

**UNIVERSIDADE ESTÁCIO DE SÁ**

**FULLSTACK**

**Mundo 03 - Nível 03   
  
BackEnd sem banco não tem.**

**Criação de aplicativo Java, com acesso   
ao banco de dados SQL Server através do**

**middleware JDBC.**

Herval Rosano Dantas  
Matrícula 202205119203

RIO DE JANEIRO – RJ  
2023

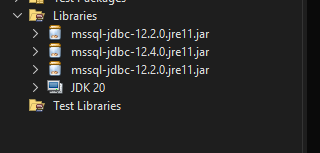
**Objetivo da Prática**

* Implementar persistência com base no middleware JDBC.
* Utilizar o padrão DAO (Data Access Object) no manuseio de dados.
* Implementar o mapeamento objeto-relacional em sistemas Java.
* Criar sistemas cadastrais com persistência em banco relacional.

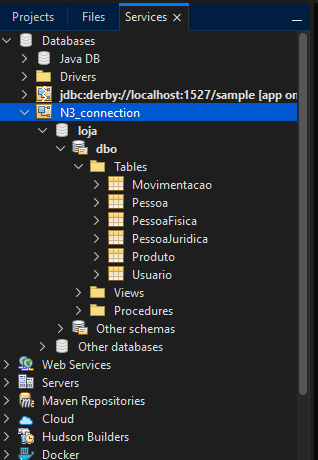
Na conclusão deste trabalho, terá sido criado um aplicativo cadastral com uso do SQL  
Server na persistência de dados.

**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.

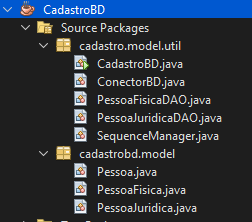


1. Criar o pacote **cadastrobd.model**, e nele criar as classes:

Classe Pessoa, Classe PessoaFisica, herdando de Pessoa, com acréscimo do campo cpf, Classe PessoaJuridica, herdando de Pessoa, com acréscimo do campo cnpj.

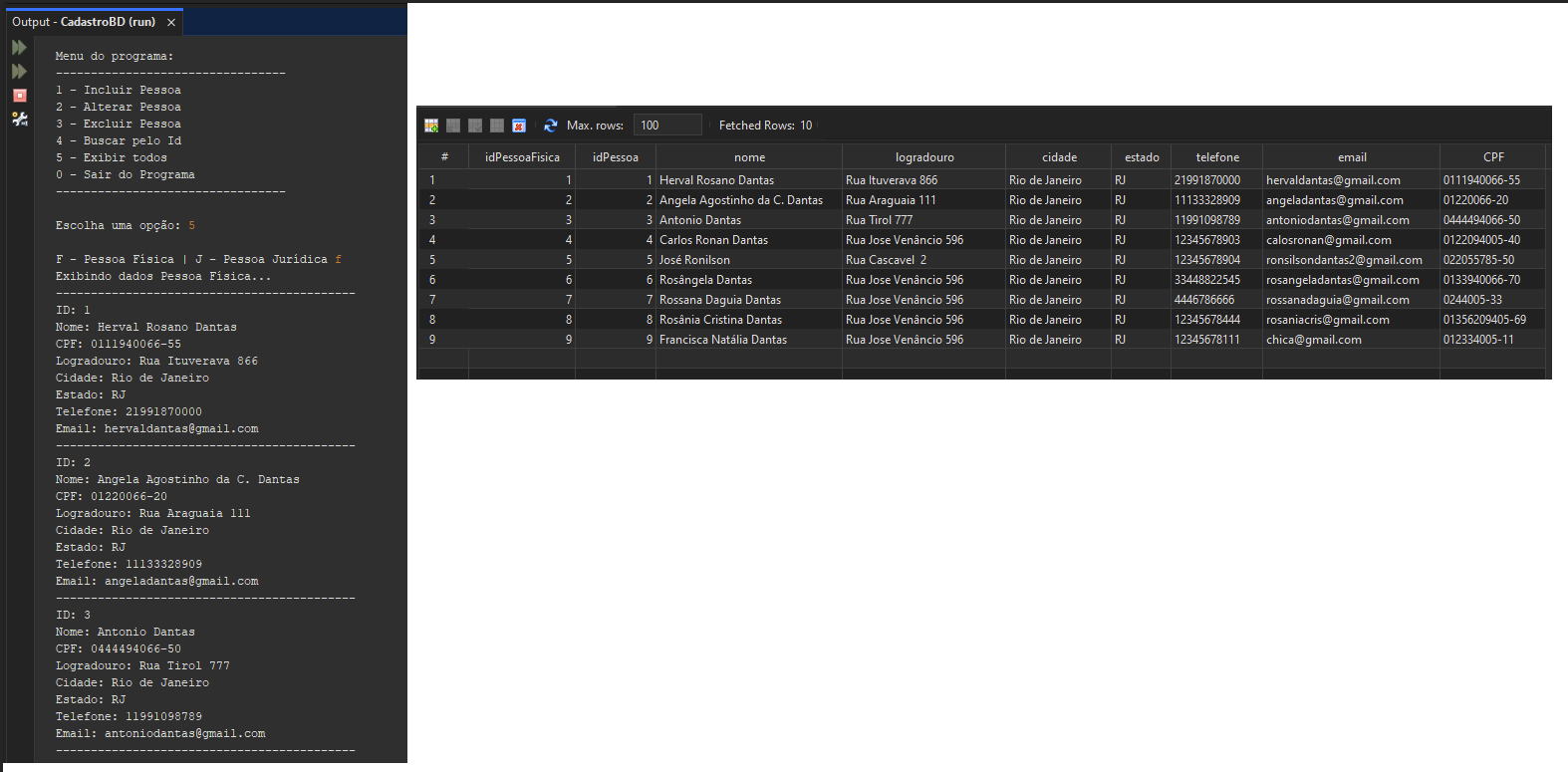
4. Criar o pacotes **cadastro.model.util**, para inclusão das classes:

