UNIVERSIDADE LUTERANA DO BRASIL

ADS

MATHEUS MAGNUS MARCILIO

SISTEMA DE LOJA EM C#

TORRES

2024

MATHEUS MAGNUS MARCILIO

SISTEMA DE LOJA EM C#

Trabalho apresentado à UNIVERSIDADE LUTERANA DO BRASIL como requisito para conclusão do curso de Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Lucas Fogaça

TORRES

2024

A programação orientada a objetos (POO) é um paradigma de programação utilizado no desenvolvimento de software moderno. Sua aplicação permite a criação de sistemas modulares, reutilizáveis e de fácil manutenção, o que é essencial em projetos de maior escala. Entre os principais conceitos da POO estão o encapsulamento, a herança, o polimorfismo e a abstração, os quais são fundamentais para a construção de sistemas mais eficientes e organizados. Neste trabalho, será apresentado um sistema de gerenciamento de produtos, clientes e pedidos, desenvolvido em linguagem C#, que exemplifica o uso destes conceitos. O sistema descrito é projetado para uma loja, permitindo o cadastro e a manipulação de produtos físicos e digitais, além de oferecer funcionalidades para a gestão de clientes e pedidos.

A classe abstrata “Produto” é o núcleo da hierarquia de produtos no sistema. Ela define as propriedades e métodos comuns que serão compartilhados entre os diferentes tipos de produtos, tais como “ProdutoFisico” e “ProdutoDigital”. Entre as propriedades, destacam-se o Nome, o Código e o Preço, que são essenciais para identificar e precificar os itens. O Código do produto é único e pode ser gerado automaticamente ou atribuído manualmente, enquanto o Preço refere-se ao valor base do produto antes de qualquer ajuste.

Além das propriedades, a classe Produto define o método abstrato CalcularPrecoFinal(), que obriga as classes derivadas a implementar sua própria lógica de cálculo do preço final do produto. Este método considera diferentes fatores, como descontos, impostos e, no caso de produtos físicos, custos de envio. A escolha de declarar Produto como uma classe abstrata é justificada pela necessidade de garantir que produtos específicos implementem uma lógica própria para o cálculo do preço, promovendo o uso do polimorfismo.

A classe ProdutoFisico herda de Produto e estende suas funcionalidades ao incluir propriedades específicas para produtos físicos. Entre elas, estão o Peso, as Dimensões (que neste caso foi criado uma classe com as propriedades de Altura, Largura e Profundidade) e a Categoria, que define o tipo de produto, como “Eletrônicos” ou “Livros”.

O método CalcularPrecoFinal() é implementado em ProdutoFisico e inclui a adição de impostos e custos de envio ao preço base. O cálculo de impostos pode ser definido por uma taxa fixa, como 10%, enquanto os custos de envio

podem ser calculados com base no peso do produto. Esse método também pode aplicar descontos promocionais, caso existam. O encapsulamento é garantido ao proteger as propriedades do produto contra modificações externas, assegurando a integridade dos dados.

Já a classe ProdutoDigital herda de Produto e representa itens digitais, como arquivos de mídia ou documentos. Suas propriedades específicas incluem o TamanhoArquivo e o Formato, que definem o tamanho e o tipo de arquivo, respectivamente. Como produtos digitais não possuem peso nem requerem envio físico, o método CalcularPrecoFinal() implementado nessa classe ignora custos de envio e pode aplicar descontos especiais para este tipo de item. Além disso, em muitos casos, produtos digitais são isentos de impostos, o que simplifica o cálculo do preço final.

Assim como em ProdutoFisico, a validação é importante em ProdutoDigital, garantindo que os valores atribuídos, como o tamanho do arquivo, sejam sempre positivos e coerentes com a natureza do produto. Essa abordagem reforça o princípio de encapsulamento ao proteger as propriedades e garantir sua consistência.

A classe Cliente gerencia as informações de cada consumidor que interage com a loja. Suas propriedades incluem o Nome (que foi utilizado o comando “IsNullOrwhiteSpace, para verificar se o valor é nulo, vazio ou contém apenas espaços em branco ”), o NumeroIdentificacao, o Endereco e o Contato. Essas informações são fundamentais para a realização de pedidos e para o gerenciamento de dados dos clientes. O método ExibirInformacoes() permite exibir essas informações de forma organizada, facilitando o acesso e a manipulação dos dados dos clientes.

As validações podem ser aplicadas nos métodos set das propriedades, garantindo que os dados inseridos, como o nome e o contato, sejam válidos e completos.

A interface ICarriavel define os métodos que são essenciais para a manipulação de produtos em um pedido. Entre os métodos definidos estão AdicionarProduto(Produto produto), RemoverProduto(Produto produto) e

CalcularTotal(). A interface garante que qualquer classe que implemente seus métodos será capaz de lidar com produtos de forma consistente e padronizada.

