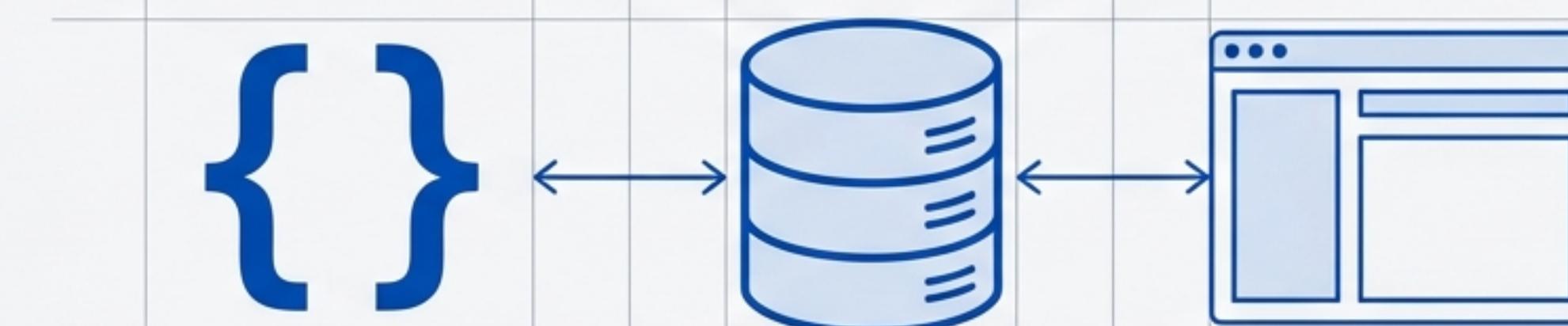


Arquitetura e Construção de Software com JavaFX

MVC, Padrões de Projeto e Integridade Transacional na Prática



Um guia prático para transformar sintaxe Java em aplicações profissionais, reutilizáveis e fáceis de manter.

Framework: Blueprint to Build

“

Conhecer os princípios da Programação Orientada a Objetos não faz de você automaticamente um bom projetista.

— Ward Cunningham & Ralph Johnson

A Necessidade de Padrões



O Problema: Projetos sem estrutura tornam-se difíceis de manter e impossíveis de reutilizar.

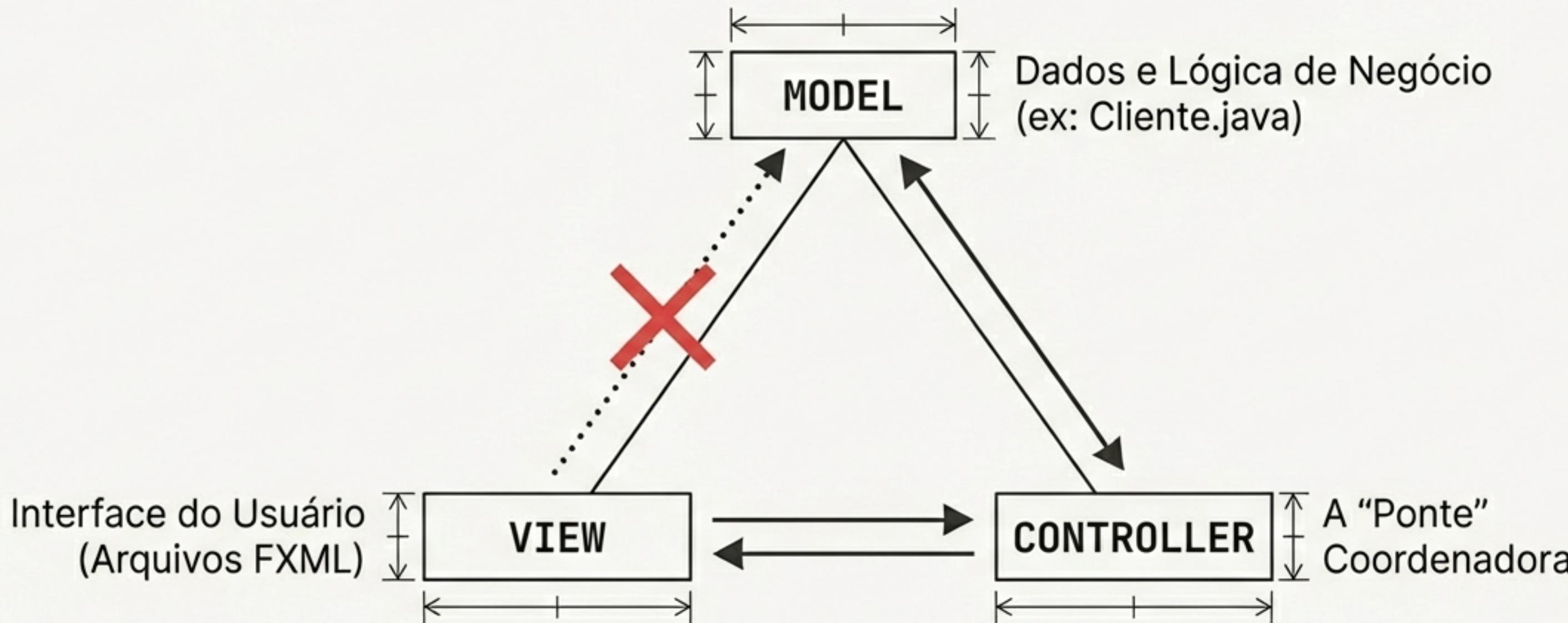


A Solução: Padrões de Projeto (Design Patterns). Soluções descobertas a partir de problemas reais.