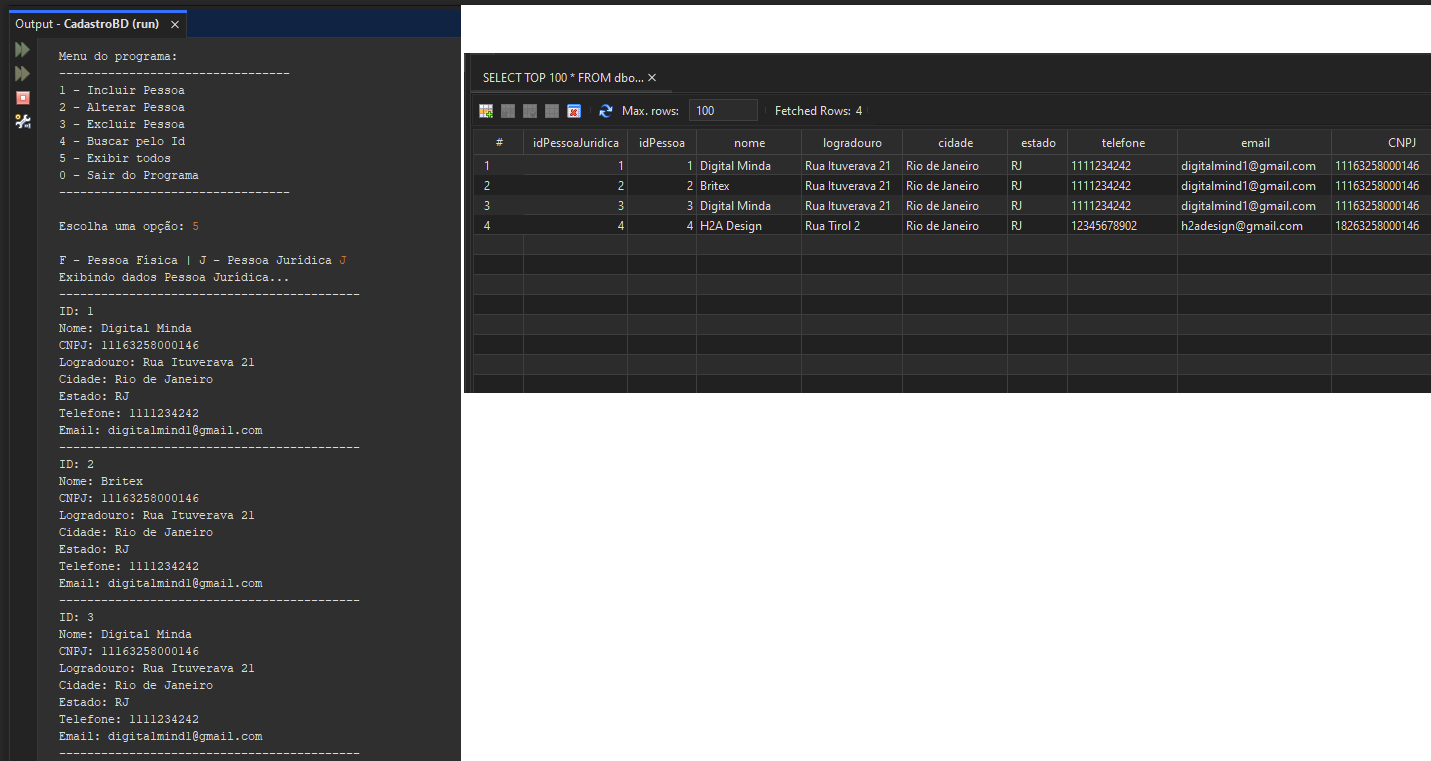
ConectorBD, SequenceManager, classes no padrão DAO PessoaFisicaDAO e PessoaJuridicaDAO.

