Sistema de Gerenciamento de Produtos

Introdução ao Problema

Vamos imaginar um novo contexto. Suponha que você está desenvolvendo um sistema para gerenciar diferentes tipos de produtos em uma loja online. O objetivo é criar um sistema que possa gerenciar eletrônicos, roupas e alimentos. Inicialmente, pode parecer uma boa ideia criar classes separadas para cada tipo de produto, onde cada classe possui seus próprios atributos e métodos. No entanto, essa abordagem pode se tornar ineficiente e difícil de manter conforme 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 string Modelo { get; set; }
    public decimal Preco { get; set; }
    public int Garantia { get; set; }

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

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

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

public class Alimento
{
    public string Nome { get; set; }
    public DateTime DataDeValidade { get; set; }
```

```
    public decimal Preco { get; set; }
    public float Peso { get; set; }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Data de Validade: {DataDeValidade}, Preço:
{Preco}, Peso: {Peso} kg");
    }
}
```

Problemas com a Implementação Inicial

Vamos refletir sobre os problemas dessa abordagem:

1. **Duplicação de Código**: Notem que as classes `Eletronico`, `Roupa` e `Alimento` possuem um atributo comum, `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 você 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 garantiríamos que ele fosse corrigido em todas as classes? Isso é eficiente para um sistema grande?

Solução Usando Herança por Extensão e Implementação

Para resolver esses problemas, podemos usar a herança tanto por extensão quanto por implementação. Vamos criar uma classe base que contém os atributos comuns e derivar classes específicas dela, ao mesmo tempo que implementamos uma interface para definir métodos comuns.

Implementação Melhorada Usando Herança por Extensão e Implementação

```
// Interface para produtos
public interface IProduto
{
    string Nome { get; set; }
    decimal Preco { get; set; }
    string CodigoDeBarras { get; set; }
    void MostrarDetalhes();
}

// Classe base para produtos
public abstract class Produto : IProduto
{
    public string Nome { get; set; }
    public decimal Preco { get; set; }
    public string CodigoDeBarras { get; set; }
```

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

    public abstract void MostrarDetalhes();
}

// Classe derivada para eletrônicos
public class Eletronico : Produto
{
    public string Marca { get; set; }
    public string Modelo { get; set; }
    public int Garantia { get; set; }

    public Eletronico(string nome, decimal preco, string codigoDeBarras, string marca,
string modelo, int garantia)
        : base(nome, preco, codigoDeBarras)
    {
        Marca = marca;
        Modelo = modelo;
        Garantia = garantia;
    }

    public override void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Marca: {Marca}, Modelo: {Modelo}, Preço:
{Preco}, Garantia: {Garantia} meses, Código de Barras: {CodigoDeBarras}");
    }
}

// Classe derivada para roupas
public class Roupa : Produto
{
    public string Tamanho { get; set; }
    public string Cor { get; set; }
    public string Material { get; set; }

    public Roupa(string nome, decimal preco, string codigoDeBarras, string tamanho,
string cor, string material)
        : base(nome, preco, codigoDeBarras)
    {
        Tamanho = tamanho;
        Cor = cor;
        Material = material;
    }

    public override void MostrarDetalhes()
    {
```

```
        Console.WriteLine($"Nome: {Nome}, Tamanho: {Tamanho}, Cor: {Cor}, Preço:
{Preco}, Material: {Material}, Código de Barras: {CodigoDeBarras}");
    }
}

// Classe derivada para alimentos
public class Alimento : Produto
{
    public DateTime DataDeValidade { get; set; }
    public float Peso { get; set; }

    public Alimento(string nome, decimal preco, string codigoDeBarras, DateTime
dataDeValidade, float peso)
        : base(nome, preco, codigoDeBarras)
    {
        DataDeValidade = dataDeValidade;
        Peso = peso;
    }

    public override void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Data de Validade: {DataDeValidade}, Preço:
{Preco}, Peso: {Peso} kg, Código de Barras: {CodigoDeBarras}");
    }
}

// Exemplo de uso
public class Program
{
    public static void Main()
    {
        Eletronico eletronico = new Eletronico("Smartphone", 1200m, "123456789",
"Samsung", "Galaxy S21", 24);
        Roupa roupa = new Roupa("Camiseta", 50m, "987654321", "M", "Azul", "Algodão");
        Alimento alimento = new Alimento("Arroz", 20m, "192837465", new DateTime(2023,
12, 31), 5);

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

Benefícios da Herança por Extensão e Implementação

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 herança permite uma melhor organização do código, refletindo a estrutura lógica da loja.

3. **Especialização**: A herança facilita a criação de comportamentos especializados para diferentes tipos de produtos, tornando o sistema mais flexível.
4. **Contratos Claros**: As interfaces definem contratos claros que as classes derivadas devem seguir, garantindo consistência no comportamento dos produtos.

Desvantagens e Considerações

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

1. **Acoplamento**: A classe derivada depende fortemente da classe base, o que pode dificultar mudanças na classe base.
2. **Sobrescrita de Métodos**: Precisamos ser cuidadosos ao sobrescrever métodos para garantir que o comportamento desejado seja implementado corretamente.
3. **Complexidade**: A combinação de herança por extensão e implementação pode aumentar a complexidade do sistema, exigindo um bom entendimento dos conceitos de orientação a objetos.

Conclusão

A combinação de herança por extensão e implementação é uma ferramenta poderosa e prática para o desenvolvimento de sistemas empresariais complexos. Ela permite que diferentes tipos de objetos compartilhem características comuns, enquanto possuem comportamentos específicos. Utilizando-a corretamente, podemos obter um código mais limpo, organizado e fácil de manter. O que você acha, está pronto para aplicar esses conceitos?

Exercícios de Fixação

Exercícios Teóricos

```
1. Explique como a herança por extensão e implementação ajuda a reduzir a duplicação de código em sistemas empresariais.
2. Descreva um cenário em que a combinação de herança por extensão e implementação pode ser desfavorável devido à complexidade.
```

Exercícios Práticos

```
1. Implemente uma hierarquia de classes para um sistema de gerenciamento escolar, incluindo classes para Aluno, Professor e FuncionarioAdministrativo, utilizando herança por extensão e implementação.
2. Crie uma aplicação que gerencie diferentes tipos de veículos em uma locadora, utilizando herança por extensão e implementação para especializar classes como Carro, Moto e Caminhao.
```