Objetivo: Criar sistemas desacoplados (ex: trocar o banco de dados sem quebrar a interface).

A Regra de Ouro: O Princípio da Separação MVC



Mandamentos

1. Não conecte diretamente objetos do Model com a View.
2. Não coloque lógica de aplicação dentro da View.

Padrão 1: O Controlador (The Brain)



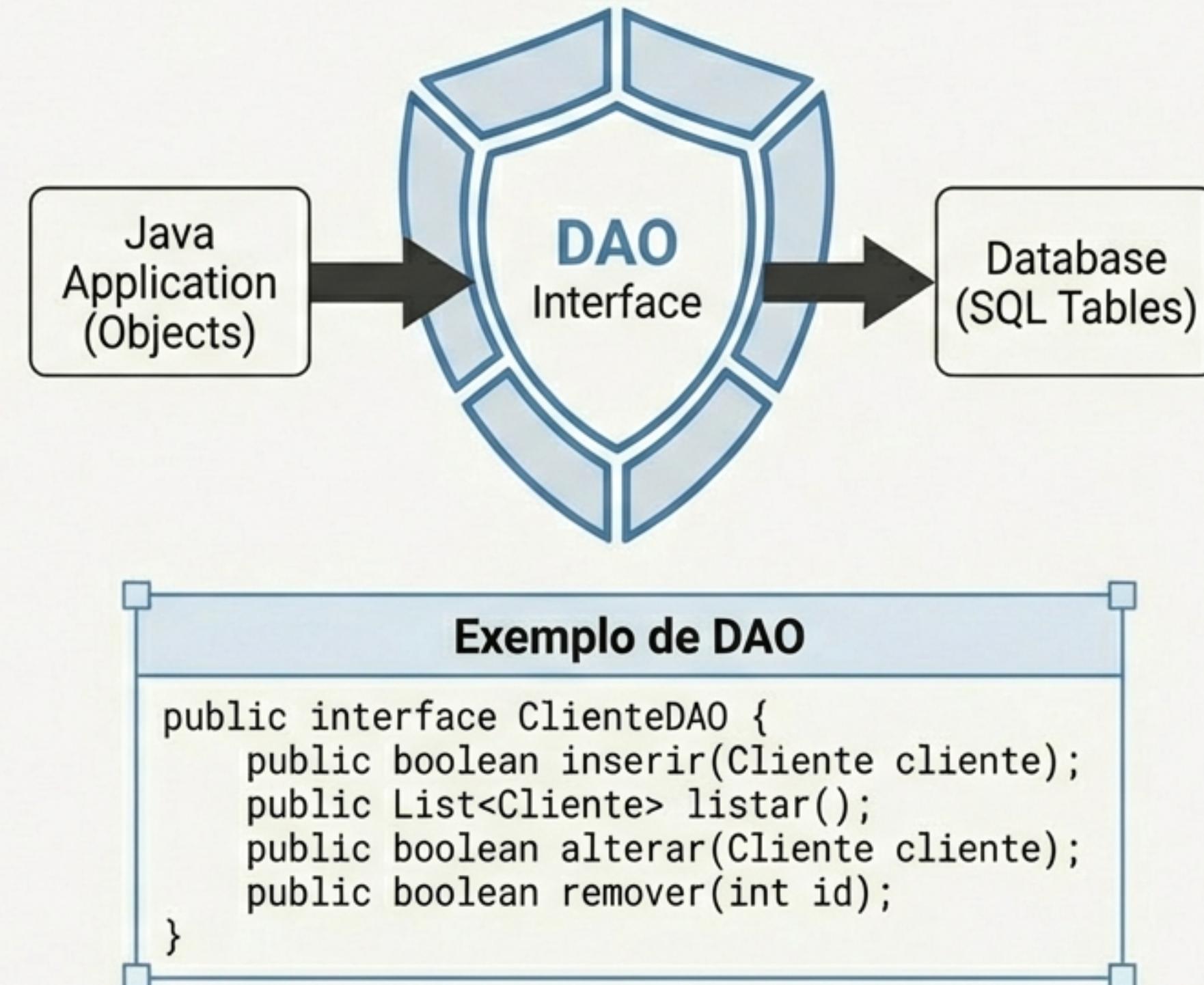
Exemplo de Controle

```
public void handleButtonInserir() {  
    // 1. Recebe da View  
    String nome = txtNome.getText();  
    // 2. Valida  
    if (validar(nome)) {  
        // 3. Chama Lógica  
        clienteDAO.inserir(new Cliente(nome));  
    }  
}
```

