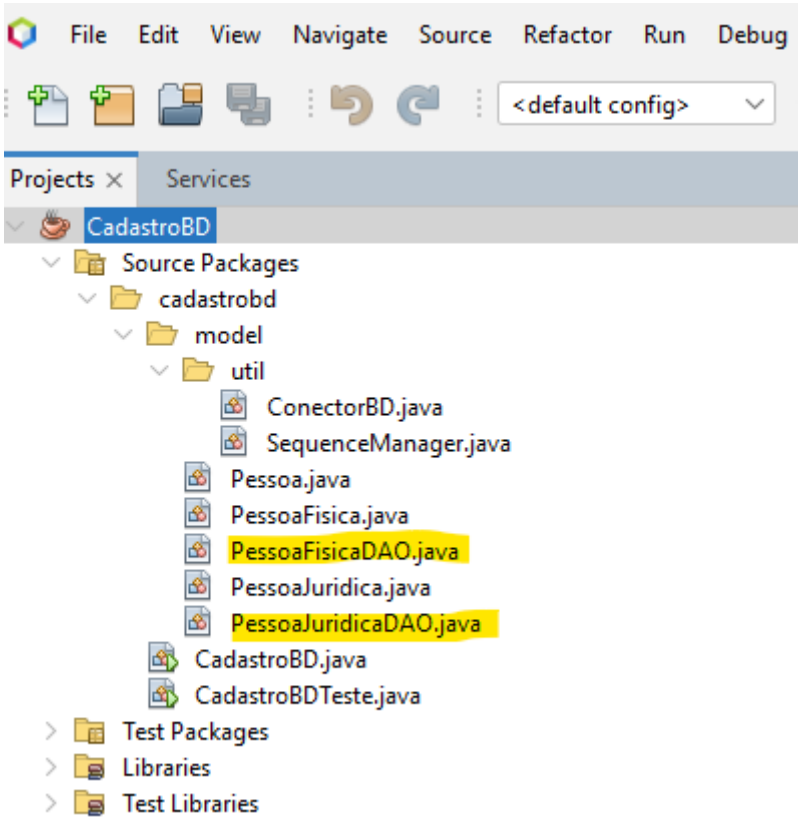
3. Voltando ao projeto, criar o pacote cadastrobd.model, e nele criar as classes apresentadas a seguir:



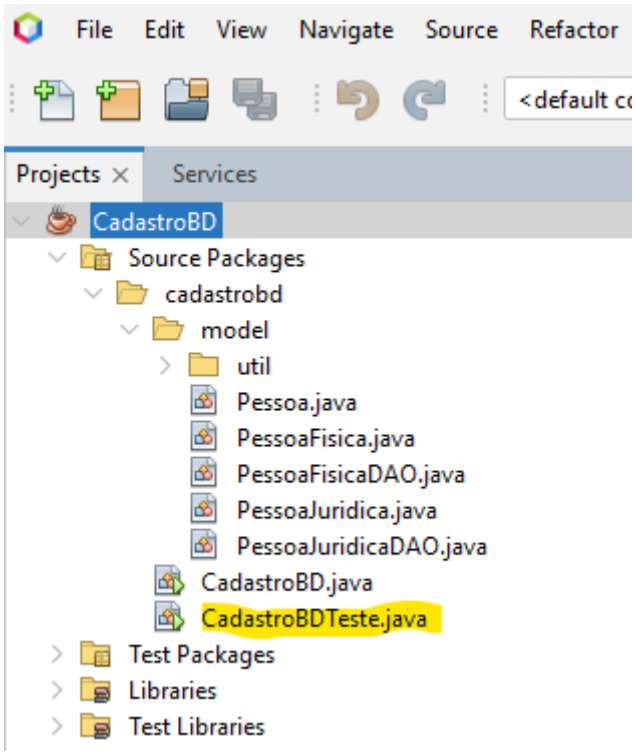
4. Criar o pacotes **cadastro.model.util**, para inclusão das classes utilitárias que são apresentadas a seguir:



5. Codificar as classes no padrão DAO, no pacote **cadastro.model**:



6. Criar uma classe principal de testes com o nome CadastroBDTeste, efetuando as operações seguintes no método main:



Análise e Conclusão do 1º Procedimento:

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

Os componentes de middleware, como o JDBC (Java Database Connectivity), desempenham um papel crucial na arquitetura de sistemas de software, especialmente em aplicações empresariais e web. Aqui estão algumas das principais razões pelas quais esses componentes são importantes:

- **Abstração:** Esconde a complexidade da infraestrutura subjacente, permitindo que os desenvolvedores se concentrem na lógica de negócio.
- **Reusabilidade:** Fornece componentes pré-construídos que podem ser reutilizados em diferentes projetos, acelerando o desenvolvimento.
- **Interoperabilidade:** Facilita a integração de diferentes sistemas e tecnologias, promovendo a heterogeneidade.
- **Escalabilidade:** Permite que as aplicações sejam escaladas de forma mais fácil e eficiente.
- **Gerenciamento:** Oferece ferramentas para monitorar e gerenciar o desempenho das aplicações.

Benefícios específicos do JDBC

- **Portabilidade:** Aplicações Java com JDBC podem ser executadas em diferentes plataformas e bancos de dados sem grandes modificações.
- **Produtividade:** Aumenta a produtividade dos desenvolvedores, pois eles não precisam escrever código SQL específico para cada banco de dados.
- **Segurança:** Oferece mecanismos para proteger os dados, como autenticação e autorização.
- **Transações:** Permite a realização de transações atômicas, garantindo a integridade dos dados.

Em resumo, o JDBC e outros componentes de middleware são fundamentais para criar aplicações que são escaláveis, seguras e fáceis de manter, ao mesmo tempo em que garantem uma comunicação eficiente e consistente com os sistemas de banco de dados.

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

Statement e **PreparedStatement** são duas interfaces do JDBC (Java Database Connectivity) utilizadas para executar comandos SQL em um banco de dados. Embora ambas sirvam para esse propósito, elas possuem características e aplicações distintas.

Statement

- **Criação:** A cada execução de um comando SQL, um novo objeto Statement é criado e enviado para o banco de dados.
- **Parâmetros:** Os parâmetros são concatenados diretamente à string do comando SQL, o que pode levar a problemas de segurança (injeção de SQL) e desempenho.
- **Compilação:** O comando SQL é compilado a cada execução.

PreparedStatement

- **Criação:** Um único objeto PreparedStatement é criado para um determinado comando SQL, e este objeto pode ser reutilizado várias vezes com diferentes valores para os parâmetros.
- **Parâmetros:** Os parâmetros são passados como valores, não como parte da string do comando SQL, o que evita a injeção de SQL e melhora a segurança.
- **Compilação:** O comando SQL é compilado apenas uma vez, na criação do PreparedStatement, o que otimiza o desempenho, especialmente em consultas complexas ou que serão executadas várias vezes.

Resumo

- **Statement** é mais adequado para consultas SQL simples e situações em que a segurança e o desempenho não são preocupações primárias.
- **PreparedStatement** é recomendado para consultas SQL complexas e para qualquer situação onde a segurança e o desempenho são importantes. Ele também facilita a manutenção do código e a gestão de parâmetros.

Em geral, o PreparedStatement é preferido na maioria das situações devido à sua segurança, desempenho e facilidade de uso.

c. Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (Data Access Object) é uma ferramenta poderosa para melhorar a manutenibilidade do software, especialmente em aplicações que interagem com bancos de dados.

