UNIVERSIDADE LUTERANA DO BRASIL

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Programação Orientada a Objetos Aplicada a um Sistema de Gestão de Pedidos

Guilherme Trevisan Bobsin

CIDADE

Maquiné

ANO

2024

1. Introdução

A Programação Orientada a Objetos (POO) é um paradigma de programação amplamente utilizado na criação de software moderno. Sua importância reside na capacidade de organizar o código de maneira mais intuitiva e eficiente, permitindo maior manutenção, escalabilidade e reutilização de código. Este projeto foi desenvolvido utilizando os princípios da POO para demonstrar a aplicação prática dos conceitos de encapsulamento, herança, polimorfismo e abstração.

Este texto tem como objetivo explorar esses pilares, ilustrando como eles foram aplicados no projeto desenvolvido. Será feita uma análise detalhada, com exemplos de código que demonstram a aplicação prática de cada conceito.

2. Encapsulamento

O encapsulamento refere-se ao conceito de restringir o acesso a certos componentes de um objeto, expondo apenas o que é necessário. No contexto do projeto, o encapsulamento foi aplicado ao controlar o acesso aos atributos das classes por meio de métodos "get" e "set".

Exemplo de código:

```
public class ProdutoFisico : Produto
{
    private int _estoque;

    public int Estoque
    {
        get { return _estoque; }
        set
        {
            if (value < 0)
                throw new ArgumentException("O estoque não pode ser negativo.");
            _estoque = value;
```

```
        }
    }
}
```

Neste exemplo, o atributo _estoque é privado e seu acesso é controlado pelos métodos públicos. Isso garante que a manipulação do estoque ocorra de maneira controlada, evitando inconsistências.

3. Herança

Herança é a capacidade de uma classe "herdar" características e comportamentos de outra classe. Esse mecanismo permite a reutilização de código e a criação de hierarquias de classes.

No projeto, a classe ProdutoFisico herda de uma classe base Produto, aproveitando atributos e métodos comuns.

Exemplo de código:

```
public class Produto :
{
    public string Nome { get; set; }
    public decimal Preco { get; set; }

    public Produto(string nome, decimal preco)
    {
        Nome = nome;
        Preco = preco;
    }
}

public class ProdutoFisico : Produto
{
    public double Peso { get; set; }
```

```
    public Dimensoes Dimensoes { get; set; }

    public ProdutoFisico(string nome, decimal preco, double peso, Dimensoes
dimensoes)
        : base(nome, preco)
    {
        Peso = peso;
        Dimensoes = dimensoes;
    }
}
```

Aqui, a classe ProdutoFisico herda de Produto, reutilizando os atributos Nome e Preco, o que facilita a extensão de funcionalidades específicas para produtos físicos sem duplicar código.

4. Polimorfismo

O polimorfismo permite que objetos de diferentes classes sejam tratados de maneira uniforme quando pertencem a uma mesma hierarquia. Isso facilita a implementação de métodos que podem manipular diversos tipos de objetos de forma flexível.

No projeto, o polimorfismo foi utilizado para manipular diferentes tipos de produtos (físicos e digitais) de forma homogênea na classe Pedido.

Exemplo de código:

```
public class Pedido
{
    public List<Produto> Produtos { get; set; } = new List<Produto>();

    public void AdicionarProduto(Produto produto)
    {
        Produtos.Add(produto);
    }
```

```
    public decimal CalcularTotal()
    {
        decimal total = 0;
        foreach (var produto in Produtos)
        {
            total += produto.Preco;
        }
        return total;
    }
}
```

Neste exemplo, o método AdicionarProduto recebe objetos do tipo Produto, que podem ser tanto ProdutoFisico quanto ProdutoDigital, demonstrando o uso do polimorfismo.

5. Abstração

A abstração é o processo de modelar classes focando apenas nos detalhes essenciais, ocultando a complexidade subjacente. Ela é fundamental para criar um modelo simplificado do mundo real no código.

No projeto, a abstração foi aplicada na criação de classes que representam os produtos e os pedidos, permitindo trabalhar com conceitos genéricos e ocultando os detalhes de implementação.

Exemplo de código:

```
public abstract class Produto
{
    public string Nome { get; set; }
    public decimal Preco { get; set; }

    public Produto(string nome, decimal preco)
```

```
    {
        Nome = nome;
        Preco = preco;
    }

    public abstract void ExibirInformacoes();
}

public class ProdutoDigital : Produto
{
    public double Tamanho { get; set; }

    public ProdutoDigital(string nome, decimal preco, double tamanho)
        : base(nome, preco)
    {
        Tamanho = tamanho;
    }

    public override void ExibirInformacoes()
    {
        Console.WriteLine($"{Nome} - {Preco:C} - {Tamanho}MB");
    }
}
```

A classe Produto define uma estrutura abstrata, enquanto a classe ProdutoDigital implementa detalhes específicos, permitindo que novas classes sejam adicionadas sem modificar o comportamento geral do sistema.

6. Conclusão

O desenvolvimento deste projeto proporcionou uma compreensão prática dos pilares da Programação Orientada a Objetos. Ao aplicar conceitos como encapsulamento, herança, polimorfismo e abstração, foi possível criar um código mais organizado, modular e flexível. O aprendizado obtido reforça a importância

desses princípios para o desenvolvimento de software escalável e de fácil manutenção.

7. Referências Bibliográficas

- DEITEL, H. M.; DEITEL, P. J. *C# Como Programar*. 6ª ed. São Paulo: Pearson Prentice Hall, 2010.
- MARTIN, Robert C. *Código Limpo: Habilidades Práticas do Agile Software*. São Paulo: Alta Books, 2017.
- SOMMERVILLE, Ian. *Engenharia de Software*. 10ª ed. São Paulo: Pearson, 2011.