

# Principais Práticas de Imutabilidade no C# para um Código mais Seguro e Eficiente

## Nagib Sabbag Filho

FIAP (Faculty of Informatics and Administration Paulista) Avenida Paulista, 1106 - 7º andar - Bela Vista, São Paulo, Brazil.

e-mail: profnagib.filho@fiap.com.br

PermaLink: <https://leaders.tec.br/article/principais-praticas-de-imutabilidade-no-c-para-um-codigo-mais-seguro-e-eficiente>  
out 21 2024

---

### Abstract:

Este artigo explora o conceito de imutabilidade em C#, destacando suas vantagens, como segurança e eficiência no desenvolvimento de software. Apresenta a implementação de classes e structs imutáveis, além de tuplas, que são imutáveis por padrão. O uso de coleções imutáveis também é discutido, enfatizando a integridade dos dados em aplicações complexas.

### Key words:

Imutabilidade, C#, código seguro, código eficiente, práticas de programação, tipos imutáveis, strings imutáveis, programação funcional, concorrência, thread-safe, design de software, padrões de projeto, encapsulamento, estado imutável, performance, manutenção de código, boas práticas, estruturas de dados imutáveis, segurança de dados.

---

## Entendendo a Imutabilidade em C#

Imutabilidade é um conceito fundamental em programação, especialmente em linguagens como C#. Quando uma variável é imutável, seu estado não pode ser alterado após a sua criação. Isso traz benefícios significativos em termos de segurança e eficiência, pois reduz o risco de efeitos colaterais indesejados e facilita o raciocínio sobre o comportamento do código. Com a crescente complexidade dos sistemas modernos, onde múltiplas partes do código podem interagir de maneiras inesperadas, a imutabilidade se torna uma ferramenta vital para garantir a integridade dos dados e a previsibilidade do comportamento das aplicações.

## Vantagens da Imutabilidade

A imutabilidade oferece várias vantagens. Primeiro, ela melhora a segurança do código. Como os dados não podem ser alterados, é mais fácil garantir que as funções não modificarão o estado de objetos inesperadamente. Além disso, a imutabilidade pode facilitar o desenvolvimento em ambientes concorrentes, onde múltiplas threads podem acessar os mesmos dados. Isso reduz a necessidade de mecanismos complexos de sincronização, que podem introduzir bugs e complexidade adicional. Outro benefício importante é a maior facilidade em testar o código, já que a previsibilidade que a imutabilidade proporciona torna os testes mais simples e eficazes.

## Classes Imutáveis em C#

Uma das formas mais comuns de implementar a imutabilidade em C# é através de classes imutáveis. Para criar uma classe imutável, você deve definir suas propriedades como somente leitura e inicializá-las através de um construtor. Abaixo está um exemplo de uma classe imutável:

```
public class Ponto
{
    public int X { get; }
    public int Y { get; }
```

```
public Ponto(int x, int y)
{
    X = x;
    Y = y;
}

public Ponto Mover(int deltaX, int deltaY)
{
    return new Ponto(X + deltaX, Y + deltaY);
}
}
```

Neste exemplo, a classe Ponto é imutável. O método Mover retorna um novo objeto Ponto com as novas coordenadas, em vez de modificar o objeto existente. Isso não só garante que o objeto original permaneça inalterado, mas também permite que os desenvolvedores rastreiem facilmente as mudanças de estado ao longo do tempo, uma vez que cada mudança resulta em um novo objeto.

## Structs Imutáveis

Em C#, structs são tipos de valor que também podem ser usados de forma imutável. Para criar um struct imutável, você pode seguir um padrão semelhante ao das classes:

```
public struct Cor
{
    public byte Vermelho { get; }
    public byte Verde { get; }
    public byte Azul { get; }

    public Cor(byte vermelho, byte verde, byte azul)
    {
        Vermelho = vermelho;
        Verde = verde;
        Azul = azul;
    }

    public Cor Misturar(Cor outra)
    {
        return new Cor(
            (byte)((Vermelho + outra.Vermelho) / 2),
            (byte)((Verde + outra.Verde) / 2),
            (byte)((Azul + outra.Azul) / 2));
    }
}
```

A struct Cor é um exemplo de um tipo de valor imutável que pode ser usado para representar cores RGB. Assim como nas classes, a imutabilidade nas structs ajuda a evitar modificações acidentais, além de permitir que cada operação que resulta em uma nova cor não afete a cor original.