1. Separação de Responsabilidades:

- **Lógica de Negócio vs. Acesso a Dados:** O DAO encapsula todas as operações de acesso ao banco de dados em uma única camada. Isso significa que a lógica de negócios da sua aplicação não precisa se preocupar com os detalhes de como os dados são persistidos. Essa separação clara facilita a compreensão e a manutenção do código.

- **Facilita Testes:** Com a lógica de acesso a dados isolada, você pode testar a sua lógica de negócios de forma independente, utilizando mocks ou stubs para simular o comportamento do banco de dados.

2. Abstração:

- **Independência do Banco de Dados:** O DAO fornece uma camada de abstração sobre o banco de dados específico. Isso significa que você pode trocar o banco de dados sem precisar modificar a lógica de negócios.
- **Facilidade de Mudanças:** Se você precisar mudar a estrutura do banco de dados ou a forma como os dados são armazenados, você pode fazer isso alterando apenas a implementação do DAO, sem afetar o resto da aplicação.

3. Reutilização:

- **Componentes Reutilizáveis:** Os DAOs podem ser reutilizados em diferentes partes da aplicação, reduzindo a duplicação de código e aumentando a consistência.

4. Facilita a Manutenção:

- **Localização de Erros:** Ao isolar a lógica de acesso a dados em um único lugar, fica mais fácil encontrar e corrigir erros relacionados ao banco de dados.
- **Evita Mudanças Cascata:** Alterações na estrutura do banco de dados tendem a ter um impacto menor na aplicação como um todo, pois estão concentradas no DAO.

  O padrão DAO melhora a manutenibilidade do software ao:

  Aumentar a coesão: Concentrando a lógica de acesso a dados em um único lugar. **Diminuir o acoplamento:** Reduzindo a dependência entre a lógica de negócios e os detalhes de implementação do banco de dados. **Facilitar a testabilidade:** Permitindo testar a lógica de negócios de forma isolada. **Aumentar a reusabilidade:** Permitindo a reutilização de componentes.

  Em resumo, o padrão DAO melhora a manutenibilidade do software ao fornecer uma estrutura clara e organizada para o acesso a dados, promovendo a separação de preocupações, encapsulamento, testabilidade, reusabilidade e facilidade de manutenção.

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

A herança, um conceito fundamental na programação orientada a objetos, não possui um mapeamento direto e nativo em bancos de dados relacionais. Isso ocorre porque bancos de dados relacionais são baseados em tabelas, que representam entidades planas e não hierárquicas como as classes em orientação a objetos.

Por que isso é um problema?

- **Modelagem de dados:** Muitas vezes, o mundo real possui hierarquias e relações de tipo-subtipo que a herança captura de forma natural.
- **Consultas:** Consultas que envolvem hierarquias podem se tornar complexas e ineficientes em um modelo puramente relacional.

Estratégias para Modelar Herança em Bancos Relacionais

Diante desse desafio, diversas estratégias foram desenvolvidas para simular a herança em bancos de dados relacionais. Cada uma possui suas vantagens e desvantagens, e a escolha da melhor abordagem depende das características específicas da aplicação e do banco de dados utilizado.

1. Tabela por Subtipo:

- **Descrição:** Cada subtipo possui sua própria tabela, contendo os atributos específicos e uma chave estrangeira para a tabela da superclasse.
- **Vantagens:** Simplicidade, boa performance para consultas em subtipos específicos.
- **Desvantagens:** Consultas que envolvem múltiplos subtipos podem ser complexas e ineficientes.

2. Tabela Única com Coluna Discriminadora:

- **Descrição:** Uma única tabela armazena todos os objetos, e uma coluna adicional (discriminadora) indica o tipo do objeto.
- **Vantagens:** Simplicidade, boa performance para consultas genéricas.
- **Desvantagens:** Pode levar a muitos valores nulos, especialmente se houver poucas diferenças entre os subtipos.

