O presente projeto tem como objetivo implementar um sistema básico de gerenciamento de clientes, produtos e estoque, com o público alvo focado em gerentes de lojas de pequeno porte ou até mesmo pequenas empresas que visam a venda de produtos online. O código será explicado e terá alguns aspectos destacados a fim de demonstrar os quatro pilares da Programação Orientada a Objetos e seus respectivos impactos.

Mas afinal, quais são os quatro pilares da Programação Orientada a Objetos?  
Eles são quatro “conceitos” que devem ser seguidos para criar um projeto com códigos mais robustos e eficientes dentro da medida do possível, são elas o encapsulamento, herança, polimorfismo e abstração.   
O encapsulamento protege os dados internos de uma classe e controla seu acesso através de métodos. Isso aumenta a segurança e facilita a manutenção do código, evitando que dados sejam alterados de forma inesperada.

A herança permite reutilizar código, facilitando a criação de novas classes baseadas em classes existentes. Isso reduz a redundância e melhora a organização do código.

O polimorfismo permite que diferentes classes utilizem uma interface comum, mas implementem comportamentos específicos. Isso aumenta a flexibilidade e a capacidade de extensão do sistema, permitindo que o mesmo código trate de diferentes tipos de objetos de maneira uniforme.

Já a abstração, simplifica a complexidade do sistema, escondendo detalhes desnecessários e expondo apenas o que é essencial. Isso torna o código mais limpo e mais fácil de entender e manter.

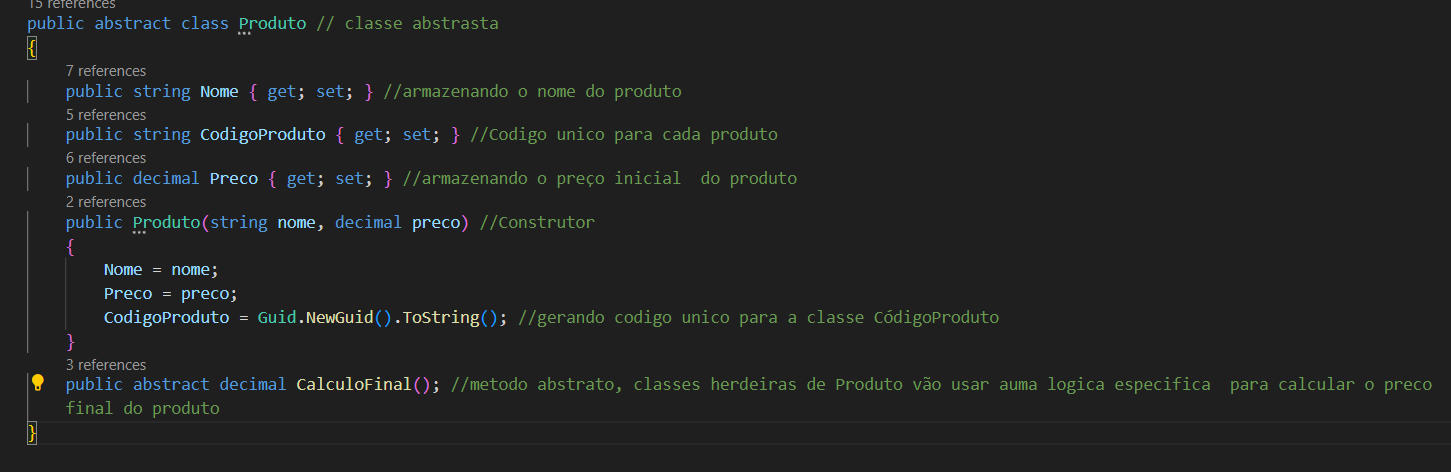
Mas agora vamos entender como o código foi estruturado e quais os seus pontos-chave para obter os resultados desejados, demonstrando exemplos dos pilares mencionados acima.

Podemos começar demonstrando os exemplos dos quatro pilares da POO citados acima.

Na classe Produto.cs temos um exemplo de um dos pilares da Programação Orientada a objetos, a abstração.

A abstração pode simplificar um processo complexo ao definir classes e interfaces que representam objetos do mundo real, assim como o ProdutoFisico.cs, o mesmo representa um objeto físico que possui peso, tamanho e valor.

