Relatório 1ºTrabalho Prático

Programação Orientada a Objetos

Comércio eletrónico

Licenciatura em Engenharia Sistemas Informáticos

2ºano

27975 – Diogo Abreu

Barcelos | dezembro, 2024

Índice

Introdução

Este projeto, desenvolvido no contexto da disciplina de Programação Orientada a Objetos (POO), visa a criação de um sistema de comércio eletrônico para a gestão de uma loja online. O sistema tem como propósito simplificar a gestão de produtos, categorias, stocks, clientes, marcas, e campanhas promocionais, proporcionando uma interface eficiente para as operações de vendas.

Desenvolvido em C#, o projeto segue princípios de design orientado a objetos, como encapsulamento e modularidade, para assegurar uma estrutura escalável e de fácil manutenção.

A primeira fase do projeto concentra-se na modelagem e implementação básica das classes e estruturas de dados essenciais, definindo as bases para uma futura implementação completa das funcionalidades.

Fase 1

1_Estrutura de Classes Identificadas:

As classes principais incluem:

- Produto

A classe Produto utiliza todas as características de um item que se encontra à venda e oferece métodos para gestão de stock e exibição de detalhes.

- Pedido

A classe Pedido armazena informações detalhadas sobre a compra, essencial para o controlo de pedidos e ver o progresso de cada pedido. Essa estrutura permite uma experiência de compra organizada e bem documentada.

- Utilizador

A classe Utilizador serve como uma classe base para padronizar os atributos e comportamentos comuns entre os diferentes tipos de utilizadores. Isso promove o reuso de código e a organização.

- Admin

A classe Admin, que herda de Utilizador, permite uma gestão segura e estruturada, oferecendo uma visão global e controle completo sobre o sistema.

- Cliente

A classe Cliente fornece os dados essenciais para a interação com o cliente, garantindo uma experiência personalizada e uma gestão eficiente dos pedidos e do suporte ao cliente.

- Categoria

A Categoria é uma classe simples, mas essencial para organizar os produtos. As categorias facilitam a filtragem e pesquisa de produtos, além de auxiliar a gerir o inventário e na experiência do cliente.

2_Implementação Essencial das Classes:

a. Classe Cliente

Fundamento: Representa um comprador do sistema, armazenando suas informações pessoais e de contato.

Atributos:

- **idCliente**: Identificador do cliente.
- **nome**: Nome do cliente.
- **email**: Email do cliente.
- **nTelemovel**: Número de telefone do cliente.
- **nContribuinte**: Número de contribuinte (NIF).

Métodos:

- **AtualizarContato()**: Atualiza os dados de contato (email e telefone).
- **ExibirDados():** Exibe os dados completos do cliente para visualização por um administrador ou para consulta interna.
- **MostrarInformacoes**: Exibe as informações do cliente, incluindo o endereço, útil para quando um administrador deseja visualizar detalhes.

b. Classe Categoria

Fundamento: Agrupa produtos similares, como "Computadores" ou "Smartphones", ajudando os clientes a navegar facilmente pelo catálogo.

Atributos:

- **idCategoria**: Identificador único da categoria.
- **nomeCategoria**: Nome da categoria.

Métodos:

- **AtualizarNome**: Atualiza o nome da categoria com base em um novo valor fornecido como argumento.
- **ExibirInfo**: Retorna uma string com as informações da categoria, facilitando sua exibição em outros contextos.

c. Classe Cliente

Fundamento: Representa um cliente do sistema, armazenando suas informações pessoais e de contato.

Atributos:

- **idCliente**: Identificador único do cliente.
- **nome**: Nome completo do cliente.
- **email**: Endereço de email do cliente, para contato e comunicações de pedidos.
- **nTelemovel**: Número de telefone, para contato rápido.
- **nContribuinte**: Número fiscal, para fins de faturação e identificação legal.
- **endereco**: Endereço de entrega.

Métodos:

- **AtualizarContato**: Permite atualizar o email e o número de telefone do cliente.
- **ExibirDados**: Exibe os dados completos do cliente para visualização por um administrador ou para consulta interna.
- **MostrarInformacoes**: Exibe as informações do cliente, incluindo o endereço, útil para quando um administrador deseja visualizar detalhes.

d. Classe Produto

Fundamento: Representa um item que está disponível para venda na loja online.

