

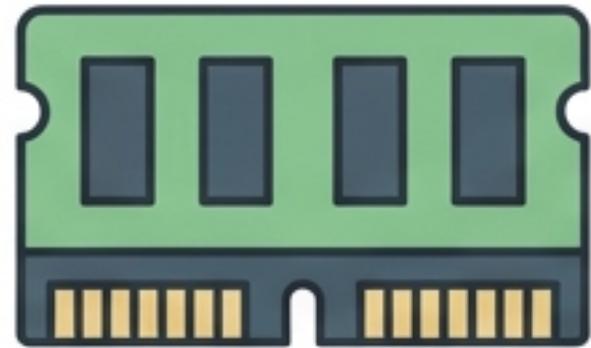


Persistência de Dados com Java JDBC

Do conceito à implementação profissional com o Padrão DAO

Baseado na obra de Deitel & Deitel e material de Rafael Vargas Mesquita.

O Desafio: A Volatilidade da Memória



Memória RAM (Volátil)

Enquanto a aplicação roda, os objetos existem. Ao fechar o programa ou desligar o computador, os dados são perdidos.

Exemplo: Objeto Cliente



Banco de Dados (Persistente)

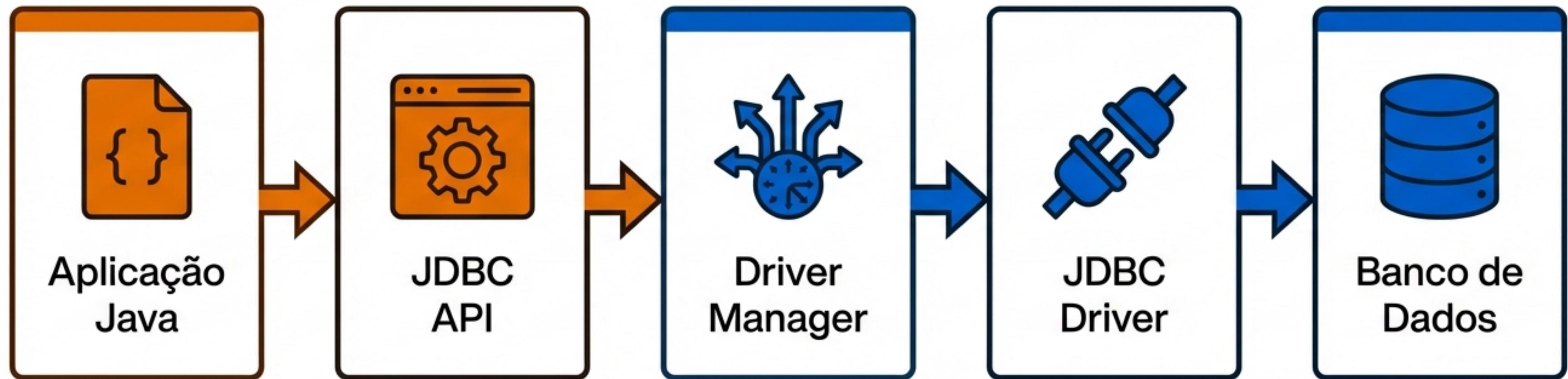
Armazenamento físico em disco. Os dados sobrevivem ao ciclo de vida da aplicação.

Exemplo: Tabela Clientes

Para que uma aplicação Java seja útil no mundo real, ela precisa transferir dados da RAM para o Disco. É aqui que entra a persistência.

A Solução: Java Database Connectivity (JDBC)

O JDBC não é o banco de dados. É uma biblioteca (API) implementada em Java que permite à aplicação enviar comandos SQL para diferentes bancos de dados de forma padronizada.



Escreva em Java, execute em SQL.

O Ecossistema: Bancos e Drivers

O JDBC funciona com “Drivers”. Se você mudar do Oracle para o MySQL, basta trocar o driver (arquivo .jar), sem reescrever todo o código da aplicação.



As Peças Chave da API (java.sql)



DriverManager

A “fábrica” que cria a conexão com o banco.