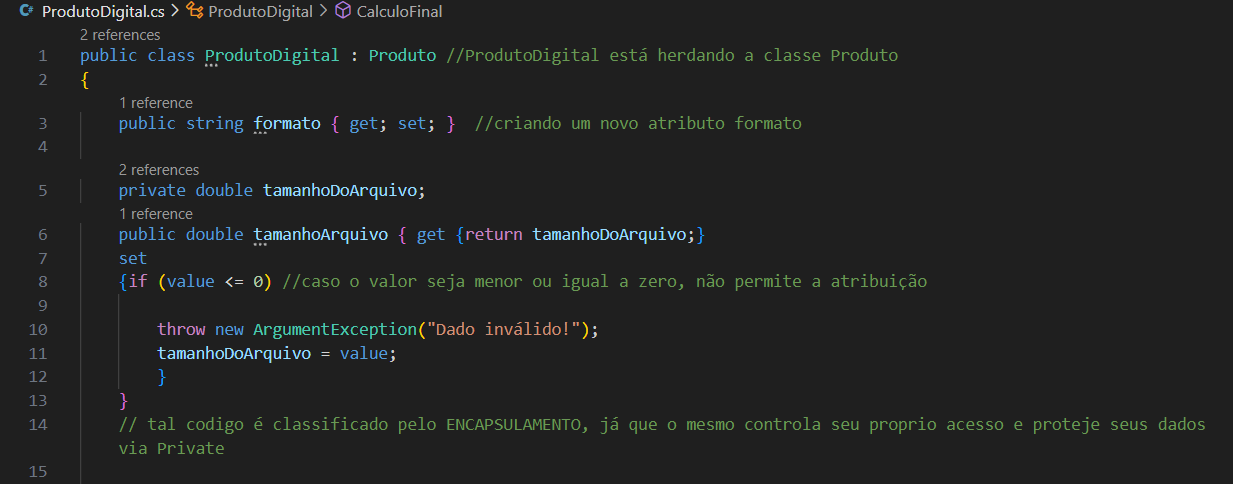
A classe Produto na linha 1 permite armazenar o nome do objeto, preço e um código único como identificador, logo em seguida podemos observar que há uma linha de código também abstrata, porém a classe CalculoFinal não possui implementação na classe base, onde classes derivadas devem fornecer uma lógica específica para calcular o preço final do produto.

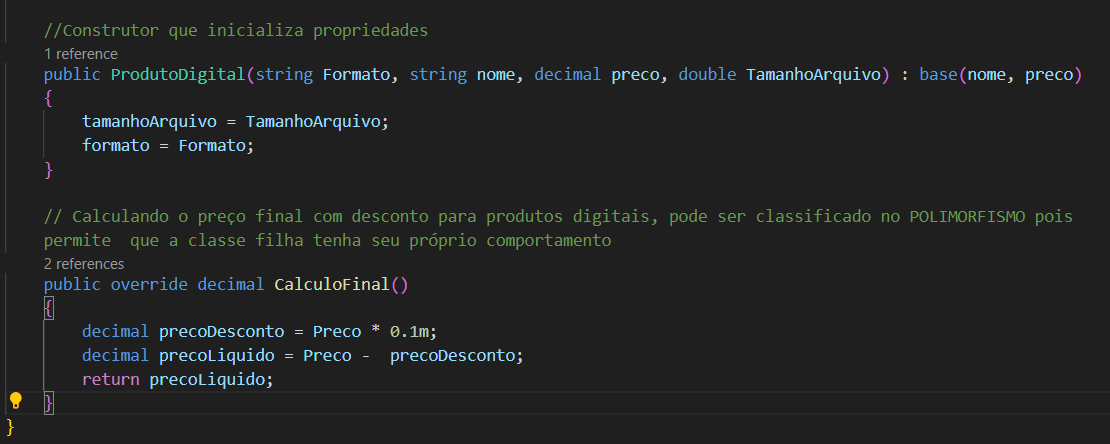


Já em ProdutoDigital.cs em específico temos exemplos de três dos quatro pilares da Programação Orientada a Objetos.

Já de primeira vista percebemos que a classe ProdutoDigital está herdando da classe Produto. Logo em seguida temos o encapsulamento demonstrado a partir da linha 6, tal código está protegendo seu conteúdo e mostrando apenas o necessário e garantindo que não sejam adulterados. Seguindo para o polimorfismo, onde sobrescreve um método da classe base para calcular o preço final do produto digital, implementando uma lógica específica para calcular o desconto.