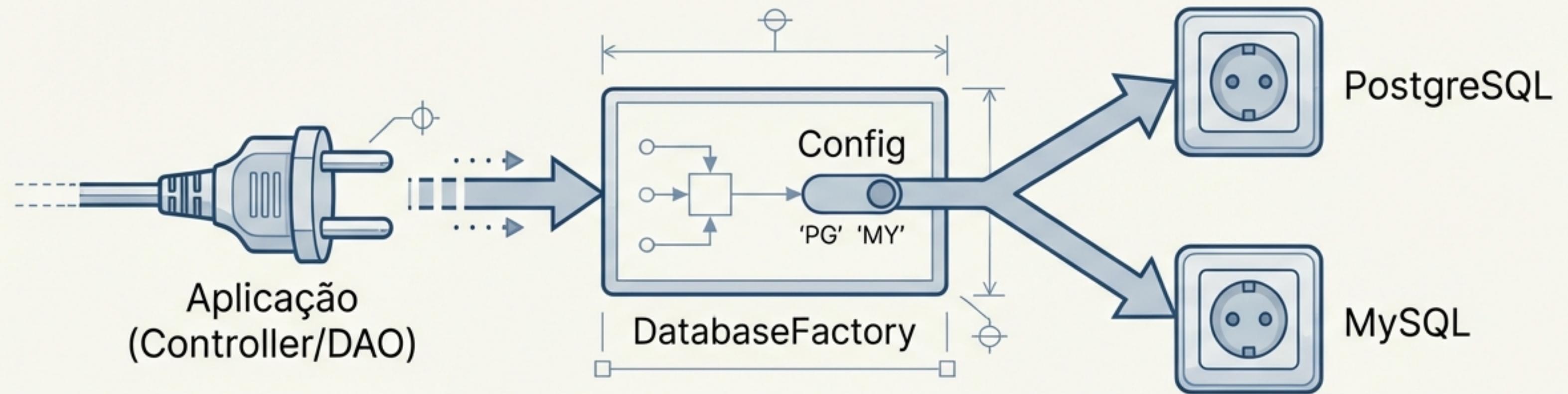
Padrão 2: DAO (Data Access Object)

Definição: Encapsula o acesso à fonte de dados. Isola o SQL das regras de negócio.

Benefício: Permite trocar o banco de dados (ex: PostgreSQL -> MySQL) sem alterar o Controller.



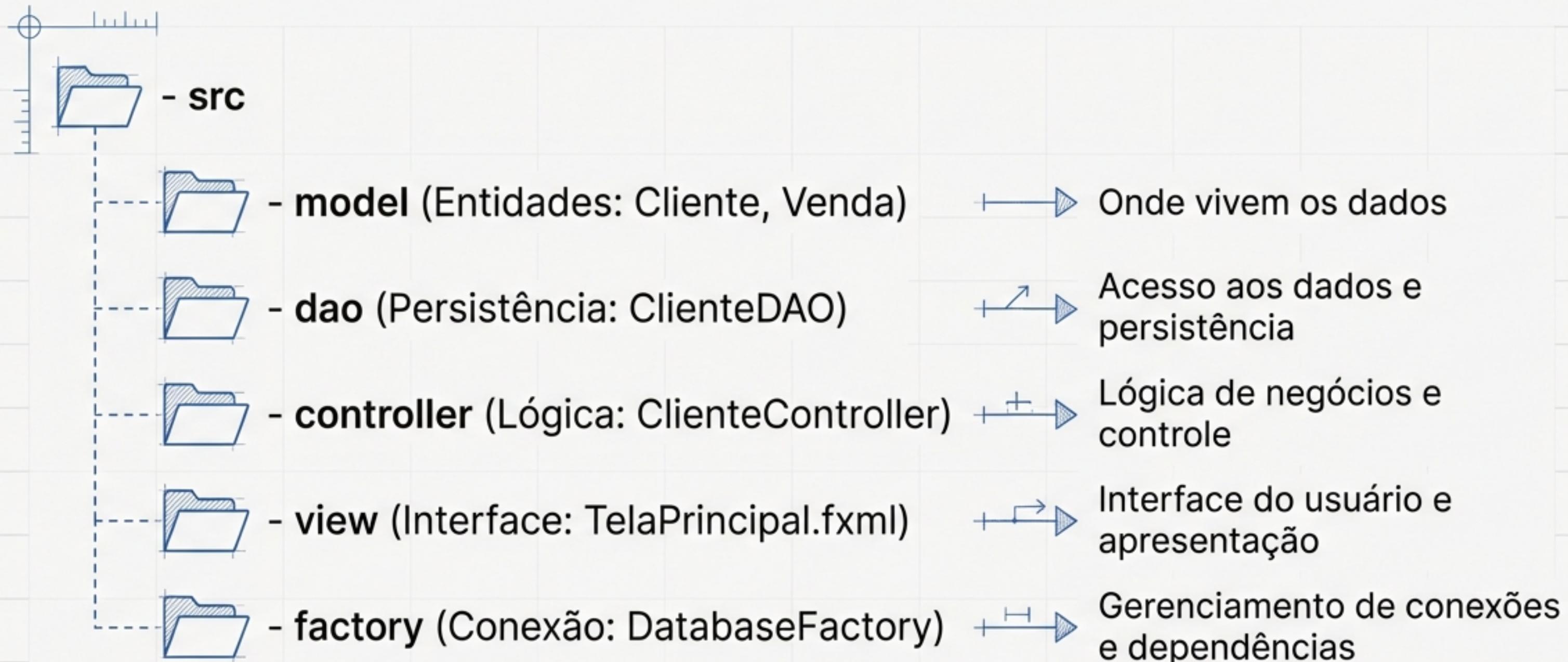
Padrão 3: Factory (Gerenciamento de Dependências)