Connection

Representa o túnel aberto (sessão) com o banco de dados.



PreparedStatement

Objeto responsável por transportar e executar os comandos SQL de forma segura.



ResultSet

A tabela de resultados retornada por uma consulta (SELECT), mantida na memória.

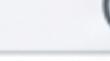
Passo 1: Configurando a Conexão

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
String URL = "jdbc:derby://localhost:1527/bdteste";
Connection con = DriverManager.getConnection(URL, "root", "123");
```

Protocolo (Tecnologia)



Endereço
do Servidor



Porta



Nome do Banco

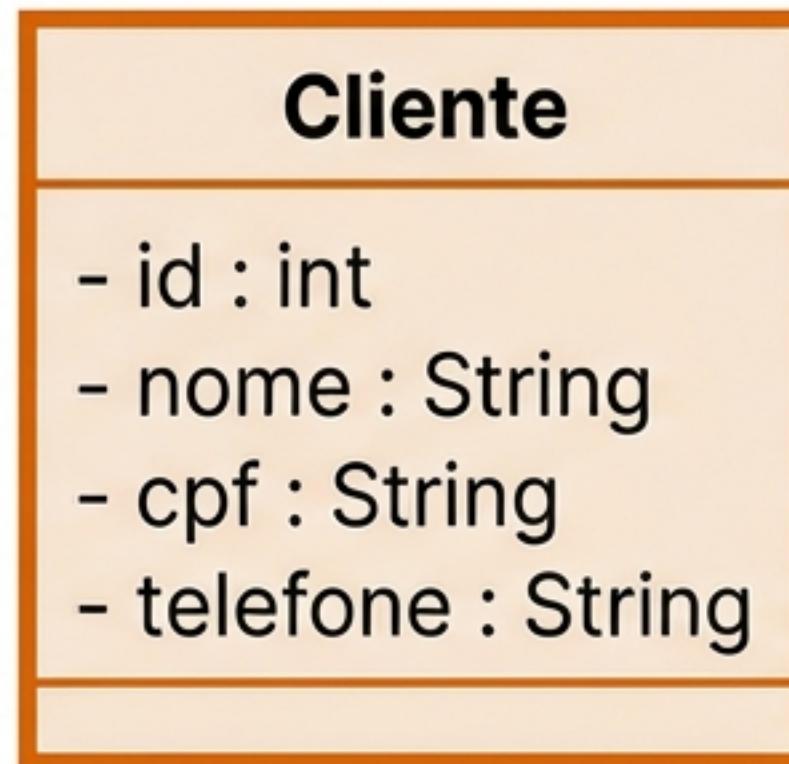
Helvetica Now Text



Nota: O tratamento de exceções (throws ClassNotFoundException, SQLException) é obrigatório.

O Mapeamento: Classe vs. Tabela

Mundo Java (Orientado a Objetos)



Mundo Relacional (SQL)



```
public Cliente(int id, String nome, String cpf, String telefone) {  
    this.id = id; this.nome = nome; this.cpf = cpf; this.telefone = telefone;  
}
```

Inserindo Dados (INSERT)

```
String sql = "INSERT INTO CLIENTES (nome, cpf, telefone) VALUES (?, ?, ?)";  
PreparedStatement ps = connection.prepareStatement(sql);  
  
Cliente cliente = new Cliente("Cliente 4",  
    "444.444.444-44", "(44) 4444-4444");  
  