3. Tabela Principal + Tabelas Filhas:

- **Descrição:** Uma tabela principal armazena os atributos comuns, e tabelas filhas armazenam os atributos específicos, com uma chave estrangeira para a tabela principal.
- **Vantagens:** Boa normalização, flexibilidade para adicionar novos subtipos.

- **Desvantagens:** Consultas podem se tornar complexas devido aos joins.

4. Herança Total:

- **Descrição:** Cada subtipo possui sua própria tabela, contendo todos os atributos, inclusive os herdados.
- **Vantagens:** Simplicidade para consultas.
- **Desvantagens:** Redundância de dados, dificuldade de manter a consistência.

5. Mapeamento Objeto-Relacional (ORM):

- **Descrição:** Frameworks ORM, como Hibernate e Entity Framework, abstraem a complexidade do mapeamento entre objetos e bancos de dados, permitindo modelar a herança de forma mais natural.
- **Vantagens:** Facilidade de desenvolvimento, mapeamento transparente da herança.
- **Desvantagens:** Pode haver perda de desempenho em algumas situações.

Qual a melhor abordagem?

A escolha da melhor estratégia depende de diversos fatores, como:

- **Frequência de consultas:** Se você precisa realizar muitas consultas que envolvem múltiplos subtipos, a tabela única com coluna discriminadora pode ser uma boa opção.
- **Número de subtipos:** Se você tiver muitos subtipos, a tabela principal com tabelas filhas pode ser mais adequada.
- **Performance:** A performance das consultas pode variar significativamente entre as diferentes abordagens.
- **Complexidade da hierarquia:** Hierarquias complexas podem exigir combinações de diferentes estratégias.

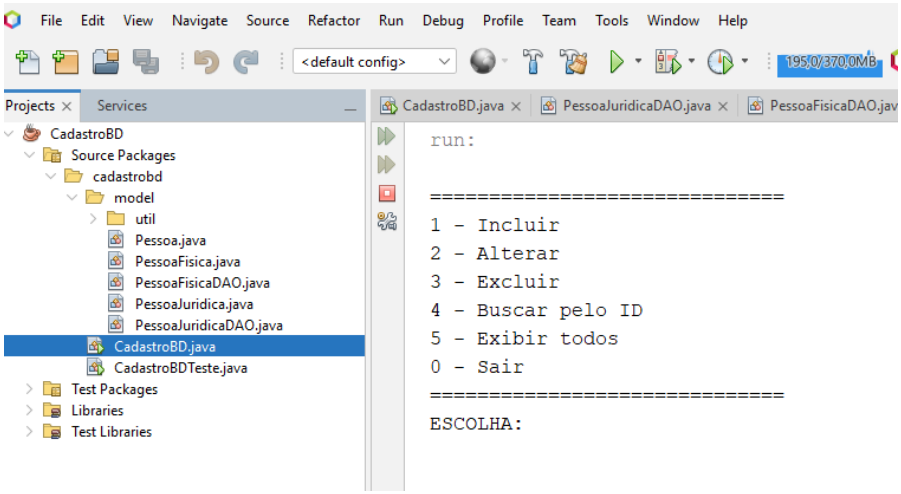
Em resumo:

A herança em bancos de dados relacionais é um desafio que exige um planejamento cuidadoso. A escolha da melhor estratégia depende das características específicas da aplicação e do banco de dados utilizado. Frameworks ORM podem simplificar o processo, mas é importante entender as implicações de cada abordagem para tomar a decisão mais adequada.