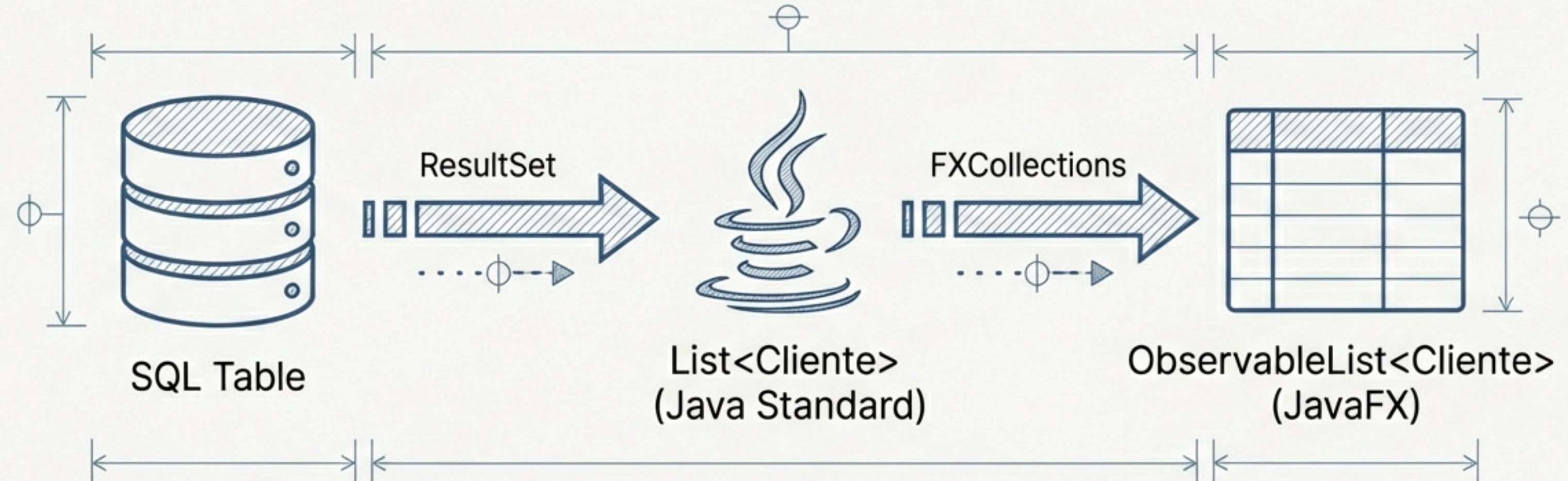
Problema: Hardcoding de credenciais espalhado pelo código.

Solução: Centralizar a criação de conexões. O sistema pede uma conexão genérica (Interface), e a Factory decide qual implementação concreta entregar (Polimorfismo).