ps.setString(1, cliente.getNome());  
ps.setString(2, cliente.getCpf());  
ps.setString(3, cliente.getTelefone());  
ps.execute();
```

Wildcards: Placeholders
que previnem SQL Injection
e erros de sintaxe.

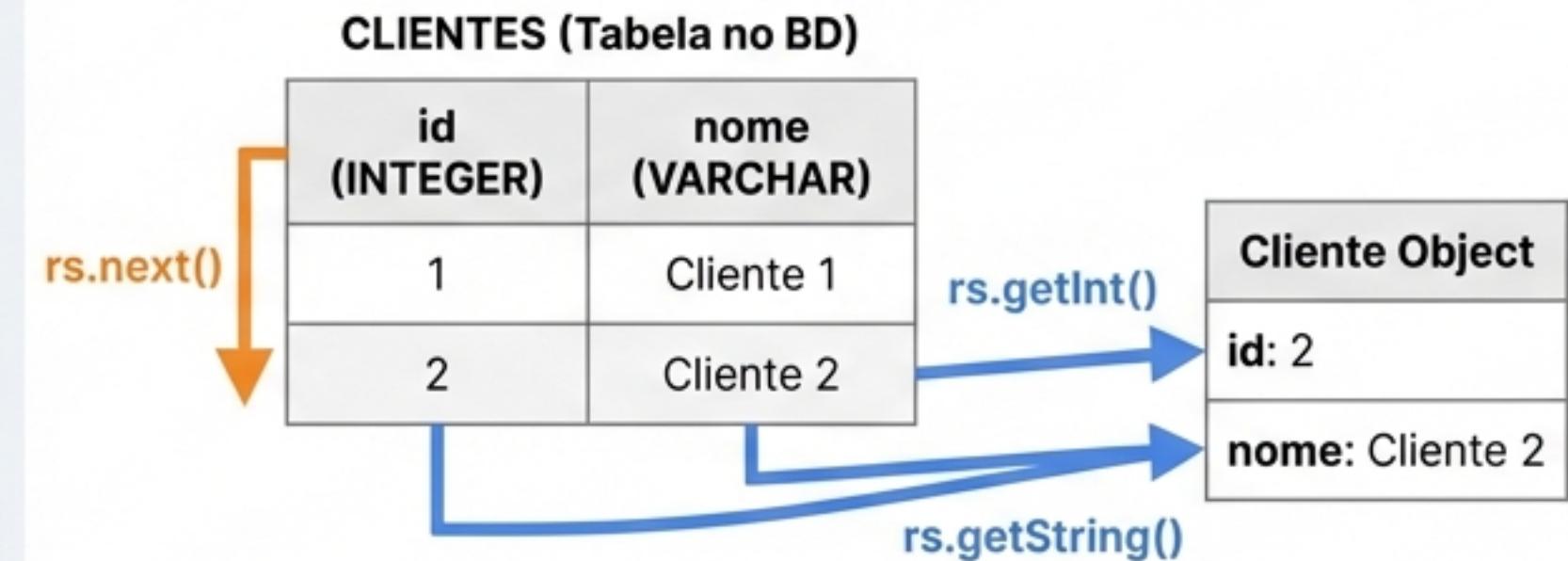
O método setString substitui
as interrogações pelos dados
reais do objeto.

Recuperando Dados (SELECT)

```
ResultSet rs = ps.executeQuery();

while(rs.next()) {
    Cliente c = new Cliente();
    c.setId(rs.getInt("id"));
    c.setNome(rs.getString("nome"));
    System.out.println(c);
}
```

Fluxo do ResultSet



```
...
>
id=1, nome=Cliente 1, cpf=111...
id=2, nome=Cliente 2, cpf=222...
>
```

Atualizando e Removendo (UPDATE / DELETE)

```
String sql = "UPDATE CLIENTES SET nome=?, cpf=? WHERE ID=?";  
PreparedStatement ps = connection.prepareStatement(sql);  
// ... setStrings ...  
ps.setInt(3, cliente.getId());  
ps.execute();
```

Crítico: Sem a cláusula WHERE, o comando alteraria TODOS os registros da tabela.

O comando DELETE segue a mesma lógica:

```
String sql = "DELETE FROM CLIENTES WHERE ID=?";
```

O Problema do Código Misturado

Acoplamento Alto

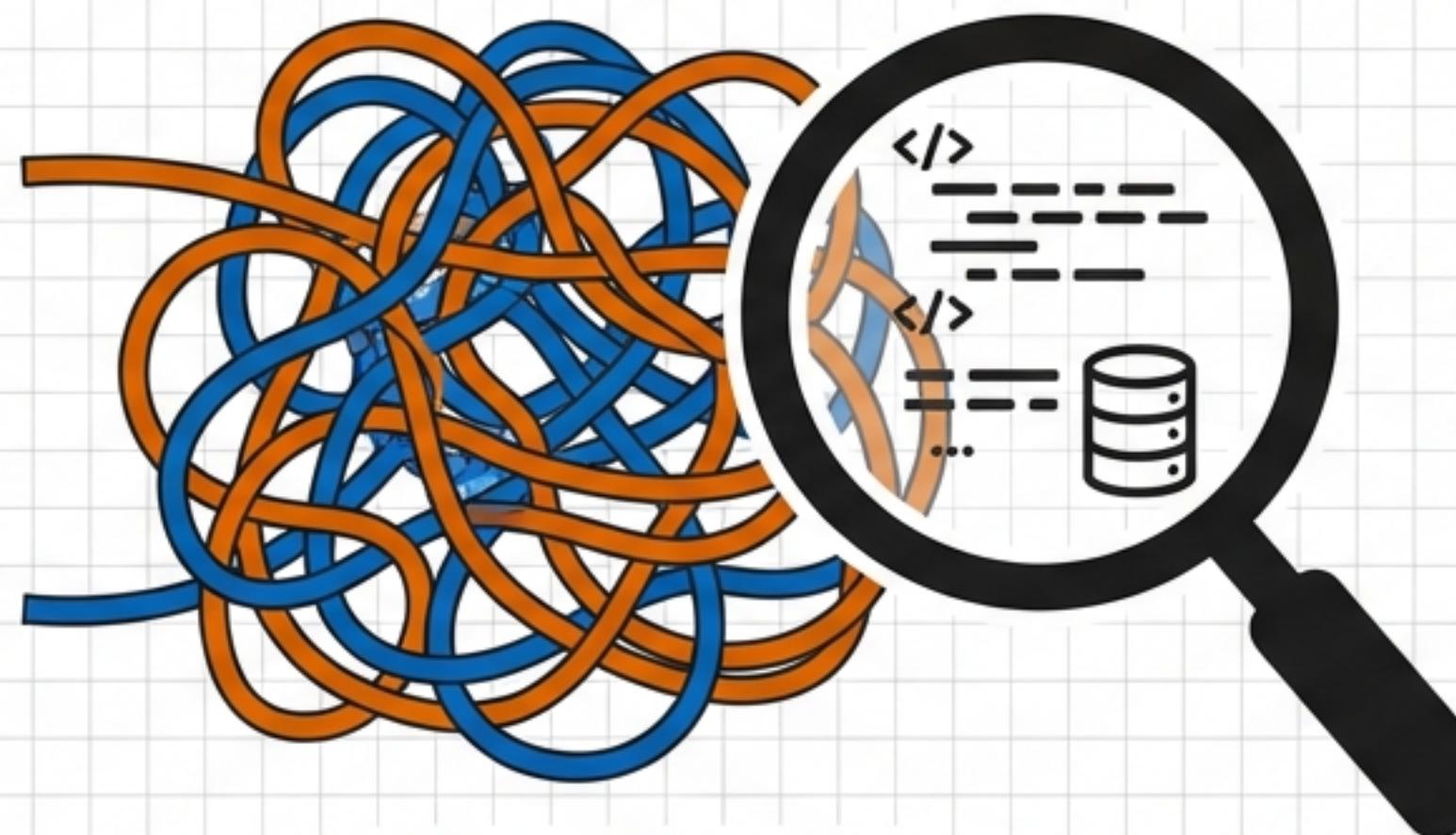
SQL misturado com a interface gráfica ou regras de negócio.

Manutenção Difícil

Alterar uma coluna no banco exige editar dezenas de classes Java.

Repetição

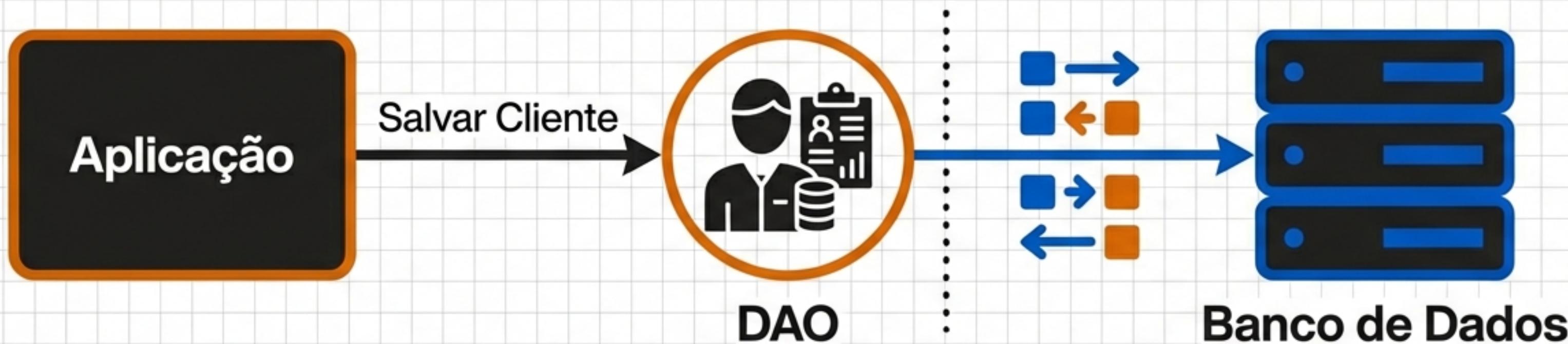
A lógica de conexão (DriverManager...) copiada em múltiplos arquivos.



Precisamos de uma arquitetura melhor para organizar essa bagunça.

A Evolução: Padrão DAO (Data Access Object)

O DAO é um padrão de projeto que abstrai e encapsula o acesso aos dados.



- Separação de Responsabilidades
- Centralização do SQL
- Facilidade de Manutenção

Anatomia da Classe ClienteDAO

