📦 Sistema de E-commerce – Documentação Técnica

🔧 Arquitetura da Solução

O sistema foi projetado com base em uma arquitetura modular, separando responsabilidades em domínios bem definidos:

- Catálogo de Produtos

- Utiliza uma classe abstrata Produto, com subclasses específicas como ProdutoEletronico e ProdutoLivro.

- Cada produto implementa seu próprio cálculo de preço final e imposto, promovendo polimorfismo.

- Carrinho de Compras

- Gerencia itens genéricos de produto com a classe CarrinhoCompras.

- Implementa a interface ISerializable para representar o carrinho em forma de string (simulação de serialização).

- Calcula o total de forma dinâmica com base nas regras de cada produto.

- Sistema de Pagamento

- Usa a interface IProcessadorPagamento, implementada por classes como PagamentoCartao.

- Utiliza IDisposable para simular o controle e liberação de recursos externos.

- Integra com o carrinho e processa valores de forma genérica.

- Sistema de Notificação

- Baseado na interface INotificador, com implementações como NotificadorEmail.

- Fornece feedback ao cliente após a conclusão do pedido.

- Programa Principal

- Demonstra o uso integrado dos módulos, simulando um fluxo real de compra com adição ao carrinho, pagamento e notificação.

🧱 Princípios de POO Aplicados

| Princípio | Aplicação |

|---|---|

| Encapsulamento | Atributos e métodos encapsulados nas classes; carrinho protege sua lista interna. |

| Herança | Produto é classe base abstrata, herdada por tipos específicos de produtos. |

| Polimorfismo | Métodos CalcularPrecoFinal() e ProcessarPagamento() com comportamento variado por tipo. |

| Interfaces | Uso de INotificador, ISerializable e IProcessadorPagamento para garantir baixo acoplamento. |

| Disposable | PagamentoCartao implementa IDisposable para simular liberação de recursos. |

⚠️ Desafios Encontrados e Soluções

- Gerenciamento de diferentes tipos de produtos no carrinho

- Desafio: Cada produto tem regras diferentes de cálculo de preço e imposto.

- Solução: Uso de herança e métodos abstratos permitiu que cada tipo de produto implementasse seu próprio comportamento.

- Integração de componentes com baixo acoplamento

- Desafio: Garantir que o carrinho, pagamento e notificação funcionem juntos sem dependências diretas.

- Solução: Interfaces (INotificador, IProcessadorPagamento) permitem que módulos dependam de abstrações e não de implementações.

- Simulação de recursos externos (como banco e envio de e-mails)

- Desafio: Demonstrar uso de recursos controlados mesmo sem conexões reais.

- Solução: Implementação de IDisposable e mensagens simuladas no console demonstram o controle do ciclo de vida.