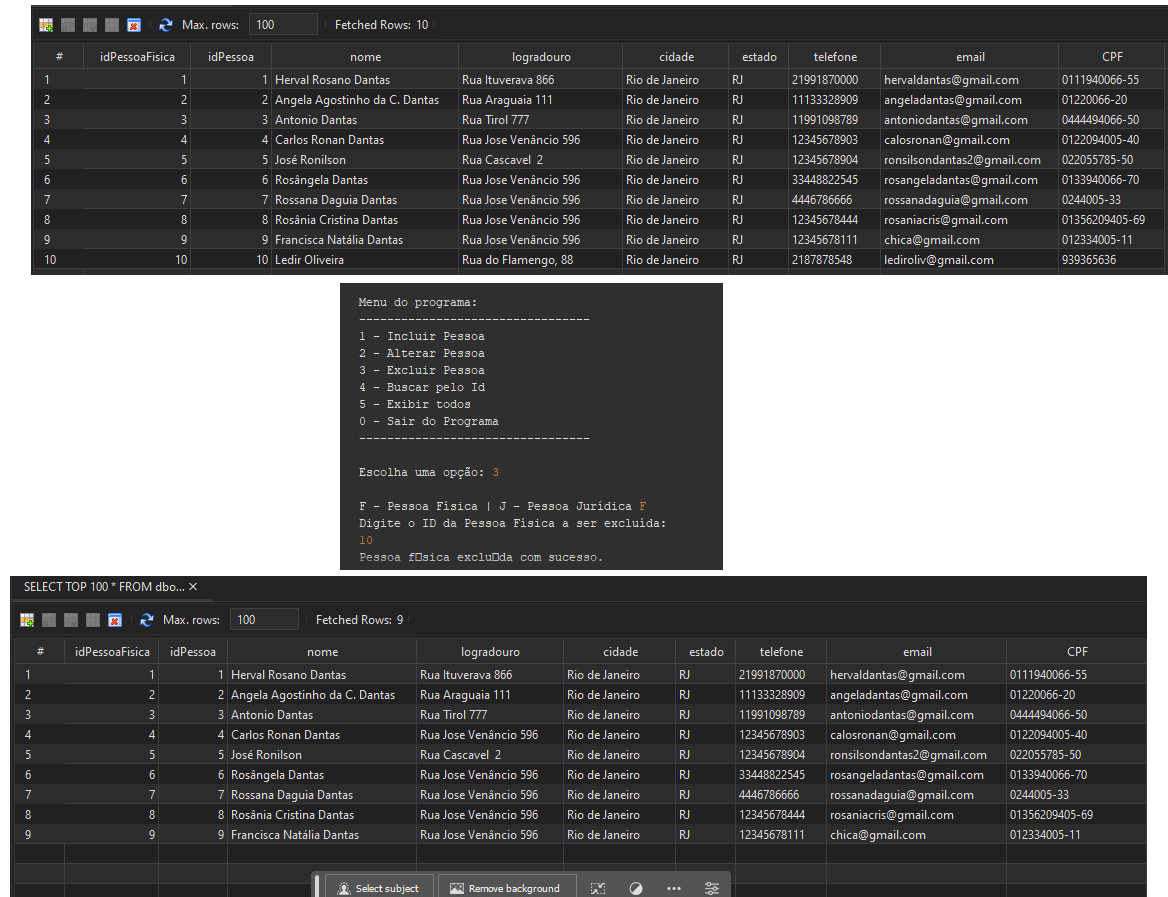


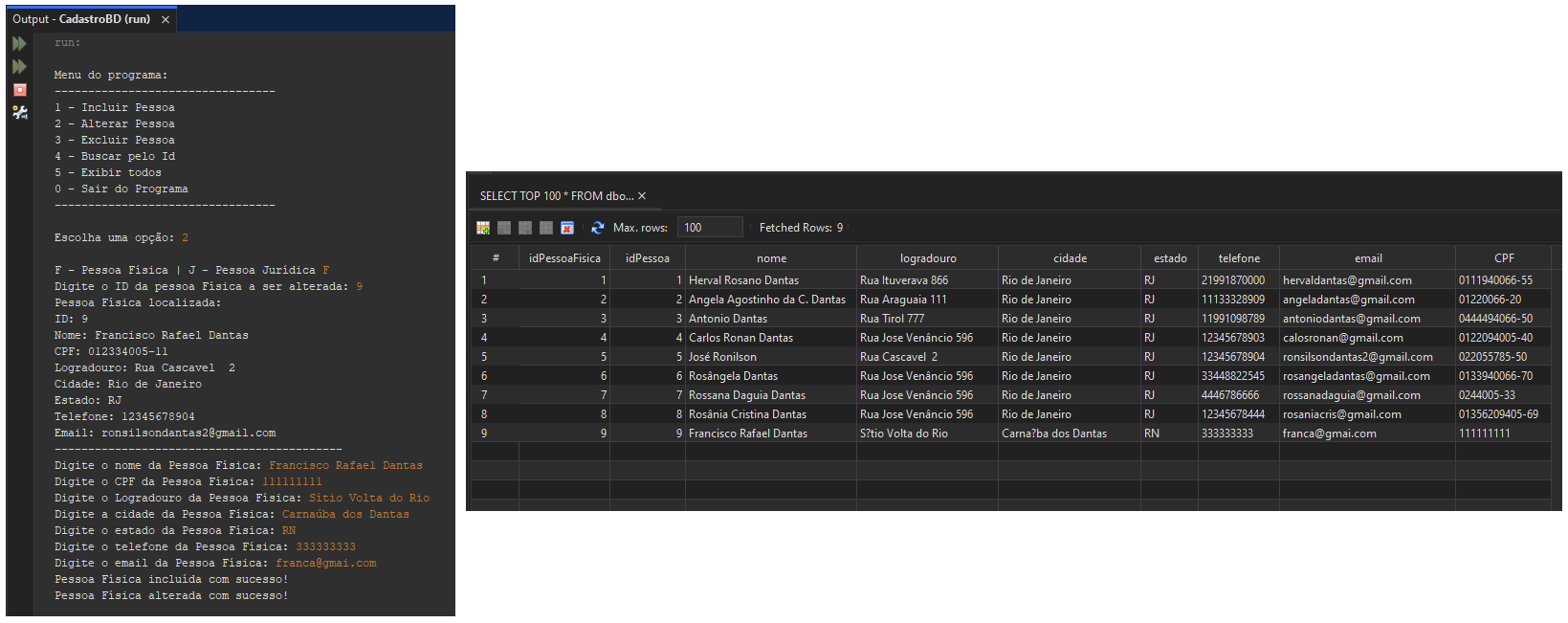
Todas as funcionalidades deste sistema de cadastro serão mostradas **no segundo procedimento** onde o mesmo será alimentado através de um cadastro em modo texto.