```
public class ClienteDAO {  
  
    private Connection connection;  
  
    public boolean inserir(Cliente cliente) { ... }  
  
    public boolean alterar(Cliente cliente) { ... }  
  
    public boolean remover(Cliente cliente) { ... }  
  
    public List<Cliente> listar() { ... }  
}
```

Todo o SQL complexo (INSERT, UPDATE, SELECT) agora vive escondido dentro destes métodos.
A conexão também é gerenciada aqui.

O Resultado: Uma Aplicação Limpa

Antes (JDBC Puro na Main)

```
try {
    Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/teste", "root", "password");
    Class.forName("com.mysql.jdbc.Driver"); // Data forName new Connection("com.mysql.jdbc.Driver");
    String sql = "INSERT INTO clientes (nome, cpf, telefone) VALUES (?, ?, ?)"
    String nome = "Rafael";
    String cpf = "111.222.333-44";
    String telefone = "9999-8888";
    Statement statement = connection.createStatement();
    statement.executeUpdate(sql);
    System.out.println("Cliente inserido com sucesso!");
} catch (SQLException e) {
    e.printStackTrace();
}
```

Depois (Com DAO)

```
Cliente c = new Cliente("Rafael", "111.222...","9999-8888");
ClienteDAO dao = new ClienteDAO();
dao.inserir(c);
```

Código legível,
orientado a objetos e
simples.

Resumo da Jornada

- ✓ **Persistência:** Salvar dados voláteis (RAM) em meio permanente (Banco).
- ✓ **JDBC:** A ponte padrão que conecta o Java a qualquer Banco de Dados.
- ✓ **PreparedStatement:** Essencial para segurança e parâmetros dinâmicos.
- ✓ **Padrão DAO:** Professionaliza o código, isolando a lógica de dados.

Referências:

Deitel, H. M., & Deitel, P. J. - Java: Como Programar.

Material de referência e exemplos de código: Rafael Vargas Mesquita.

Repositório GitHub: github.com/ravarmes/jdbc-exemplos-java