No sistema descrito, a classe Pedido implementa a interface ICarriavel, fornecendo a lógica necessária para adicionar e remover produtos do pedido, além de calcular o valor total dos itens incluídos. Essa separação de responsabilidades é um exemplo do uso de polimorfismo, onde diferentes implementações podem ser usadas para gerenciar produtos de maneiras variadas, sem alterar a interface.

A classe Pedido gerencia as interações entre produtos e clientes. Suas propriedades incluem o Cliente, a DataPedido (onde foi declarado o tipo DateTime, tipo esse que se adequa a propriedades de data), o Status (onde foi utilizado o tipo “enum”, para atribuir os 3 valores a Status, neste caso são EmProcessamento, Concluído e Cancelado) e uma lista de Produtos. Ao implementar a interface ICarriavel, Pedido é capaz de adicionar o produto caso ele não tenha o valor nulo, e remover produtos da lista caso existam, além de calcular o total dos itens adquiridos, adicionando o valor de cada um à variável “total”.

O método FinalizarPedido() atualiza o status do pedido para “Concluído” e pode realizar outras operações, como a atualização do estoque dos produtos físicos. Essa classe também utiliza encapsulamento para proteger a lista de produtos contra alterações indevidas, garantindo que o pedido seja tratado de forma segura e eficiente.

A classe Loja em C# é responsável por gerenciar a estrutura básica de uma loja, que envolve a manipulação de produtos, clientes e pedidos. Ela contém três propriedades principais, que são listas de objetos: Produtos, Clientes e, opcionalmente, Pedidos. A lista de produtos armazena todos os itens disponíveis na loja, a de clientes guarda as informações dos consumidores cadastrados, e a de pedidos registra os pedidos realizados.

Para o gerenciamento dos produtos, o método CadastrarProduto é responsável por adicionar novos produtos à lista, onde foi utilizado o método “Any”, esse método é utilizado para verificar se uma coleção ou sequência

contém pelo menos um elemento que satisfaz uma condição específica. Ele retorna um valor booleano (true ou false).

O método ConsultarProdutoPorCodigo permite buscar um produto na lista usando o método FirstOrDefault, onde ele percorre os produtos até encontrar um que esteja de acordo com o solicitado, neste caso é o código já existir.. Já o método ListarProdutos exibe todos os produtos disponíveis na loja, mostrando suas informações de forma organizada.

No gerenciamento de clientes, o método CadastrarCliente cadastra um novo cliente, garantindo que não haja duplicidade e que os dados sejam válidos. O método ConsultarClientePorID busca um cliente pelo seu número de identificação, retornando-o se for encontrado. O método ListarClientes exibe uma lista com todos os clientes registrados na loja.

Quanto ao gerenciamento de pedidos, o método CriarPedido inicia um novo pedido para um cliente existente, usando o método “Contains”, onde ele verifica se existe o cliente na lista de Clientes, associando-o ao cliente e adicionando-o à lista de pedidos. O método FinalizarPedido conclui um pedido, atualizando seu status e realizando ajustes, como a redução do estoque dos produtos comprados. O método ListarPedidos, embora opcional, pode ser implementado para mostrar todos os pedidos já feitos na loja.

O encapsulamento e a segurança dos dados são garantidos protegendo as listas internas, de modo que elas não possam ser modificadas diretamente. A manipulação dos dados ocorre apenas por meio de métodos específicos que controlam o acesso e a alteração de produtos, clientes e pedidos, mantendo a integridade das informações da loja.

O sistema apresentado exemplifica a aplicação dos principais conceitos de programação orientada a objetos, como abstração, herança, polimorfismo e encapsulamento. Através da definição de classes abstratas e da implementação de interfaces, o sistema garante flexibilidade e escalabilidade, permitindo que novos tipos de produtos ou funcionalidades sejam adicionados sem a necessidade de grandes modificações no código existente. A estrutura modular promove a reutilização de código e facilita a manutenção, ao mesmo tempo que protege os dados sensíveis por meio de técnicas de encapsulamento. Assim,

este projeto representa uma solução robusta e eficiente para o gerenciamento de produtos, clientes e pedidos em uma loja, ilustrando as boas práticas de desenvolvimento em C#.

REFERÊNCIAS

1 MICROSOFT. **String.IsNullOrWhiteSpace(String) Método**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/api/system.linq.enumerable.firstordefault?view=net-8.0. Acesso em: 19 out. 2024.

2 MICROSOFT. **Guid.NewGuid Método**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/api/system.guid.newguid?view=net-8.0. Acesso em: 20 out. 2024.

3 MICROSOFT. **Enumerable.Any Método**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/api/system.linq.enumerable.any?view=net-8.0. Acesso em: 21 out. 2024.

4 MICROSOFT. **Enumerable.FirstOrDefault Method**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/api/system.linq.enumerable.firstordefault?view=net-8.0. Acesso em: 21 out. 2024.

5 MICROSOFT. **String.Contains Método**. Disponível em: https://learn.microsoft.com/pt-br/dotnet/api/system.string.contains?view=net-8.0. Acesso em: 22 out. 2024.