Sistema de Gestão de Inventário

Introdução ao Problema

Imagine que você está desenvolvendo um sistema para gerenciar o inventário de uma loja. A loja vende diferentes tipos de produtos, como eletrônicos, móveis e roupas. Cada produto tem características específicas, mas também há características comuns a todos os produtos. Inicialmente, você pode pensar em criar classes separadas para cada tipo de produto, com seus próprios atributos e métodos. No entanto, essa abordagem pode se tornar ineficiente e difícil de manter à medida que a loja cresce. Vamos explorar uma implementação inicial e identificar os problemas antes de propor uma solução ideal.

Implementação Inicial (Incorreta)

Aqui está uma implementação inicial onde cada tipo de produto é representado por uma classe separada:

```
public class Eletronico
{
    public string Nome { get; set; }
    public string Marca { get; set; }
    public decimal Preco { get; set; }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Eletrônico: {Nome}, Marca: {Marca}, Preço: {Preco}");
    }
}

public class Movel
{
    public string Nome { get; set; }
    public string Material { get; set; }
    public decimal Preco { get; set; }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Móvel: {Nome}, Material: {Material}, Preço: {Preco}");
    }
}

public class Roupa
{
    public string Nome { get; set; }
    public string Tamanho { get; set; }
    public decimal Preco { get; set; }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Roupa: {Nome}, Tamanho: {Tamanho}, Preço: {Preco}");
```

```
    }
}
```

Problemas com a Implementação Inicial

Vamos refletir sobre os problemas dessa abordagem:

1. **Duplicação de Código**: As classes Eletronico, Movel e Roupa possuem atributos comuns, como Nome e Preco. Isso é ineficiente porque estamos repetindo o mesmo código em várias classes.
   - **Pergunta**: O que acontece se precisarmos adicionar um novo atributo comum a todos os produtos, como CodigoDeBarras? Teríamos que modificar cada classe individualmente. Isso parece prático?
2. **Manutenção Difícil**: Se precisar adicionar ou alterar um atributo comum, terá que fazer isso em várias classes, aumentando a chance de erro.
   - **Pergunta**: Se houver um erro em um atributo comum a todos os produtos, como garantir que ele seja corrigido em todas as classes? Isso é eficiente para um sistema grande?

Solução Usando Composição

Para resolver esses problemas, podemos usar composição. Vamos criar uma classe base que contém os atributos comuns e usá-la em outras classes específicas de produtos.

Implementação Melhorada Usando Composição

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

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

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Produto: {Nome}, Preço: {Preco}");
    }
}

public class Eletronico
{
    public Produto Produto { get; set; }
    public string Marca { get; set; }

    public Eletronico(Produto produto, string marca)
    {
        Produto = produto;
        Marca = marca;
    }
```

```
    public void MostrarDetalhes()
    {
        Produto.MostrarDetalhes();
        Console.WriteLine($"Marca: {Marca}");
    }
}

public class Movel
{
    public Produto Produto { get; set; }
    public string Material { get; set; }

    public Movel(Produto produto, string material)
    {
        Produto = produto;
        Material = material;
    }

    public void MostrarDetalhes()
    {
        Produto.MostrarDetalhes();
        Console.WriteLine($"Material: {Material}");
    }
}

public class Roupa
{
    public Produto Produto { get; set; }
    public string Tamanho { get; set; }

    public Roupa(Produto produto, string tamanho)
    {
        Produto = produto;
        Tamanho = tamanho;
    }

    public void MostrarDetalhes()
    {
        Produto.MostrarDetalhes();
        Console.WriteLine($"Tamanho: {Tamanho}");
    }
}

// Exemplo de uso
public class Program
{
    public static void Main()
    {
        Produto eletronicoProduto = new Produto("Smartphone", 1500m);
        Eletronico eletronico = new Eletronico(eletronicoProduto, "Samsung");
```

```
        Produto movelProduto = new Produto("Sofá", 2000m);
        Movel movel = new Movel(movelProduto, "Couro");

        Produto roupaProduto = new Produto("Camiseta", 50m);
        Roupa roupa = new Roupa(roupaProduto, "M");

        eletronico.MostrarDetalhes();
        movel.MostrarDetalhes();
        roupa.MostrarDetalhes();
    }
}
```

Benefícios da Composição

Vamos refletir sobre os benefícios que essa abordagem traz:

1. **Reutilização de Código**: Reduzimos a duplicação de código comum entre diferentes tipos de produtos, facilitando a manutenção.
2. **Organização e Estrutura**: A composição permite uma melhor organização do código, refletindo a estrutura lógica do sistema de inventário.
3. **Flexibilidade**: A composição facilita a criação de comportamentos especializados para diferentes tipos de produtos, tornando o sistema mais flexível.

Desvantagens e Considerações

Mas nem tudo são flores. Precisamos estar atentos a alguns pontos:

1. **Gerenciamento de Objetos**: A composição pode aumentar a complexidade no gerenciamento de objetos, especialmente em sistemas grandes com muitos componentes.
2. **Desempenho**: Em alguns casos, a composição pode introduzir overhead adicional, afetando o desempenho do sistema.

Conclusão

A composição é uma técnica poderosa na programação orientada a objetos, permitindo criar sistemas flexíveis e modulares. Utilizando composição, podemos reduzir a duplicação de código, aumentar a reutilização e facilitar a manutenção do sistema. O que você acha, está pronto para aplicar esses conceitos?

Exercícios de Fixação

Exercícios Teóricos

1. Explique como a composição ajuda a reduzir a duplicação de código em sistemas empresariais.
2. Descreva um cenário em que o uso de composição pode ser desfavorável devido à complexidade no gerenciamento de objetos.

Exercícios Práticos

1. Implemente uma hierarquia de classes para um sistema de gerenciamento de bibliotecas, incluindo classes para `Livro`, `Autor` e `Categoria`, utilizando composição.

2. Crie uma aplicação que gerencie diferentes tipos de eventos em uma agenda, utilizando composição para especializar classes como `EventoPessoal`, `EventoProfissional` e `EventoSocial`.