PADRÃO DE CODIFICAÇÃO

APLICAÇÕES .NET C#

Sumário

1. Histórico de revisões

Data	Versão	Descrição	Autor (es)
14/09/2018	1.0	Versão Inicial	Marcos Tadeu Torres Álvaro Menezes

2. Resumo

Este documento descreve os padrões de codificação que devem ser adotados pelos desenvolvedores do Instituto de Tecnologia da Informação e Comunicação do Estado do Espírito Santo – PRODEST. Estão incluídos padrões de nomenclatura, identação e de estruturação de classes, além das definições de alguns indicadores de qualidade do código-fonte. O documento é voltado para a linguagem C# .NET.

3. Convenções de Nomenclatura

Nesta seção descrevemos as regras para padronização de nomes utilizados no código.

3.1. Estilos de nomenclaturas (Notação)

3.1.1. Pascal Case

É a denominação em inglês para a prática de escrever palavras compostas ou frases, onde a primeira letra do identificar e a letra inicial de cada palavra dentro do identificador devem estar em maiúsculo. Usada para todos os identificadores, exceto os nomes de parâmetro e variáveis locais.
Exemplos:

```
1. PropertyDescriptor
2. HtmlTag
```

Um caso especial é feito para as abreviações de duas ou mais letras em que as letras iniciais estão em maiúsculas, conforme mostrado no seguinte identificador:

```
1. IOStream
2. CPF
```

3.1.2. Camel Case

É a denominação em inglês para a prática de escrever palavras compostas ou frases, onde a primeira letra de cada palavra dentro do identificador deve estar em maiúsculo, todas as outras em minúsculo, inclusive a primeira letra do identificador. Usada somente para os nomes de parâmetro e variáveis locais.
Exemplos:

```
1. propertyDescriptor
2. htmlTag
3. ioStream
4. cpf
```

3.2. Regras Gerais de Nomeclatura

- Não utilizar acentos.
- Não utilizar espaços.
- Não utilizar sublinhado (subtraço), hifens, ou qualquer outro caractere não alfanumérico.
- Não é recomendado utilizar abreviações. Siglas são aceitáveis. Embora a clareza nos nomes seja essencial mesmo que o nome fique longo. Sendo assim, siglas só devem ser utilizadas quando forem uma referência muito comum no modelo de negócio sendo tratado, por exemplo: CPF (Cadastro de Pessoas Físicas).
- Usar PascalCasing para todos os membros, tipos e namespaces públicos onde os identificadores são palavras compostas ou frases.
- Usar camelCasing para nomes de parâmetro.
- Qualquer API acessada externamente não deve depender da diferenciação entre maiúsculas e minúsculas para distinguir dois nomes no mesmo contexto. C# é uma linguagem case-sensitive, mas isso não pode ser assumido como regra para qualquer linguagem externa.
- Não utilizar notação húngara.
- Prefixar os nomes das interfaces com I, para indicar que o tipo é uma interface.

3.3. Namespace

A regra geral para nomenclatura de namespaces é:

```
<EmpresaOuSecretaria>.(<Produto>|<Sistema>)[.<Função|Módulo>][.<Subnamespace>]
```

4. Estilo

4.1. Geral

- Caracteres de tabulação não devem ser utilizados.
- Nunca declare mais de um namespace por arquivo.
- Evite colocar mais de uma classe por arquivo.
- Não codifique o texto de mensagens direto no código. Use arquivos de recurso (resource files).
- Sempre use chaves ({ e }) numa nova linha em instruções condicionais. Ex:

```
protected void Page_Load(object sender, EventArgs e)
{
      bool IsTrue;
      if (IsTrue == true)
      {
          // Do something;
        //....
        return false;
      }
}
```

- Use uma linha branca para separar grupos lógicos de código. Ex:

```
1.  public void SayHello(string name)
2.  {
3.      string fullMessage = "Hello " + name;
4.      DateTime currentTime = DateTime.Now;
5.
6.      string message = fullMessage + ", time:" + currentTime.ToShortTimeString();
7.
8.      MessageBox.Show(message);
9.  }
```

- Use regiões (#region) para agrupar partes relacionadas de código. Ex:

```
1.  namespace SampleApplication
2.  {
3.   public partial class _Default : System.Web.UI.Page
4.   {
5.
6.     #region Private Members
7.     // code here
8.     #endregion
9.
10.    #region Constructors
11.    // code here
12.    #endregion
13.
14.    #region Public Properties
15.    // code here
16.    #endregion
17.
18.    #region Private Properties
19.    // code here
20.    #endregion
21.
22.    #region Public Methods
23.    // code here
24.    #endregion
25.
26.    #region Private Methods
27.    // code here
28.    #endregion
```

- Não se devem utilizar blocos no interior de métodos. A possível necessidade de organização de código nestes casos deve ser superada com a criação de novos métodos e com a utilização de blocos para agrupar estes métodos. Ex:

```
1.  #region Controle do Formulário
2.  public void metodoInicio()
3.  {
4.      ...
5.      metodoInterno();
6.  }
7.  private void metodoInterno()
8.  {
9.      ...
10. }
11. #endregion
```

4.2. Comentário

- Comentário deve estar no mesmo nível do código (mesma indentação), mas numa linha separada, nunca no final da linha de código. Ex:

```
1.  // Format a message and display.
2.  string fullMessage = "Hello " + name;
3.  DateTime currentTime = DateTime.Now;
```

- Insira um espaço entre o delimitador de comentário e o texto do comentário. Ex:

```
1.  // The following declaration creates a query. It does not run
2.  // the query.
```

5. Qualidade

A qualidade do código-fonte será medida e avaliada através de análise estática de código com a ferramenta SonarQube 7.0 ou superior. Alguns dos indicadores de qualidade do código-fonte que podem ser utilizados estão relacionados a seguir:

Grupo	Indicador	Unidade	Meta Sugerida
Arquitetura	Directory Tangle Index (ciclos de dependências entre pacotes e classes)	%	<= 2%
Projeto	Complexity / file	média total	<= 10
	Complexity / class	média total	<= 10
	Complexity / function	média total	<= 3
	Duplications	%	<= 4%
	Security Issue Tags	unidades	= 0
	Technical Debt ratio	%	<= 2,5%
	Sqale Rating	Nota	= A
Violações de código (possíveis bugs, estilo de codificação, más práticas de codificação):	Rules Compliance Index	%	>= 95%
	Critical Issues	unidades	= 0
	Blocker Issues	unidades	= 0
	Unit Tests Coverage - camada negócio	%	>= 70%

Indicadores relacionados a testes	Unit Test Success	%	= 100%
	Skipped Tests	unidades	= 0

6. Referências

- https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines
- https://rules.sonarsource.com/csharp