Atributos:

- **id**: Identificador único para cada produto.
- **nome**: Nome do produto.
- **descricao**: Descrição detalhada, para que o cliente entenda suas características e funcionalidades.
- **preco**: Preço unitário do produto.
- **categoria**: Categoria à qual o produto pertence
- **marca**: Marca do produto, para facilitar a escolha de produtos de fabricantes específicos.
- **quantidadeEmStock**: Quantidade disponível em estoque, essencial para controle de inventário.
- **garantiaMeses**: Duração da garantia em meses, para informar o cliente sobre o suporte que receberá após a compra.

Métodos:

- **AtualizarStock**: Permite alterar a quantidade de produtos em estoque, uma operação comum no gerenciamento de inventário.
- **ExibirInformacoes**: Exibe as principais informações do produto para visualização pelo administrador ou cliente.

e. Classe Pedido

Fundamento: Regista informações sobre uma compra específica feita por um cliente, incluindo os produtos adquiridos, detalhes de pagamento e estado do pedido.

Atributos:

- **idPedido**: Identificador único do pedido.
- **produtoPedido**: Nome ou lista dos produtos incluídos no pedido.
- **cliente**: Nome do cliente que fez o pedido.
- **estadoDaEncomenda**: Estado atual do pedido (ex.: "Pendente", "Enviado", "Concluído").
- **dataEncomenda**: Data em que o pedido foi realizado, para referência de tempo e controlo.
- **valorEncomenda**: Valor total do pedido antes de impostos.
- **moradaAEnviar**: Endereço para o qual o pedido deve ser enviado.

Métodos:

- **AtualizarEstadoEncomenda**: Permite atualizar o estado do pedido conforme ele avança pelo processo de atendimento.
- **CalcularValorComImposto**: Calcula o valor total do pedido, adicionando o imposto aplicável, uma operação necessária para a transparência de preços e para o cliente saber o valor total.

f. Classe Admin

Fundamento: Representa um administrador do sistema, com permissões elevadas para gerir o sistema de comércio eletrônico, incluindo a supervisão de produtos, pedidos e utilizadores.

Atributos:

- **nivelAcesso**: Define o nível de acesso do administrador, diferenciando as permissões e responsabilidades.

Métodos:

- **MostrarInformacoes**: Exibe as informações do administrador, incluindo o nível de acesso, para facilitar a distinção de privilégios.

g. Classe Categorias

Funcionamento: Gere uma coleção de objetos da classe Categoria. A classe fornece métodos para adicionar e listar e categorias por ID.

Atributos:

- **listaCategorias**: Lista que armazena objetos do tipo Categoria.

Construtor:

- Inicializa a lista listaCategorias vazia.

Métodos:

- **AdicionarCategoria**: Adiciona um objeto Categoria à lista.
- **ListarCategorias**: Exibe todas as categorias e suas informações.

h. Classe Clientes

Funcionamento: Gere uma lista de objetos da classe Cliente. Essa classe fornece métodos para adicionar e listar clientes por ID.

Atributos:

- **listaClientes**: Lista que armazena objetos do tipo Cliente.

Construtor:

- Inicializa a lista listaClientes vazia.

Métodos:

- **AdicionarCliente**: Adiciona um objeto Cliente à lista.
- **ListarClientes**: Exibe todos os clientes e suas informações.

i. Classe Pedidos

Funcionamento: Gere uma coleção de objetos da classe Pedido. Essa classe fornece métodos para adicionar e lista pedidos por ID.

Atributos:

- **listaPedidos**: Lista que armazena objetos do tipo Pedido.

Construtor:

- Inicializa a lista listaPedidos vazia.

Métodos:

- **AdicionarPedido**: Adiciona um objeto Pedido à lista.
- **ListarPedidos**: Exibe todos os pedidos com suas informações básicas.

j.Classe Utilizadores

Funcionamento**:** Gere uma lista de objetos da classe Utilizador. Esta classe fornece métodos para adicionar, remover e listar utilizadores, bem como buscar um utilizador específico por ID.

Lista de Utilizadores:

- Armazena os objetos Utilizador em uma lista (List<Utilizador>).

Construtor:

- Inicializa a lista vazia no momento da criação de um objeto Utilizadores.