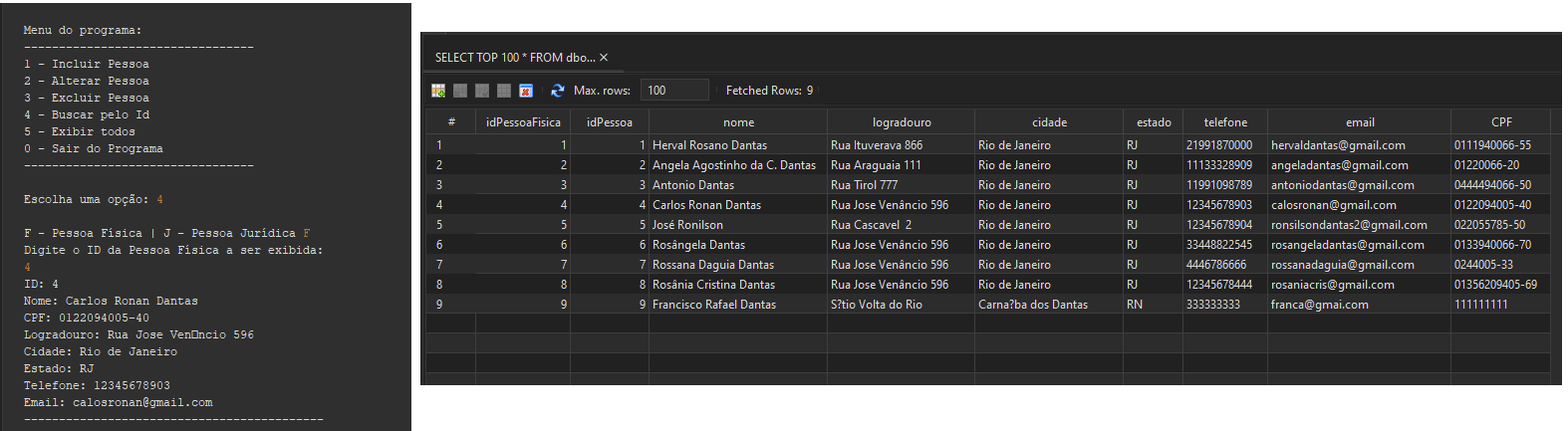
Exibindo todas Pessoas Físicas



Exibindo todas Pessoas Jurídicas

Exclusão baseado no ID – caso abaixo foi o ID 10

Alteração baseado no ID – caso abaixo foi o ID 9

Buscar baseado no ID – caso abaixo foi o ID 4

Incluir uma Pessoa Física

**Análise e Conclusão:**

**1 - Qual a importância dos componentes de middleware, como o JDBC?**

O JDBC (Java Database Connectivity), desempenha um importante papel quando se trata de conexões com Banco de Dados, e como se trata de um middleware (Middle = meio) o mesmo consegue fazer esta conexão com diferentes Sistemas de Gerenciamento de Banco de Dados (SGBD) do tipo: MySQL, Oracle, SQL Server, PostgreSQL, etc.

O JDBC fornece uma camada de abstração que permite se desenvolva a programação dos códigos Java independente do SGBD o que o torna bastante portátil. Algumas das principais funcionalidade do JDBC são:

1. Gerenciamento de Conexões principalmente abrindo e fechando as conexões com os BD, garantindo economia e eficácia nos recursos de processamento, e evitando vazamentos nas conexões.
2. Execução de Consultas e Comandos que permite seja enviada consultas SQL e comandos para o banco de dados. Isso é essencial para realizar operações de leitura e gravação.
3. Preparação de Declarações SQL o JDBC suporta a preparação de declarações SQL parametrizadas, evitando a injeção de SQL o que melhora a segurança das consultas.
4. Gerenciamento de Transações dá suporte as transações de forma a permitir que se inicie, comprometa ou até mesmo que se reverta as transações (rollback) caso haja inconsistência dos dados nas operações complexas.

**2 - Qual a diferença no uso de Statement ou PreparedStatement para a manipulação de dados?**

A diferença principal é que PreparedStatement é mais seguro e tem um melhor desempenho especialmente para consultas dinâmicas. Já o Statement pode ser mais simples, mas requer cuidados adicionais para evitar vulnerabilidades de injeção de SQL.