Organização do Canteiro de Obras

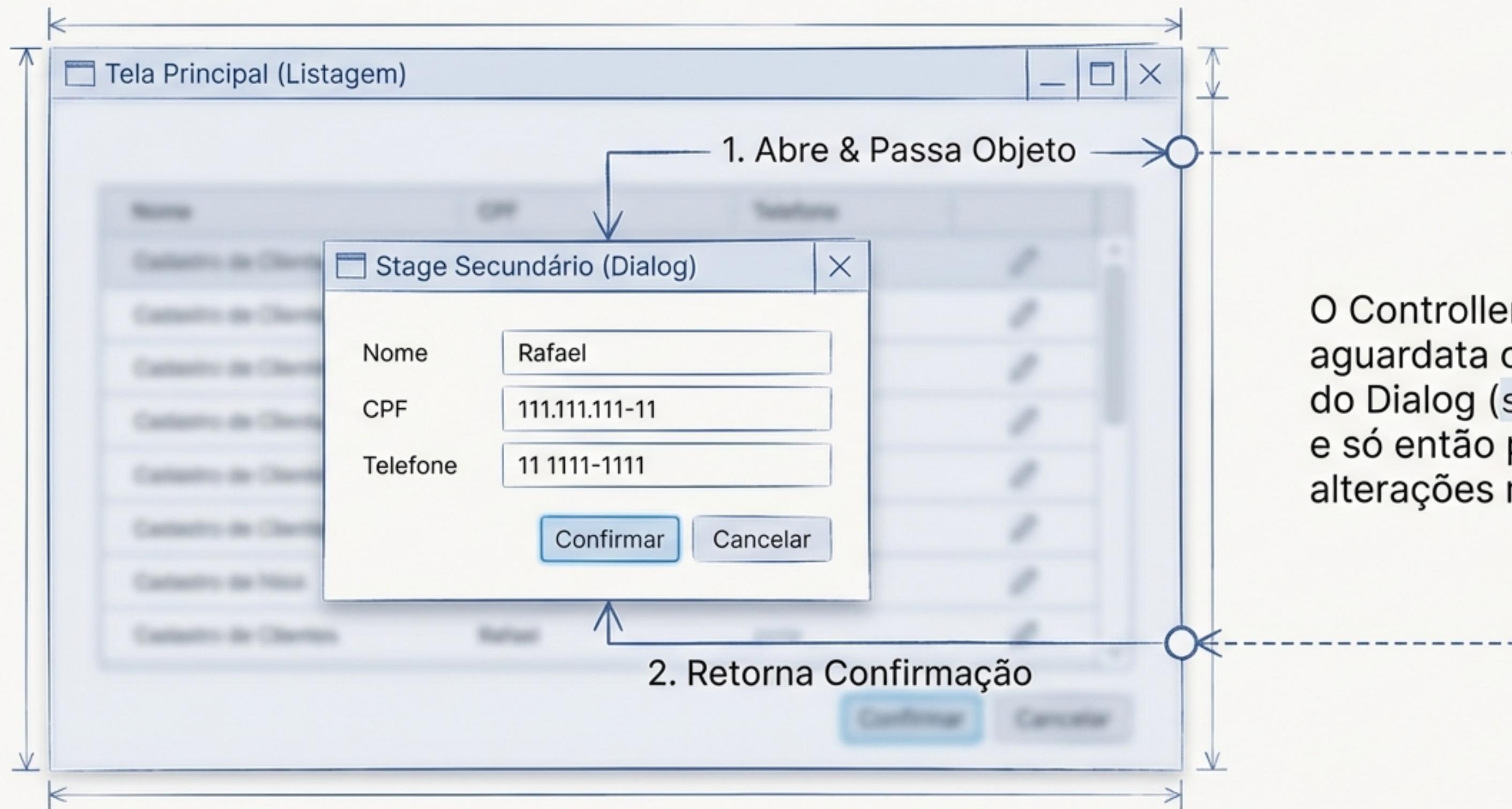


Prática Fase 1: Listagem de Clientes (The Read)