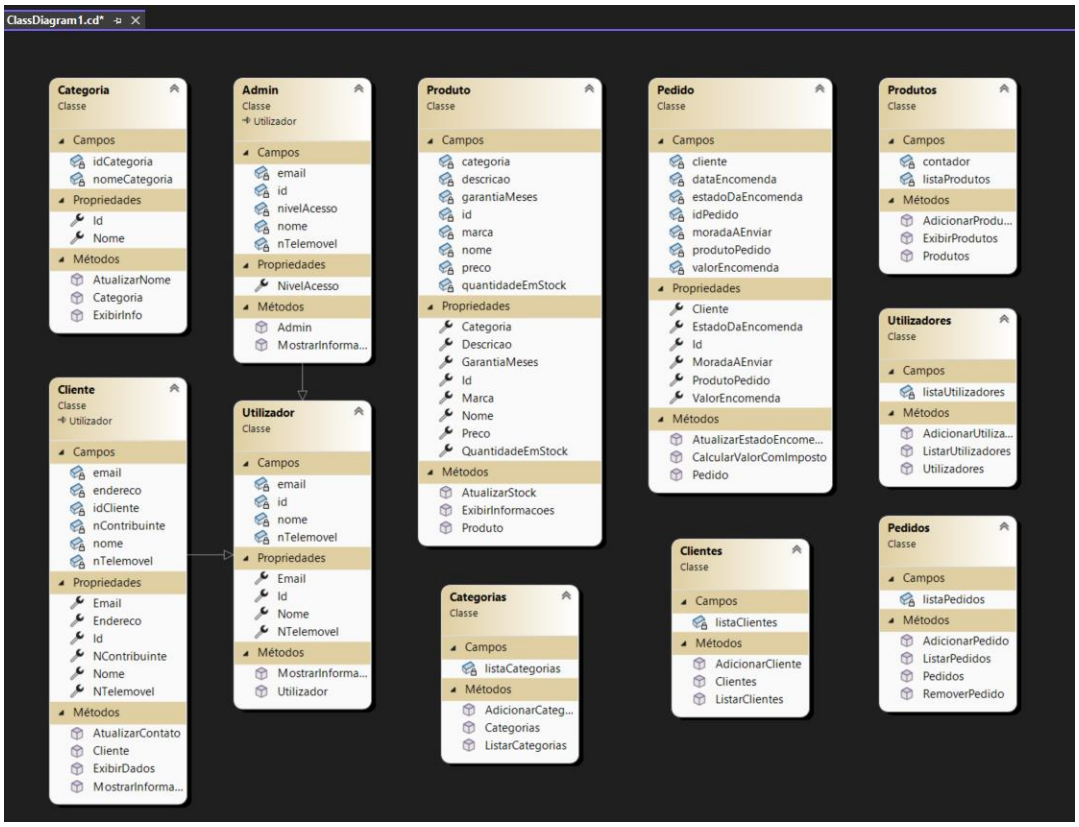
Métodos:

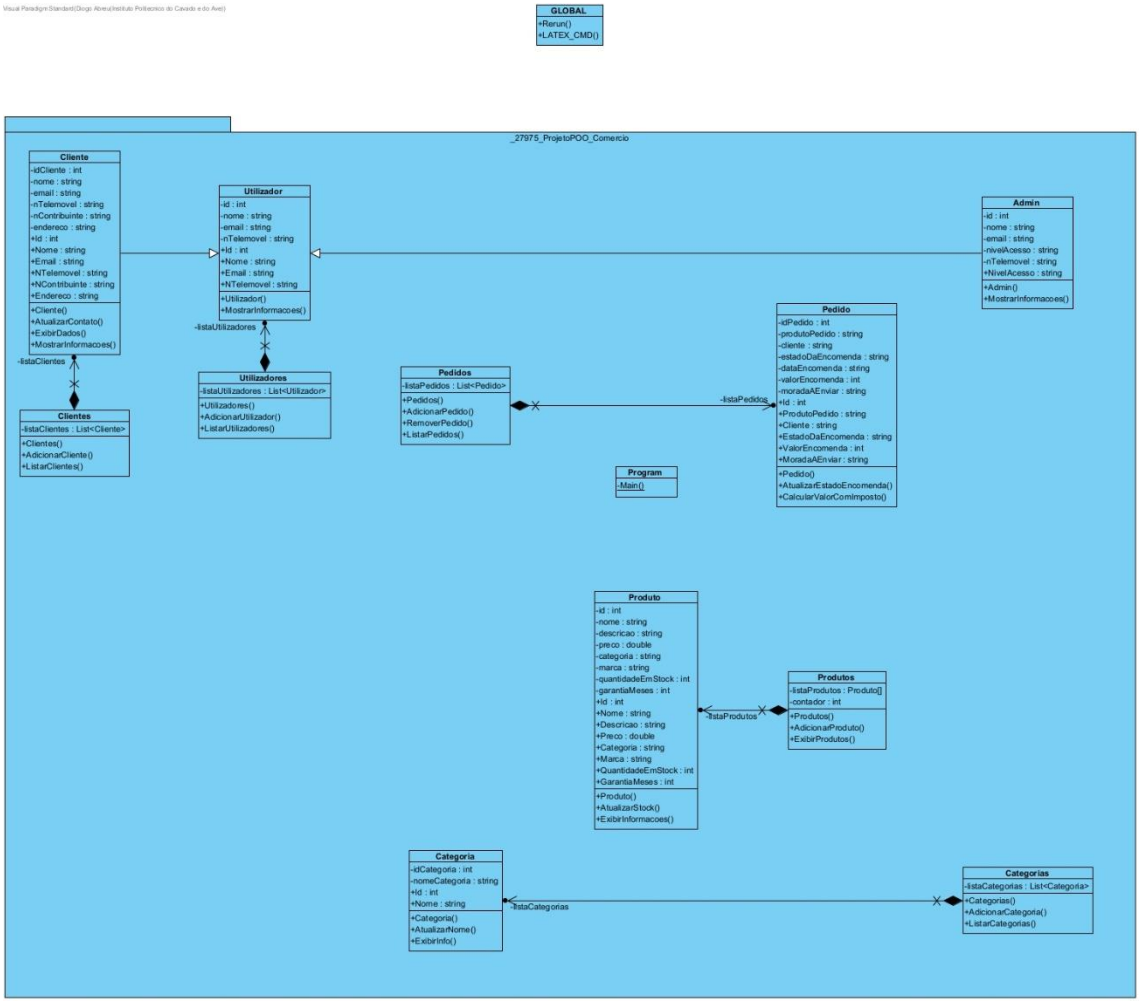
- **AdicionarUtilizador**: Adiciona um novo objeto Utilizador à lista.
- **ListarUtilizadores**: Exibe todas as informações dos utilizadores armazenados na lista.

