**Guia de Implementação**

Versão 1.0

SUMÁRIO

[1. Introdução 3](#_Toc17102)

[1.1. Referências 3](#_Toc13904)

[2. Organização e Estilo do Código C# 3](#_Toc15143)

[2.1. Comentários em C# 3](#_Toc3002)

[2.2. Nomeação 3](#_Toc17707)

[2.2.1. Nomeação de projetos 4](#_Toc32267)

[2.2.2. Nomeação de enumeradores 4](#_Toc12229)

[2.2.3. Nomeação de interfaces de serviço 4](#_Toc8635)

[2.2.4. Nomeação de interfaces de repositório 4](#_Toc8667)

[2.2.5. Nomeação de implementações de serviço 4](#_Toc7455)

[2.2.6. Nomeação de implementações de repositório 4](#_Toc32596)

[2.2.7. Nomeação classes de validação 4](#_Toc18403)

[2.2.8. Nomeação classes de conversores 4](#_Toc7969)

[3. Tratamento de Erros e de Exceções 5](#_Toc19290)

[4. teste unitário 5](#_Toc8899)

[5. Diretrizes Gerais 5](#_Toc29307)

# Introdução

Este documento têm como finalidade servir como referência para boas práticas de escrita de código, deve ser muito bem conhecido pelos desenvolvedores e servir como parte indispensável nas tarefas de revisão de código.

## Referências

* [Convenções de codificação em C#](https://docs.microsoft.com/pt-br/dotnet/csharp/fundamentals/coding-style/coding-conventions)
* [Angular coding style guide](https://angular.io/guide/styleguide" \l "angular-coding-style-guide" \o "Link to this heading)
* [Effective Dart: Style](https://dart.dev/guides/language/effective-dart/style)

# Organização e Estilo do Código C#

* Todo código deve ser indentado com 4 espaços ou tabulação;
* Todo item público deve ser documentado utilizando a notação de comentários em XML do C# (ver seção 2.1);
* Deve haver uma linha em branco entre as definições de método e de propriedades;
* Uma linha não deve conter mais de uma instrução ou declaração;
* Caso seja necessário encadear chamadas de métodos, faça isto quebrando a linha;

## Comentários em C#

Para adicionar estes comentários pelo Visual Studio, basta posicioinar cursor na linha diretamente acima da classe, propriedade ou método e pressionar a tecla “/” três vezes. Assim, uma estrutura XML será gerada, bastando preenchê-la com as informações de documentação.

Esta documentação será útil enquanto se desenvolve o código, no Visual Studio enquanto pairamos o mouse sobre um determinado tipo ele exibe as informações a partir desta documentação

**Exemplo**

/// <summary>

/// Classe base para implementações de repositórios.

/// </summary>

## Nomeação

Utilize **PascalCase** para nomear:

* + Classes;
  + Interfaces;
  + Enums;
  + Structs;
  + Em membros de tipo públicos (propriedades, eventos, métodos, funções);
  + Nomes de projetos;

Utilize **camelCase** para nomear:

* + Membros privados (estes devem possuir ainda o prefixo “\_”. Exemplo: “\_servicoBase”);
  + Parâmetros de métodos;

### **Nomeação de projetos**

Nome de projetos devem possuir a seguinte estrutura:

<Empresa>.<Projeto>.<Componente>

Exemplo:

Dlls.Comcer.Api

### **Nomeação de enumeradores**

Enumeradores devem possuir o prefixo “Enum<Nome>”

Exemplo:

EnumTipoDePessoa.cs

### **Nomeação de interfaces de serviço**

Interfaces de serviço devem possuir o prefixo “IServicoDe<Nome>”.

Exemplo:

IServicoDeComanda.cs

### **Nomeação de interfaces de repositório**

Interfaces de repositório devem possuir o prefixo “IRepositorioDe<Nome>”.

Exemplo:

IRepositorioDeComanda.cs

### **Nomeação de implementações de serviço**

Implementações de serviço devem possuir o prefixo “ServicoDe<Nome>”, e o sufixo “Impl”.

Exemplo:

ServicoDeComandaImpl.cs

### **Nomeação de implementações de repositório**

Interfaces de repositório devem possuir o prefixo “RepositorioDe<Nome>”, e o sufixo “Impl”.

Exemplo:

RepositorioDeComandaImpl.cs

### **Nomeação classes de validação**

Classes de validação devem possuir o prefixo “ValidacoesDe<Nome>”

Exemplo:

ValidacoesDeComanda.cs

### **Nomeação classes de conversores**

Classes de conversores devem possuir o prefixo “ConversorDe<Nome>”

Exemplo:

ConversorDeComanda.cs

# Tratamento de Erros e de Exceções

[Esta seção fornece regras e diretrizes sobre como os erros e as exceções devem ser tratados. Ex.:

* Criar uma casse de tratamento de exceções na camada de apresentação.
* Garantir que as exceções subam para a camada de apresentação, usando a clausula Trows nos métodos envolvidos.

]

# teste unitário

* Toda classe de validação deve possuir testes unitários;
* Todo conversor deve possuir testes unitários;

# Diretrizes Gerais

[Esta seção deve descrever, caso existam, diretrizes e estratégias a serem usadas no desenvolvimento].