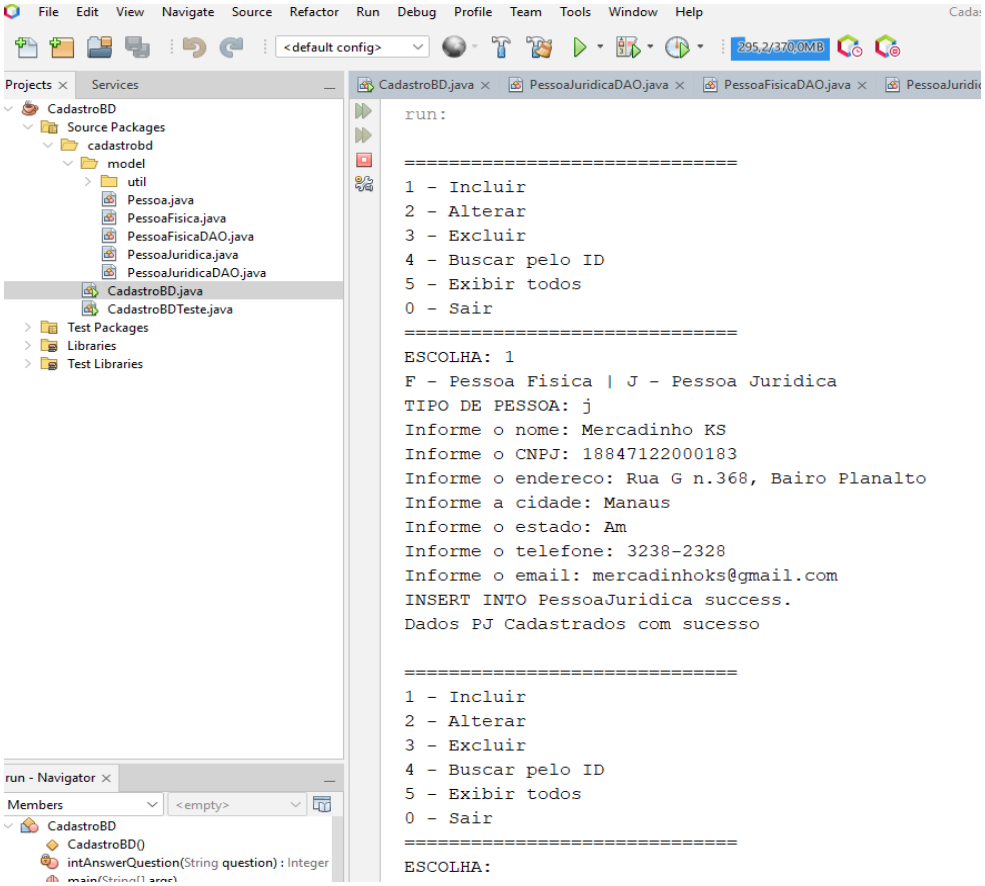
2º Procedimento | Alimentando a Base

1. Alterar o método **main** da classe principal do projeto, para implementação do cadastro em modo texto:

a) Apresentar as opções do programa para o usuário, sendo 1 para incluir, 2 para alterar, 3 para excluir, 4 para exibir pelo id, 5 para exibir todos e 0 para finalizar a execução.



b) Selecionada a opção **incluir**, escolher o tipo (Física ou Jurídica), receber os dados a partir do teclado e adicionar no banco de dados através da classe DAO correta.



c) Selecionada a opção **alterar,** escolher o tipo (Física ou Jurídica), receber o id a partir do teclado, apresentar os dados atuais, solicitar os novos dados e alterar no banco de dados através do DAO.

```
==============================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
==============================
ESCOLHA: 2
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: j
Informe o ID da Pessoa Juridica: 10
Informe o nome: Mercadinho KS2
Informe o CNPJ: 18847122000183
Informe o endereco: Rua G n.368, Bairo Planalto 2
Informe a cidade: Manaus2
Informe o estado: AM
Informe o telefone: 3238-1495
Informe o email: mercadinhoks2@gmail.com
Dados PJ ALTERADOS com sucesso
```

d) Selecionada a opção **excluir**, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e remover do banco de dados através do DAO.

```
==============================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
==============================
ESCOLHA: 3
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: J
Informe o ID da Pessoa Juridica: 10
Excluido com sucesso.
```

e) Selecionada a opção **obter**, escolher o tipo (Física ou Jurídica), receber o id a partir do teclado e apresentar os dados atuais, recuperados do banco através do DAO.