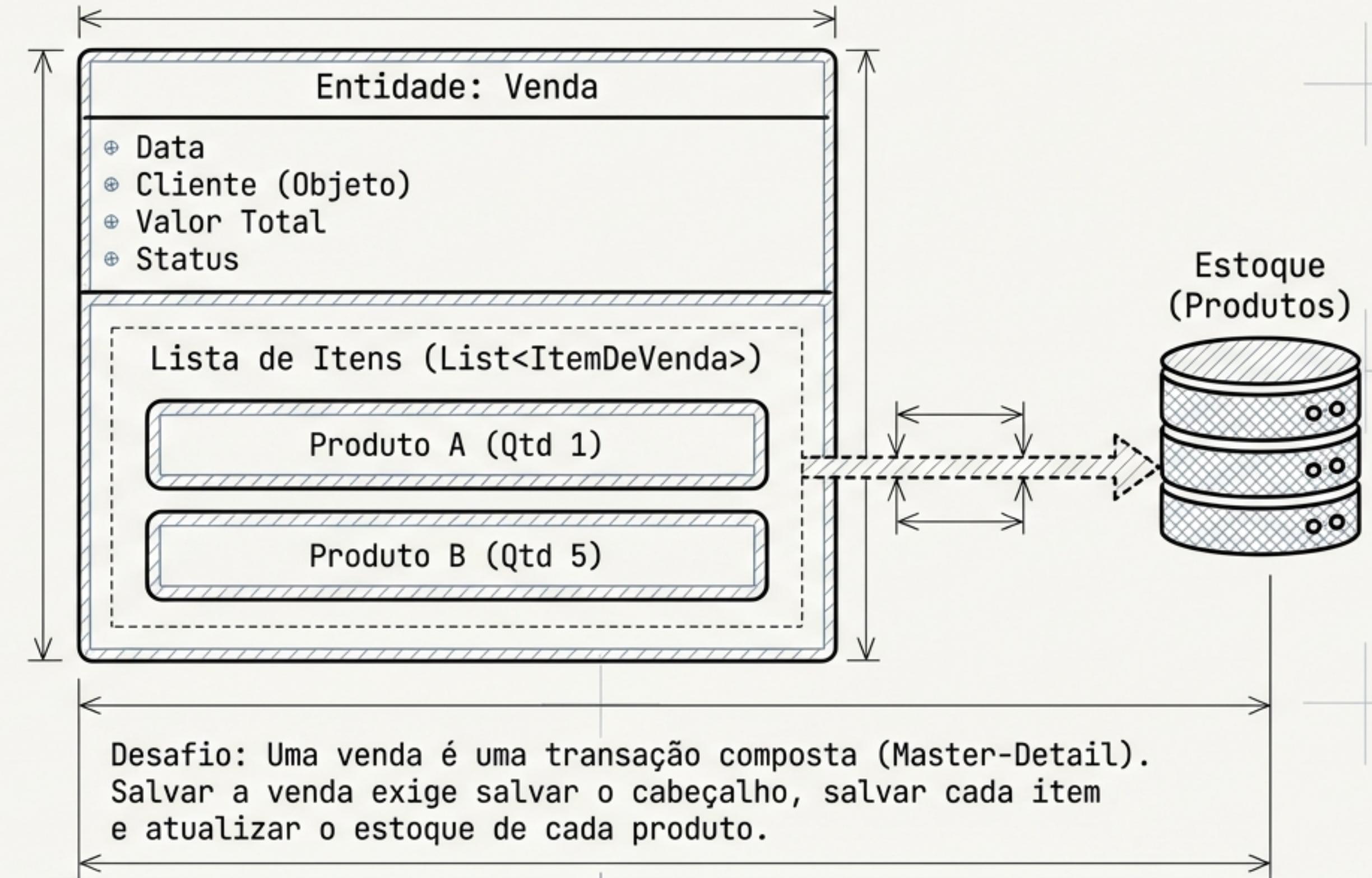


- 1. **DAO**: O método `listar()` retorna uma `List` Java padrão.
- 2. **Controller**: Converte a `List` para `ObservableList` (Requisito do JavaFX).
- 3. **View**: O `TableView` se ‘amarra’ (Bind) à `ObservableList` para atualizações automáticas.

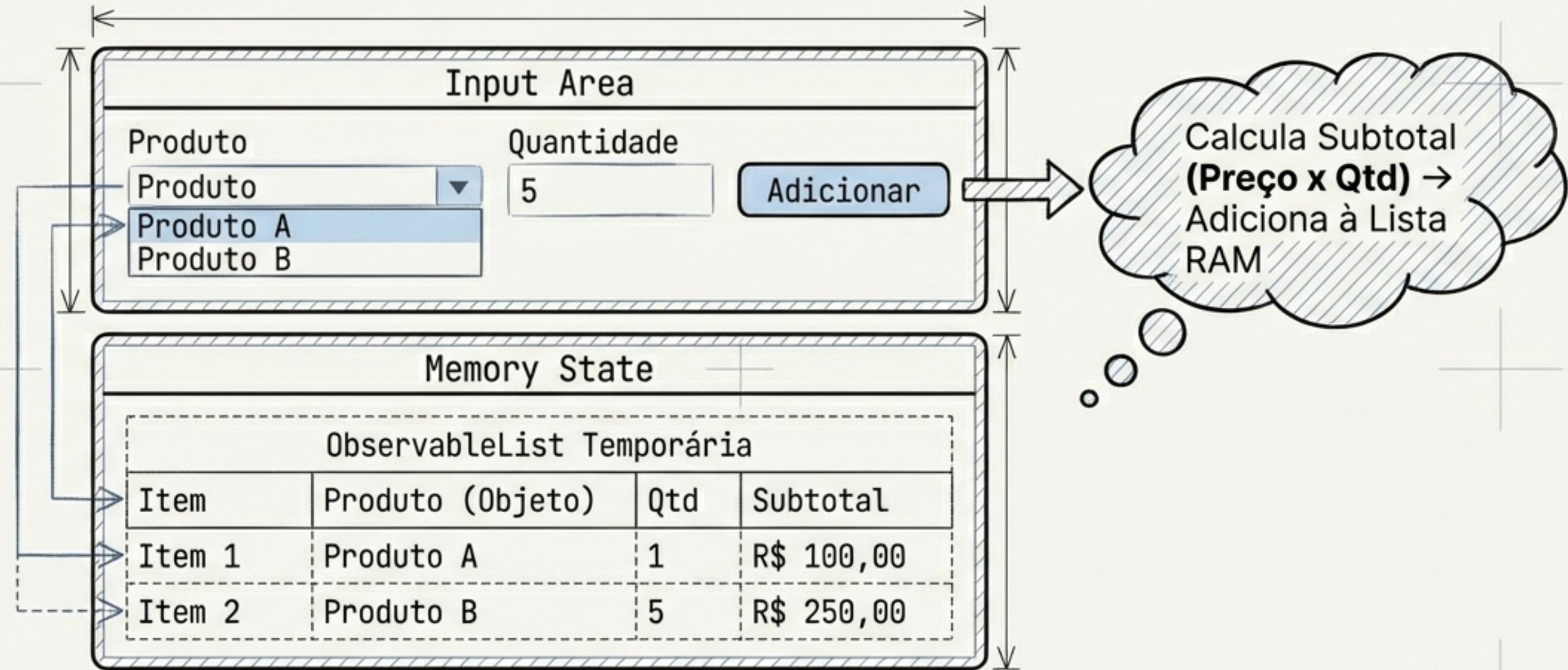
Prática Fase 1: Interação via Dialog (Create/Update)