k. Classe Produtos

Funcionamento: A classe Produtos armazena uma coleção de objetos Produto em um array.

- **Classe Produtos**: Contém um array listaProdutos de objetos Produto e um contador para gerenciar o índice atual.
- **Método AdicionarProduto(Produto produto)**: Adiciona um produto ao array, verificando se ainda há espaço.
- **Método ExibirProdutos()**: Percorre o array até o índice contador e exibe as informações de cada produto.


ClassDiagram1.cd*
Categoria
Classe
Campos
idCategoria
nomeCategoria
Propriedades
Id
Nome
Métodos
AtualizarNome
Categoria
ExibirInfo
Admin
Classe
Utilizador
Campos
email
id
nivelAcesso
nome
nTelemovel
Propriedades
NivelAcesso
Métodos
Admin
MostrarInforma...
Produto
Classe
Campos
categoria
descricao
garantiaMeses
id
marca
nome
preco
quantidadeEmStock
Propriedades
Categoria
Descricao
GarantiaMeses
Id
Marca
Nome
Preco
QuantidadeEmStock
Métodos
AtualizarStock
ExibirInformacoes
Produto
Pedido
Classe
Campos
cliente
dataEncomenda
estadoDaEncomenda
idPedido
moradaAEnviar
produtoPedido
valorEncomenda
Propriedades
Cliente
EstadoDaEncomenda
Id
MoradaAEnviar
ProdutoPedido
ValorEncomenda
Métodos
AtualizarEstadoEncome...
CalcularValorComImposto
Pedido
Produtos
Classe
Campos
contador
listaProdutos
Métodos
AdicionarProdu...
ExibirProdutos
Produtos
Utilizadores
Classe
Campos
listaUtilizadores
Métodos
AdicionarUtiliza...
ListarUtilizadores
Utilizadores
Cliente
Classe
Utilizador
Campos
email
endereco
idCliente
nContribuinte
nome
nTelemovel
Propriedades
Email
Endereco
Id
NContribuinte
Nome
NTelemovel
Métodos
AtualizarContato
Cliente
ExibirDados
MostrarInforma...
Utilizador
Classe
Campos
email
id
nome
nTelemovel
Propriedades
Email
Id
Nome
NTelemovel
Métodos
MostrarInforma...
Utilizador
Categorias
Classe
Campos
listaCategorias
Métodos
AdicionarCateg...
Categorias
ListarCategorias
Clientes
Classe
Campos
listaClientes
Métodos
AdicionarCliente
Clientes
ListarClientes
Pedidos
Classe
Campos
listaPedidos
Métodos
AdicionarPedido
ListarPedidos
Pedidos
RemoverPedido