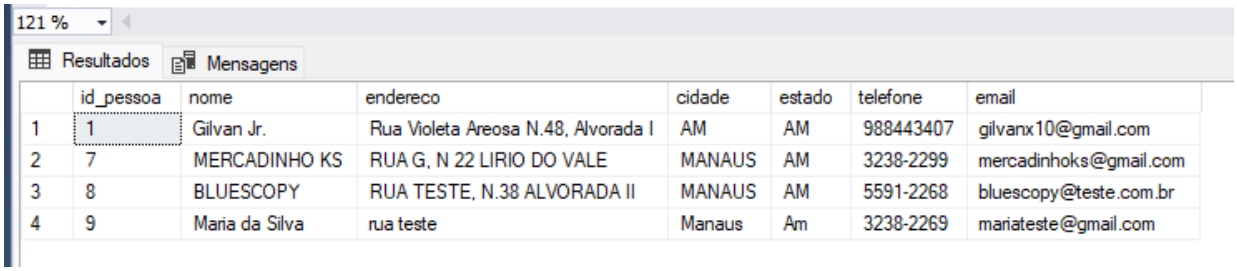
```
==============================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
==============================
ESCOLHA: 4
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: j
Informe o ID da Pessoa Juridica: 10
id: 10
nome: Mercadinho KS2
endereco: Rua G n.368, Bairo Planalto 2
cidade: Manaus2
estado: AM
telefone: 3238-1495
email: mercadinhoks2@gmail.com
CNPJ: 18847122000183
```

f) Selecionada a opção **obterTodos**, escolher o tipo (Física ou Jurídica) e apresentar os dados de todas as entidades presentes no banco de dados por intermédio do DAO.

```
id: 7
nome: MERCADINHO KS
endereco: RUA G, N 22 LIRIO DO VALE
cidade: MANAUS
estado: AM
telefone: 3238-2299
email: mercadinhoks@gmail.com
CNPJ: 18625187000184
---------------------------------
id: 8
nome: BLUESCOPY
endereco: RUA TESTE, N.38 ALVORADA II
cidade: MANAUS
estado: AM
telefone: 5591-2268
email: bluescopy@teste.com.br
CNPJ: 21028604000116
---------------------------------

================================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
================================
ESCOLHA:
```

Exemplo dos registros no banco Sql Server.



121 %

Resultados | Mensagens

	id_pessoa	nome	endereco	cidade	estado	telefone	email
1	1	Gilvan Jr.	Rua Violeta Areosa N.48, Alvorada I	AM	AM	988443407	gilvanx10@gmail.com
2	7	MERCADINHO KS	RUA G, N 22 LIRIO DO VALE	MANAUS	AM	3238-2299	mercadinhoks@gmail.com
3	8	BLUESCOPY	RUA TESTE, N.38 ALVORADA II	MANAUS	AM	5591-2268	bluescopy@teste.com.br
4	9	Maria da Silva	rua teste	Manaus	Am	3238-2269	mariateste@gmail.com

g) Qualquer **exceção** que possa ocorrer durante a execução do sistema deverá ser tratada.

```
================================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
================================
ESCOLHA: 1
F - Pessoa Fisica | J - Pessoa Juridica
TIPO DE PESSOA: teste
Erro: Escolha Invalida!

================================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
================================
ESCOLHA:
```

(exemplo de erro: uma exceção é lançada quando o usuário não informa o tipo de pessoa F ou J corretamente)