## Utilizando Tuplas Imutáveis

As tuplas são introduzidas no C# 7 e são imutáveis por padrão. Elas oferecem uma maneira rápida e fácil de agrupar dados sem a necessidade de criar uma classe ou struct. Veja um exemplo:

```
var ponto = (X: 10, Y: 20);  
var novoPonto = (X: ponto.X + 5, Y: ponto.Y + 5);
```

Aqui, ponto é uma tupla imutável. O novo ponto é criado sem modificar o original. Isso torna as tuplas uma escolha popular para retornar múltiplos valores de métodos sem a complexidade de definir uma nova classe ou struct.

## Padrões Funcionais e Imutabilidade

A imutabilidade é um conceito central em programação funcional. Em C#, podemos adotar padrões funcionais utilizando delegates e expressões lambda. Veja um exemplo usando LINQ:

```
var numeros = new List { 1, 2, 3, 4, 5 };  
var numerosQuadrados = numeros.Select(n => n * n).ToList();
```

Neste exemplo, a operação de mapear os números para seus quadrados é realizada sem modificar a lista original, promovendo a imutabilidade. Esse estilo de programação não apenas torna o código mais legível, mas também facilita a manutenção e a compreensão de como os dados fluem através do sistema.

## Uso de Immutable Collections

Desde o C# 7.0, a biblioteca System.Collections.Immutable permite o uso de coleções imutáveis. Essas coleções são úteis quando se deseja garantir que uma lista ou dicionário não seja alterado. Um exemplo de uso é:

```
using System.Collections.Immutable;  
  
var listaImutavel = ImmutableList.Create(1, 2, 3);  
var novaLista = listaImutavel.Add(4);
```

A lista original listaImutavel permanece inalterada, enquanto novaLista contém o novo elemento. Isso é especialmente útil em aplicações onde a integridade dos dados é crítica, pois evita modificações inesperadas em coleções que podem ser acessadas de várias partes do código.

## Práticas Recomendadas para Implementação de Imutabilidade

Adotar a imutabilidade não é apenas sobre usar classes e structs imutáveis. Existem práticas recomendadas que podem ajudar a implementar a imutabilidade de maneira eficaz:

Utilize o modificador readonly para campos que não devem ser alterados após a inicialização. Isso ajuda a reforçar a intenção de que esses campos são imutáveis.

Evite métodos que alterem o estado do objeto; em vez disso, crie métodos que retornem novos objetos. Essa abordagem permite que você mantenha a integridade dos objetos existentes.

Considere o uso de coleções imutáveis para evitar modificações indesejadas. Coleções imutáveis garantem que os dados não sejam alterados inadvertidamente em partes do código que não deveriam ter acesso a essas alterações.

Documente claramente as intenções de imutabilidade em seu código. Isso ajuda outros desenvolvedores (ou você mesmo no futuro) a compreender rapidamente as razões por trás das decisões de design.

Utilize a imutabilidade em combinação com padrões de design como o Builder ou o Prototype, que podem ajudar a criar instâncias de objetos complexos de maneira controlada e previsível.

## Considerações Finais sobre Imutabilidade

A imutabilidade é uma prática poderosa que pode levar a um código mais seguro e eficiente. Em um mundo onde a concorrência é cada vez mais comum, implementar a imutabilidade pode ser um grande diferencial na qualidade do software. C# oferece várias ferramentas e técnicas para facilitar a adoção da imutabilidade, e é uma estratégia que todos os desenvolvedores devem considerar. Ao priorizar a imutabilidade, você não apenas melhora a segurança e a previsibilidade de seu código, mas também cria uma base sólida para o desenvolvimento ágil e sustentável.

## Referências

Microsoft. (2021). C# Programming Guide. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>

Microsoft. (2021). Immutable Collections. Disponível em: <https://docs.microsoft.com/en-us/dotnet/standard/collections/immutable/>

---

Nagib é Professor Universitário e Tech Manager. Possui uma trajetória de conquistas em certificações técnicas e ágeis, incluindo MCSD, MCSA e PSM1. PG em Gestão de TI pelo SENAC e MBA em Tecnologia de Software pela USP, Nagib cursou programas de extensão do MIT e Universidade de Chicago. Outras conquistas incluem a autoria de um artigo sobre chatbots, revisado por pares e apresentado na Universidade de Barcelona.