3_Estruturas de Dados a Utilizar:

As classes fornecidas não parecem utilizar explicitamente estruturas de dados complexas, como listas ou dicionários, dentro do código fornecido. No entanto, as classes utilizam:

- Tipos básicos (primitivos): int e string são usados para armazenar informações como IDs, nome, email e número de telefone.
- Propriedades (get e set): São usados para aceder e modificar os valores dos atributos, oferecendo encapsulamento.
- Para fases futuras do projeto, o uso de coleções, como ArrayList para armazenamento de produtos, clientes e pedidos, facilitará a gestão dinâmica de dados.

Fase 2

1_Mudança para Listas

A mudança do uso de ArrayList para List<T> reflete o avanço na linguagem C# em termos de desempenho, segurança e facilidade de uso. O List<T> é mais eficiente e adequado para a maioria das situações, sendo o padrão atual para coleções na linguagem.

O uso de ArrayList é considerado obsoleto e deve ser evitado em novos projetos.

A ListasLib, fornece exemplos práticos de como instanciar e manipular objetos das classes disponíveis.

No programa, são criados objetos das seguintes classes:

- **Clientes**: Representa um cliente com atributos como nome, e-mail e idade.
- **Produtos**: Define informações sobre um produto, incluindo nome, preço e quantidade em stock.
- **Pedidos**: Associa um cliente a um produto, registando a quantidade adquirida.
- **Categorias**: Organiza produtos em categorias, com nome e descrição.
- **Utilizadores**: Representa utilizadores do sistema, com nome, e-mail e permissões.

2_MyCompare

Esta classe é utilizada para realizar comparações personalizadas entre objetos de diferentes tipos (Cliente, Pedido, Categoria), com base em critérios específicos.

As principais partes do código e as razões por trás destas escolhas:

- Objetivo da Classe: Facilitar a ordenação personalizada com base em critérios de negócio.
- Razão da Escolha do IComparer: Garante que a classe seja compatível com coleções padrões do .NET, como listas.
- Extensibilidade: A estrutura pode ser facilmente expandida para suportar novos tipos de objetos ou critérios de comparação.
- Robustez: Verificação de null e tratamento adequado de tipos desconhecidos previnem erros em tempo de execução.

3_Exceções

As exceções são um recurso essencial para garantir que um programa possa lidar com falhas de maneira robusta e, quando possível, fornecer informações úteis para corrigir ou evitar tais falhas no futuro.

Essas exceções foram projetadas para tratar casos específicos de erros em um sistema, como a falta de um cliente, produtos em pedidos ou problemas com o stock e preços.

ClienteNaoEncontrado.cs:

- Define a exceção ClienteNaoEncontrado, derivada de ApplicationException.
- Tem três construtores: um sem parâmetros, um que recebe uma mensagem e outro que aceita uma mensagem e uma exceção interna.
- Mensagem padrão: "Cliente não encontrado."

NaoPodeInserir.cs:

- Define a exceção NaoPodeInserir, também derivada de ApplicationException.
- Tem três construtores: um sem parâmetros com uma mensagem padrão "impossível", outro que recebe uma mensagem personalizada e um que recebe uma exceção interna.
- O construtor personalizado lança uma nova exceção com uma mensagem modificada: "Nem penses".

NaoPodeInserirUtilizador.cs:

- Define a exceção NaoPodeInserirUtilizador, derivada de ApplicationException.
- Tem um construtor padrão com uma mensagem: "Não foi possível inserir o item devido a uma regra de negócio".
- Também tem um construtor que permite passar uma mensagem personalizada e uma exceção interna.

PedidosSemProduto.cs:

- Define a exceção PedidosSemProduto, derivada de ApplicationException.
- Tem três construtores: um sem parâmetros, um com uma mensagem personalizada e outro que também aceita uma exceção interna.

- Mensagem padrão: "O pedido não contém produtos."

PrecoInvalido.cs:

- Define a exceção PrecoInvalido, derivada de ApplicationException.
- Tem três construtores: um sem parâmetros, um com uma mensagem personalizada e outro com uma exceção interna.
- Mensagem padrão: "O preço do produto é inválido (deve ser maior que zero)."

StockInsuficiente.cs:

- Define a exceção StockInsuficiente, derivada de ApplicationException.
- Tem três construtores: um sem parâmetros, um com uma mensagem personalizada e um com uma exceção interna.
- Mensagem padrão: "Stock insuficiente para o produto."