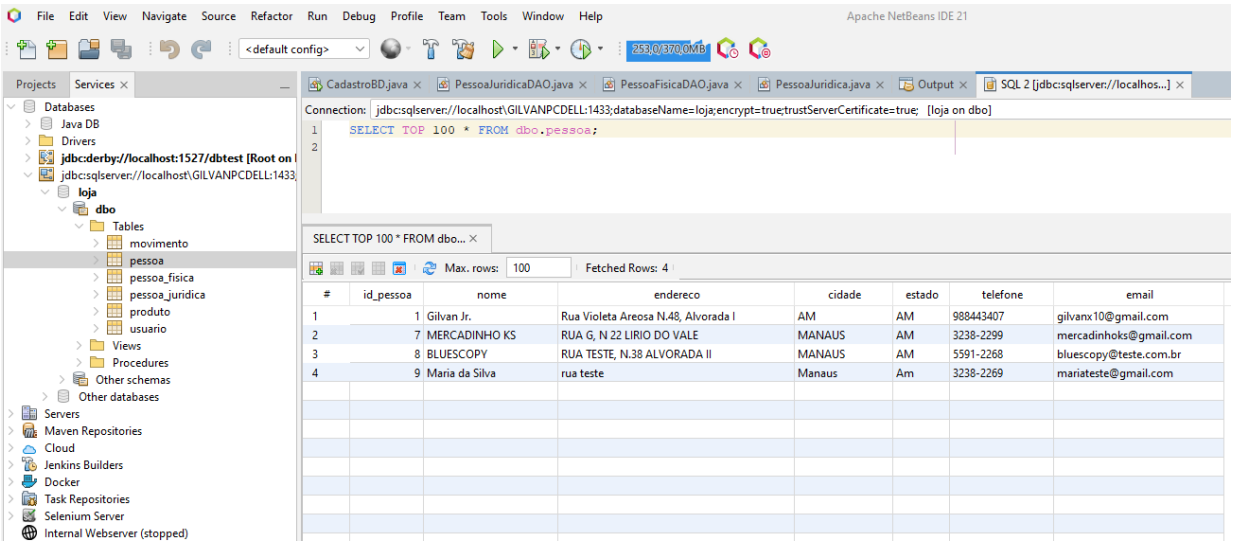
h) Selecionada a opção **sair**, finalizar a execução do sistema.

```
run:

==============================
1 - Incluir
2 - Alterar
3 - Excluir
4 - Buscar pelo ID
5 - Exibir todos
0 - Sair
==============================
ESCOLHA: 0
Sistema Finalizado!
BUILD SUCCESSFUL (total time: 2 seconds)
```

2. Testar as funcionalidades do sistema:

Feitas as operações, verificar os dados no SQL Server, com a utilização da aba Services, divisão Databases, do NetBeans, ou através do SQL Server Management Studio.



Análise e Conclusão do 2º Procedimento:

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

A persistência em arquivos e a persistência em bancos de dados são duas abordagens distintas para armazenar e recuperar dados. Ambas têm seus usos, vantagens e desvantagens, e a escolha entre elas depende das necessidades específicas da aplicação. Segue abaixo as principais diferenças entre essas duas abordagens:

Persistência em Arquivo:

- Melhor para dados simples, pequenas aplicações e casos em que a flexibilidade e simplicidade são mais importantes.
- Menos suporte para operações complexas, escalabilidade e integridade dos dados.

Persistência em Banco de Dados:

- Melhor para aplicações complexas, grandes volumes de dados e onde a integridade, segurança e desempenho são críticos.
- Oferece suporte avançado para consultas, transações, escalabilidade e administração.

A escolha entre persistência em arquivo e persistência em banco de dados deve ser feita com base nas necessidades específicas da aplicação, no volume de dados, nos requisitos de desempenho e segurança, e na complexidade das operações necessárias.

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

No Java 8 e em versões subsequentes, o uso de operadores lambda trouxe uma série de melhorias na sintaxe e na expressividade do código, especialmente ao lidar com coleções e operações de fluxo de dados. Vou explicar como os operadores lambda simplificaram a impressão dos valores contidos nas entidades e como isso se reflete na prática.