Prática Fase 2: O Processo de Venda (Complexidade)



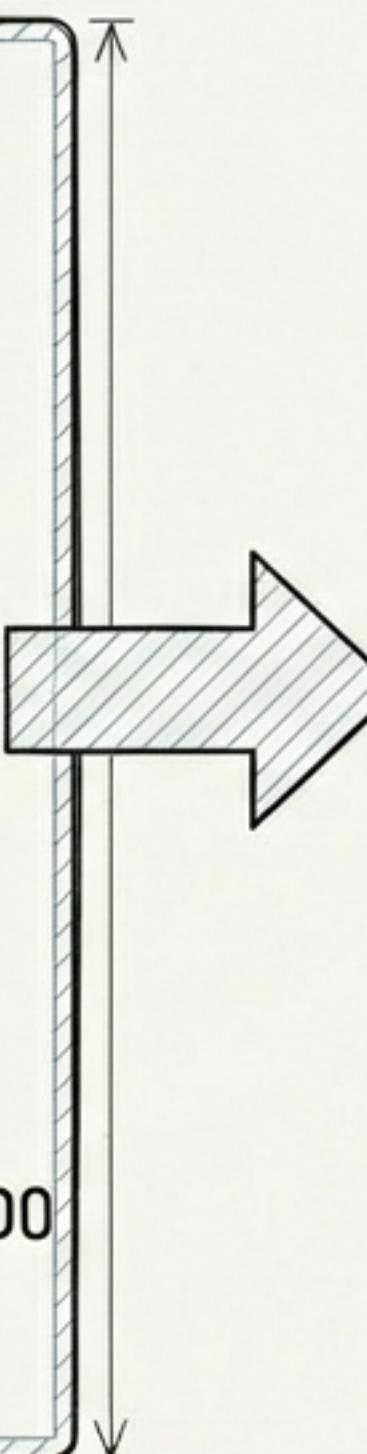
Lógica de Interface: O "Carrinho" em Memória



NENHUM SQL É EXECUTADO AQUI. Toda a manipulação ocorre em objetos na memória até o clique final em 'Salvar'.

Integridade de Dados: Gerenciamento de Transações

```
try {  
    // 1. Desliga o Save Automático  
    connection.setAutoCommit(false);  
  
    // 2. Operações em Cadeia  
    vendaDAO.inserir(venda);  
    for (Item item : itens) {  
        itemDAO.inserir(item);  
        produtoDAO.baixarEstoque(item);  
    }  
  
    // 3. Se tudo deu certo: GRAVA  
    connection.commit();  
} catch (Exception e) {  
    // 4. Se algo deu errado: DESFAZ TUDO  
    connection.rollback();  
}
```

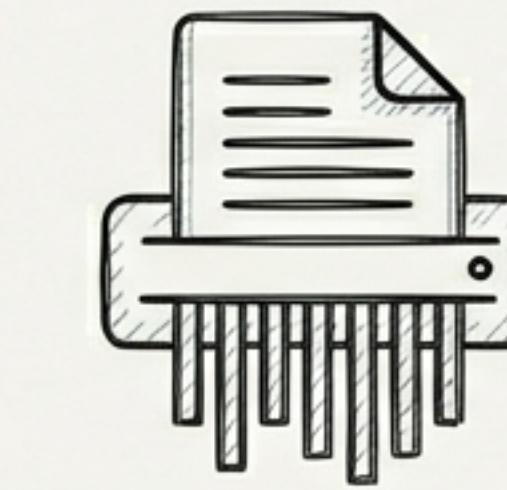
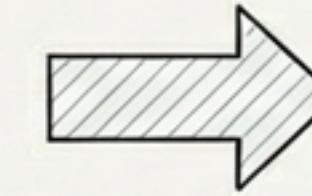
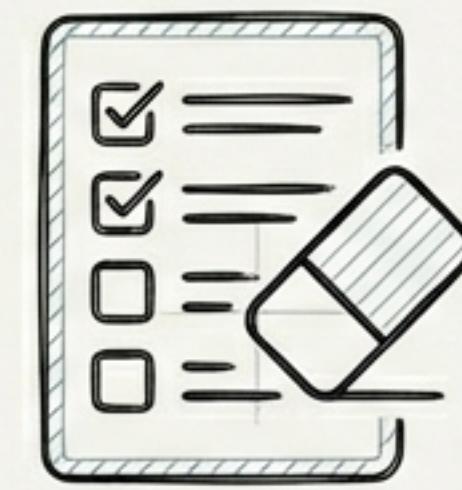
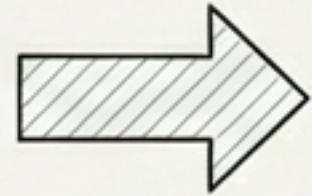
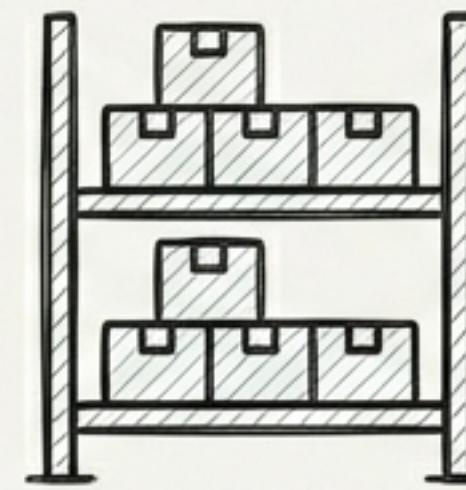