4_Regras

A biblioteca RegrasLib é uma coleção de classes em C# destinada para gerir as regras e entidades comuns em um sistema de negócios. Ela fornece funcionalidades básicas para a manipulação de clientes, pedidos, produtos e utilizadores, servindo como base para a lógica de negócios de aplicações empresariais.

Estrutura da Biblioteca

A RegrasLib está organizada em várias classes que representam entidades de domínio e encapsulam a lógica de negócio associada a elas. Algumas das principais classes incluídas são:

- **RegraCliente**: Representa clientes e fornece propriedades como Nome e Email para armazenar informações básicas sobre os clientes.
- **RegraPedido**: Gere pedidos, com propriedades como Id e Descricao, permitindo o ver o caminho de operações comerciais.
- **RegraProduto**: Modela produtos, armazenando detalhes como Nome e Preco.
- **RegraUtilizador**: Facilita a gestão de utilizadores com propriedades como email e Senha.

Exemplos de Uso

Para demonstrar as funcionalidades da biblioteca, foi implementado um programa simples em consola (RegrasLibDemo) que ilustra a criação e manipulação das entidades fornecidas.

5_Testes Unitários

Os testes unitários são uma etapa essencial para garantir a confiabilidade e o comportamento esperado dos métodos de uma classe. No caso da classe Pedido, criamos testes para validar a funcionalidade de dois métodos principais: AtualizarEstadoEncomenda e CalcularValorComImposto. Abaixo, descrevemos o propósito de cada teste, os cenários abordados e a implementação do código.

Objetivo dos Testes Unitários

- Garantir que os métodos da classe Pedido funcionem corretamente em diferentes cenários.
- Validar o comportamento esperado quando valores válidos e inválidos são usados como entrada.
- Aumentar a robustez e a confiabilidade do sistema, identificando possíveis falhas durante o desenvolvimento.

Ferramentas Utilizadas

Os testes foram implementados usando o Microsoft Visual Studio Test Framework, utilizando os atributos [TestClass] e [TestMethod] para definir classes de teste e métodos de teste, respetivamente.

Métodos Testados

1. AtualizarEstadoEncomenda

Esse método é responsável por atualizar o estado de um pedido. Ele recebe como parâmetro uma string representando o novo estado e atualiza o atributo estadoDaEncomenda.

Cenários Testados:

- Atualização bem-sucedida: Quando um estado válido é fornecido.
- Estado vazio: Verifica se o método mantém o estado anterior caso o novo estado seja inválido.

2.CalcularValorComImposto

Esse método calcula o valor total de um pedido, adicionando uma taxa de imposto ao valor base. Ele recebe a taxa de imposto (em formato decimal) como parâmetro.

Cenários Testados:

- Imposto de 20%: Verifica se o cálculo do valor com imposto está correto.
- Taxa de imposto zero: Garante que o valor original seja retornado.
- Valores limítrofes: Pode ser estendido para testar comportamentos com taxas negativas ou valores extremos.

Resultados dos Testes

Os testes cobriram os principais cenários de uso dos métodos. Todos os casos passaram com sucesso, garantindo que:

- O método AtualizarEstadoEncomenda altera corretamente o estado de um pedido, exceto quando o novo estado é inválido.
- O método CalcularValorComImposto realiza o cálculo correto para diferentes taxas de imposto, incluindo o caso onde a taxa é zero.

Síntese:

Os testes unitários demonstraram que a classe Pedido está a funcionar corretamente para os cenários abordados. Eles também servem como base para futuras alterações, permitindo que o comportamento esperado dos métodos seja validado de forma contínua.

Sugestões para melhorias futuras:

- Adicionar validação para impedir atualizações com estados inválidos.
- Estender os testes para cobrir cenários com valores fora do esperado, como taxas de imposto negativas.

Conclusão

Nesta fase inicial, foi possível identificar e implementar as principais classes e estruturas de dados que sustentam o sistema de comércio eletrônico. Através da aplicação dos conceitos de POO, foi criada uma base sólida e flexível, capaz de suportar operações essenciais como cadastro de produtos, gestão de stock, e controlo de pedidos e clientes. Com esta base implementada, o sistema está apto a evoluir para fases posteriores, que incluirão funcionalidades avançadas e maior interação com o utilizador.

Esta experiência proporcionou uma compreensão aprofundada do design orientado a objetos em C# e estabeleceu uma base promissora para o desenvolvimento contínuo do projeto.