Impressão de Valores em Coleções:

Antes do Java 8, a impressão de valores em uma coleção, como uma lista ou um conjunto, geralmente envolvia o uso de loops explícitos, como for ou foreach. Com a introdução das expressões lambda e das APIs de Streams no Java 8, a sintaxe para realizar operações em coleções, incluindo a impressão de valores, tornou-se mais concisa e legível.

Comparação das Abordagens

- **Código Antes do Java 8:** Requer um loop explícito e o gerenciamento manual da lógica de impressão. Pode ser mais verboso e propenso a erros quando a lógica é mais complexa.
- **Código Usando Lambda e Streams:** Simplifica o código ao permitir operações funcionais diretas sobre coleções. A sintaxe é mais declarativa e a operação é mais fácil de entender. Usar referências de método reduz ainda mais a necessidade de código adicional.

Benefícios Adicionais

1. **Legibilidade:** O uso de lambda e streams melhora a legibilidade do código ao remover a necessidade de loops explícitos e ao tornar a intenção do código mais clara.
2. **Manutenção:** Com menos código e uma sintaxe mais clara, o código é mais fácil de manter e menos propenso a erros.
3. **Flexibilidade:** As APIs de Streams permitem operações complexas como filtragem, mapeamento e redução de dados de forma fluida e encadeada, o que pode ser combinado com a impressão para operações mais sofisticadas.

   Em resumo, o uso de operadores lambda e APIs de Streams no Java simplifica a impressão e manipulação de valores em coleções ao fornecer uma sintaxe mais concisa e expressiva, além de suportar operações funcionais e encadeadas de maneira fluida.

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

Em Java, o método main é o ponto de entrada para a execução de um programa. Ele precisa ser marcado como static porque é chamado pela Java Virtual Machine (JVM) para iniciar a aplicação, e isso ocorre sem a criação de uma instância da classe onde o método main está definido. Vou explicar em detalhes por que o método main e, por conseguinte, outros métodos acionados diretamente por ele precisam ser estáticos:

1. Conceito de Métodos Estáticos

- **Métodos Estáticos:** São métodos que pertencem à classe em vez de a uma instância específica dessa classe. Eles podem ser chamados diretamente usando o nome da classe sem a necessidade de criar uma instância da classe.
- **Métodos de Instância:** São métodos que pertencem a uma instância específica de uma classe e requerem uma instância (objeto) da classe para serem chamados.

2. Método main como Ponto de Entrada

O método main é definido como:

```
public static void main(String[] args) {
  // Código
}
```

Razões para o Método main ser Estático:

1. **Chamada Sem Instância:**

- O método main é o ponto de entrada do programa e é chamado pela JVM quando o programa é iniciado. A JVM não cria uma instância da classe antes de chamar o método main. Portanto, o método main deve ser estático para que a JVM possa invocá-lo diretamente, sem a necessidade de criar uma instância da classe.

2. **Acesso a Métodos e Variáveis Estáticas:**

- Dentro do método main, você pode acessar outros métodos e variáveis estáticas diretamente, sem criar uma instância da classe. Isso é útil para funções utilitárias ou operações que não dependem do estado de uma instância específica da classe.

3. Consistência e Simplicidade

- **Consistência:** Marcar o método main como static é consistente com o fato de que ele deve ser executado independentemente de qualquer instância da classe. Isso evita a complexidade desnecessária de ter que criar uma instância da classe apenas para iniciar a execução do programa.
- **Simplicidade:** Permite que o programa comece sua execução de forma direta e simples. Sem a necessidade de instanciar um objeto, o ponto de entrada do programa pode ser facilmente acessado e iniciado pela JVM.

Em resumo, métodos acionados diretamente pelo método main precisam ser marcados como static porque o método main é executado sem a necessidade de uma instância da classe. A marcação como static permite que métodos e variáveis sejam acessados diretamente sem criar um objeto, simplificando a execução inicial do programa.