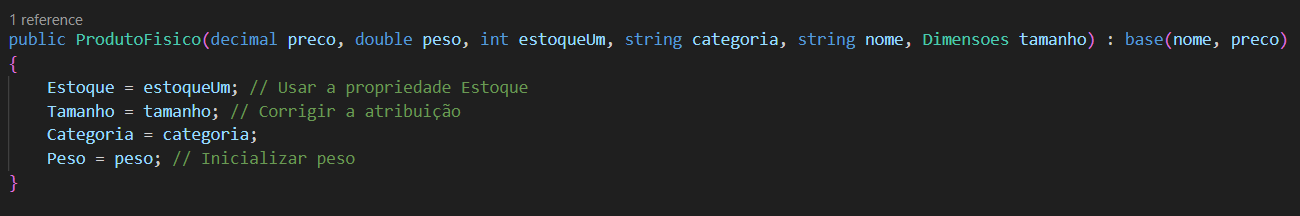
Em resumo, o código organiza e protege dados, reutiliza funcionalidades existentes, e permite que a classe derivada tenha um comportamento específico e diferente da classe base.



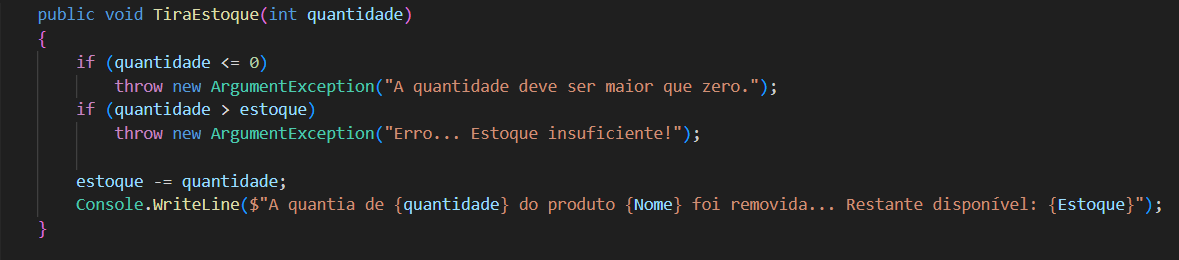


Em relação a ProdutoFisico.cs, não há muitas diferenças em relação ao ProdutoDigital.cs, mas vale a pena mencionar algumas.

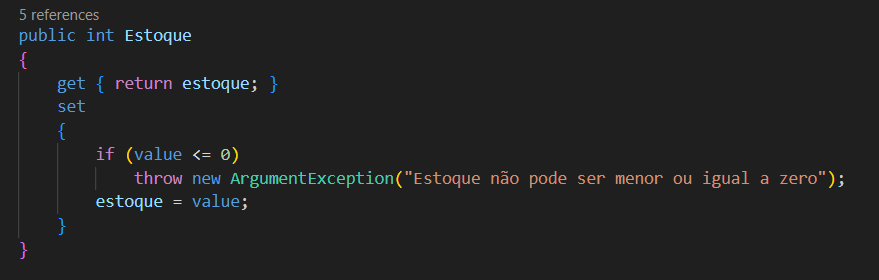
A estrutura de dimensões apresenta uma sequência de "string" e "decimal" onde é possível implementar características do produto direto na Program.cs, onde a ordem de fatores que veremos no final é decidida neste Struct.



A classe TiraEstoque é implementada para remover uma quantidade válida do estoque, caso seja imposto um valor abaixo de zero ou acima do que informado no estoque, uma mensagem informando tal erro será apresentada.



Para finalizar temos alguns métodos que usamos para manter a manipulação de dados do estoque sob controle, para que assim não sejam retirados e postos valores irreais.



Temos a classe Cliente.cs, pela qual não há muito o que comentar, ela segue uma sequência comum de cadastro de nome, identificação, endereço e meios de contato para cada cliente registrado, estas informações serão imprimidas ao final do projeto.

A classe Loja é composta por diversos métodos de cadastro, consulta, listagem, exclusão e criação de pedido, esta classe é tida como a parte backend onde o usuário mais interage, é por ela que alterações em todo o sistema são efetuadas.

E por último, a classe Program.cs é responsável pelo fluxo de movimentação da loja, imprimindo e listando clientes, produtos e estoque disponível.

Na linha 2 por exemplo, exibido como -"var produtoFisico = new ProdutoFisico(80, 5, 7, "Periféricos", "Teclado", new ProdutoFisico.Dimensoes (5, 10, 2));"- podemos identificar uma série de dígitos pela qual a ordem foi decidida por outra classe, sendo respectivamente preço -> peso -> estoque base -> categoria -> nome do produto -> dimensões do produto, onde fora inicializadas tais propriedades ao criar uma nova instância da classe ProdutoFisico.

Este tipo de processo é utilizado também na linha 5, 7 e 8 pela qual chama métodos e parâmetros de uma variável de outras classes, onde mais abaixo no código é visível linhas que imprimem tais resultados.