Atomicidade: Ou grava tudo, ou não grava nada.



O Fluxo de Remoção e Restauração de Estoque

Transação Única (Rollback em caso de erro)



1. Restaurar Estoque

Ler itens da venda e somar quantidade de volta aos Produtos.

2. Limpar Detalhes

`Delete from itens_venda
where venda_id = X`

3. Remover Cabeçalho

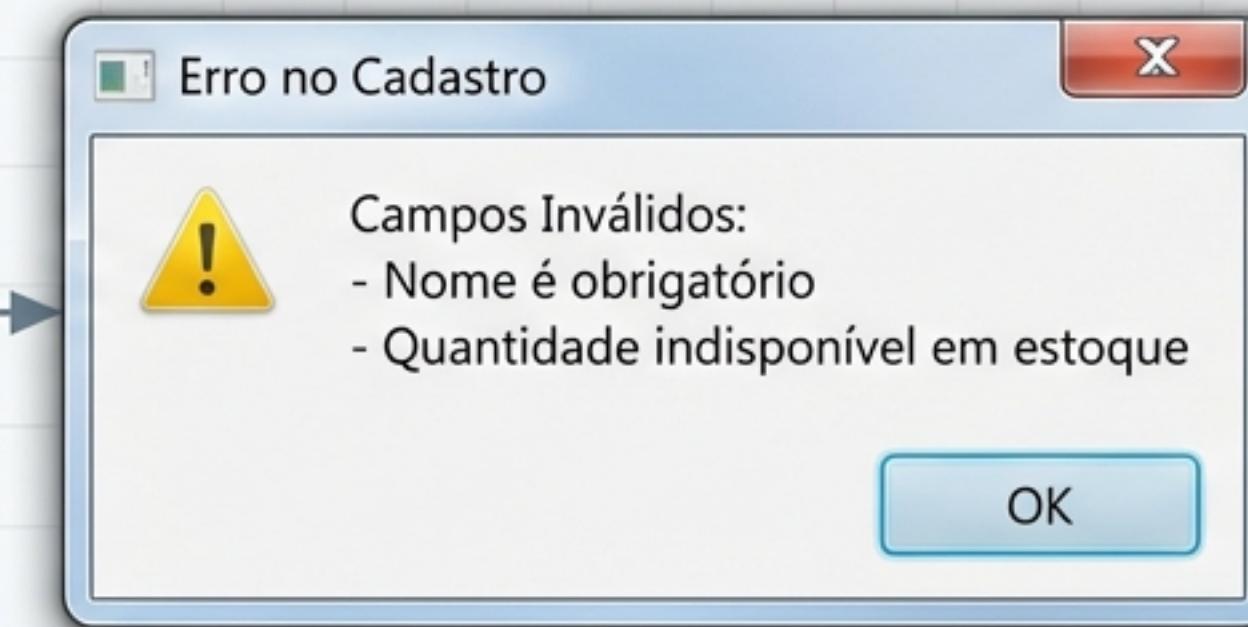
`Delete from vendas
where id = X`

Blindando a Aplicação: Validação e Feedback

Validation

```
if (txtNome.getText().isEmpty()) {  
    errorMessage += 'Nome inválido\n';  
}
```

Feedback



Proteja o banco de dados de entradas nulas ou inconsistentes.

O Resultado: Arquitetura Sustentável

-  **MVC:** Interface (FXML) totalmente desacoplada das Regras de Negócio.
-  **DAO:** Camada SQL isolada e substituível.
-  **Factory:** Flexibilidade para trocar de Database Vendor instantaneamente.
-  **Transações:** Garantia absoluta de integridade (ACID) em operações complexas.

Construir software dá trabalho.
Construir software que dura exige arquitetura.