Veja estas comparações:

|  |  |  |
| --- | --- | --- |
| **Aspecto** | **Statement** | **PreparedStatement** |
| Segurança | Vulnerável a ataques de injeção de SQL | Protege contra injeção de SQL automaticamente |
| Desempenho | Menos eficiente para consultas repetidas | Mais eficiente para consultas repetidas devido à pré-compilação |
| Usabilidade | Simples de usar para consultas estáticas | Mais flexível para consultas dinâmicas |
| Leitura de Código | Pode tornar o código menos legível devido à concatenação de SQL | Tende a tornar o código mais legível, pois consulta e parâmetros são separados |

**3 - Como o padrão DAO melhora a manutenibilidade do software?**

O padrão DAO ajuda a melhorar a manutenibilidade do software pois o mesmo oferece uma arquitetura bem organizada, reutilização de código e a abstração do banco de dados além de tornar o código mais limpo e seguro. Este padrão promove a separação clara das responsabilidades pois isola a lógica de acesso ao banco de dados em uma camada, facilitando a compreensão e a manutenção do código, uma vez que cada componente tem um propósito bem definido.

Ao encapsularmos a lógica de acesso ao banco de dados em objetos DAO, podemos reutilizar estes códigos em várias partes do aplicativo. Isso evita a duplicação e facilita a manutenção, uma vez que qualquer mudança nas operações de banco de dados poderá ser feita em apenas um lugar. Existem muitas outras melhorias que o padrão DAO proporciona.

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

Existem duas abordagens principais para implementar herança em um modelo de banco de dados relacional: herança de tabela única (single-table inheritance) e herança de tabela múltipla (multiple-table inheritance). A herança de tabela única usa uma única tabela com uma coluna adicional para indicar o tipo de entidade. É um esquema simples, principalmente quando se refere a facilidade de consulta. Todavia, pode levar a uma tabela muito ampla e de esquema denormalizado. Ou seja: tudo numa tabela.

Já a herança de tabela múltipla usa uma tabela separada para cada classe, mantendo a normalização, e usando-se as chaves estrangeiras se faz os relacionamentos, mas também pode complicar as consultas na hora de fazer as junções. Mas com certeza é mais fácil de adicionar novas entidades e de fazer manutenção.

**5 - Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?**

A persistência em arquivo é mais simples e adequada para casos de uso simples ou armazenamento de dados temporários, enquanto a persistência em banco de dados oferece recursos mais avançados, segurança, escalabilidade e eficiência para aplicativos que requerem gerenciamento e consulta de dados de maneira mais sofisticada e robusta.

Só posso acreditar que a persistência de dados em arquivo foi útil no processo inicial da computação, pois não tem como se comparar com a robustez da persistência em banco de dados que tem um sistema de gerenciamento muito versátil, o famoso SGBD. Além de ter também uma poderosa linguagem de consulta SQL que permite consultas poderosas e flexíveis usando uma linguagem de alto nível (mais humana). Tudo bem que tá em inglês, mas por exemplo: selecione todas colunas da tabela tal, cujo campo valor unitário seja igual a 100 ==> SELECT \* FROM dbo.carros WHERE valueUnid = 100; - pra mim tá mais em linguagem humana do que em linguagem de máquina.

**6 - 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 no Java simplificou significativamente. Antes da introdução de lambdas, era comum usar expressões anônimas e iterar manualmente sobre coleções para realizar operações como a impressão de valores. Agora com o operador lambda reduziu-se muito a quantidade de código, a sintaxe é mais concisa além de tornar o código mais legível e de fácil compreensão. Exemplo bem simples:

Sem Lambda:

// Imprimir os nomes ordenados

for (String nome : nomes) {  
 System.out.println(nome);   
}

Com Lambda:

nomes.forEach(nome -> System.out.println(nome));

Veja como a iteração ficou muito mais simples.

**7 - 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 em Java precisam ser marcados como static porque o main é o ponto de entrada do programa, e o modificador static permite que os métodos sejam chamados diretamente pelas suas próprias classes sem que haja a necessidade de instanciá-lo, no caso a classe main.