Atividade 1: Pedidos de Venda

CONCEITUAÇÃO:

Ritual Comum de Processamento

O processamento de pedidos segue um fluxo fixo de três etapas:

Validar o pedido: Verificar dados essenciais (ex.: itens, cliente, origem).

Calcular o total: Somar valores dos itens, aplicando regras específicas de impostos/taxas.

Emitir recibo: Gerar um documento formal conforme o tipo de pedido (ex.: NF-e para nacional, Commercial Invoice para internacional).

Variações: Nacional vs. Internacional

Aspecto	Pedido Nacional	Pedido Internacional
Impostos/Taxas	Impostos internos (ICMS, IPI)	Taxas de importação, custos aduaneiros, câmbio
Formato do Recibo	Nota Fiscal Eletrônica (NF-e)	Commercial Invoice (com dados de exportação)

Justificativa para Herança:

- A herança por especialização é adequada porque:
- O ritual central (Validar → Calcular → Emitir) é invariável, mas a implementação de cada etapa varia conforme a origem do pedido.
- Classes base (ex.: Pedido) podem definir o esqueleto do processo (via Template Method), enquanto subclasses (ex.: PedidoNacional, PedidoInternacional) especializam:
    - Cálculos fiscais.
    - Validações regionais.
    - Formatação de recibos.

- Isso evita duplicação de código do ritual comum e isolas diferenças nas subclasses.

Políticas Extras: Composição

Exemplos de políticas: Frete Expresso, Embalagem para Presente, Seguro contra Avarias, Promoção de Desconto.

Justificativa para Composição:

- As políticas são independentes e combináveis (ex.: um pedido pode ter frete expresso + seguro).
- Herança seria inflexível (explosão de subclasses como PedidoComFreteESeguro).
- Composição permite:
  - Injetar políticas como dependências (ex.: Pedido com IFrete, IEmbalagem).
  - Alterar políticas sem modificar a lógica central do pedido.
  - Cumprir o Open/Closed Principle: novas políticas são adicionadas sem alterar código existente.

Conclusão

Herança organiza as variações inerentes ao tipo de pedido (nacional/internacional).

Composição gerencia políticas acessórias e combináveis (frete, embalagem, etc.), garantindo flexibilidade e baixo acoplamento.

DESIGN:

Contrato da Classe `Pedido` (Base)

```
public abstract class Pedido
{
    // Ritual fixo - NÃO virtual (template method)
    public void Processar()
    {
```

```
        Validar();
        decimal subtotal = CalcularSubtotal();
        decimal total = AplicarPoliticasPlugaveis(subtotal);
        EmitirRecibo(total);
    }

    // Ganchos para especialização (protected virtual)
    protected virtual void Validar()
    {
        // Validações mínimas comuns
        // - Itens não vazios
        // - Dados do cliente presentes
        // - Origem/destino válidos
    }

    protected abstract decimal CalcularSubtotal();
    protected abstract string EmitirRecibo(decimal total);

    // Composição para políticas
    private decimal AplicarPoliticasPlugaveis(decimal subtotal)
    {
        decimal valorAtual = subtotal;

        valorAtual = _frete?.Invoke(valorAtual) ?? valorAtual;
        valorAtual = _embalagem?.Invoke(valorAtual) ?? valorAtual;
        valorAtual = _seguro?.Invoke(valorAtual) ?? valorAtual;
        valorAtual = _promocao?.Invoke(valorAtual) ?? valorAtual;

        return valorAtual;
    }

    // Delegates para políticas plugáveis
    public Func<decimal, decimal>? Frete { set => _frete = value; }
    public Func<decimal, decimal>? Embalagem { set => _embalagem = valu
e; }
    public Func<decimal, decimal>? Seguro { set => _seguro = value; }
    public Func<decimal, decimal>? Promocao { set => _promocao = value; }
```

```
    private Func<decimal, decimal>? _frete;
    private Func<decimal, decimal>? _embalagem;
    private Func<decimal, decimal>? _seguro;
    private Func<decimal, decimal>? _promocao;
}
```

Regras de LSP (Liskov Substitution Principle)

1. Substituibilidade

- **Regra**: Qualquer cliente que usa `Pedido` via `Processar()` deve funcionar identicamente com `PedidoNacional`, `PedidoInternacional`, etc.
- **Garantia**:
  - Não é necessário `is` / `as` / `downcast`
  - Cliente trabalha exclusivamente com tipo base `Pedido`
  - Exemplo de uso:

```
Pedido pedido = new PedidoNacional(); // Ou PedidoInternacional
pedido.Processar(); // Funciona igual para todos os tipos
```

2. Invariantes Preservadas

- **Regra**: Validações mínimas da base não podem ser enfraquecidas
- **Implementação**:
  - Método `Validar()` da base estabelece contrato mínimo
  - Derivadas podem **fortalecer** (adicionar validações) mas não **enfraquecer**
  - Padrão: derivadas chamam `base.Validar()` antes de suas validações específicas

```
protected override void Validar()
{
    base.Validar(); // Preserva invariantes da base
    // Validações específicas da derivada
    ValidarTaxaImportacao();
```

```
    ValidarCambio();
}
```

3. Contratos de Saída Equivalentes

- **Regra**: `Processar()` sempre produz resultado coerente e previsível
- **Garantias**:
  - Sempre retorna recibo formatado corretamente para o tipo de pedido
  - Valor do recibo corresponde exatamente ao total calculado
  - Não introduz exceções inesperadas em relação ao contrato base
  - Exceções específicas são subtipos das exceções documentadas na base

---

Eixos Plugáveis (Delegates)

Política	Delegate	Assinatura	Papel
Frete	`Func<decimal, decimal>`	`decimal → decimal`	Aplica custo de transporte sobre o valor corrente. Ex: `valor ⇒ valor + 15.00`
Embalagem	`Func<decimal, decimal>`	`decimal → decimal`	Adiciona custo de embalagem/proteção. Ex: `valor ⇒ valor + 5.50`
Seguro	`Func<decimal, decimal>`	`decimal → decimal`	Aplica prêmio/percentual de seguro. Ex: `valor ⇒ valor * 1.02` (2%)
Promoção	`Func<decimal, decimal>`	`decimal → decimal`	Aplica desconto/cupom. Ex: `valor ⇒ valor * 0.90` (10% off)

Exemplo de Uso das Políticas:

```
var pedido = new PedidoNacional();
pedido.Frete = valor ⇒ valor + 25.00m;        // Frete fixo
pedido.Seguro = valor ⇒ valor * 1.03m;        // 3% de seguro
pedido.Promocao = valor ⇒ valor - 10.00m;     // Desconto fixo

pedido.Processar(); // Aplica políticas na ordem definida
```

Fluxo de Execução do `Processar()`

1. `Validar()` → Validações básicas + específicas do tipo
2. `CalcularSubtotal()` → Subtotal com impostos/taxas específicos
3. **Políticas Plugáveis** → Aplicadas em sequência:
    - Frete → Embalagem → Seguro → Promoção
4. `EmitirRecibo(total)` → Formato específico com valor final

Resultado: Sistema extensível onde novos tipos de pedido herdam o ritual, e novas políticas compõem-se dinamicamente.