Estándar de codificación

para la

construcción del

Italian Pizza

Versión 1.0

Preparada por:

Calderón Blas Javier Alberto

Camarillo Vila José Daniel

Domínguez García Antonio de Jesús

Alejo Barrientos Rubén Isaí

19 de octubre de 2021

Índice

Índice de tablas

1. Introducción

Para poder desarrollar un software exitoso, es importante que los equipos de desarrollo adopten procesos que formen parte de la ingeniería de software, utilizando estándares y metodologías para diseñar, programar, probar y analizar el software desarrollado con el fin de brindar mayor confiabilidad, mantenibilidad y cierto nivel de calidad a partir de requisitos identificados previamente. Asimismo, también se garantiza una coherencia dentro del código, haciéndolo más fácil más manejables en sentido de entendimiento y desarrollo, beneficiando los aspectos de eficiencia de producción y capacidades de control del equipo, abarcando desde la construcción hasta su despliegue.

2. Propósito y Alcance

Una forma de producir código de calidad es desarrollando un estándar de programación donde se especifiquen requerimientos de como debe ser escrito dicho código, abarcando aspectos como el estilo, estándares para métodos, convenciones de nomenclatura etc. Además, es importante que el equipo este en constantes revisiones para verificar que el estándar este siendo cumplido. En este documento se encuentran técnicas de programación y la colección de prácticas de codificación que se llevarán acabo para la realización de aplicaciones en el lenguaje de C#.

3. Estilos de código fuente

3.1 Sangría

Se debe usar un estilo de sangría de lenguajes de programación tabular o delimitar bloques lógicos de código para que así el código realizado sea más legible.

Ejemplo correcto	Ejemplo de violación a la regla

`public Item (int id, string category, int price)`
`{`
`  Id = id;`
`  Category = category;`
`  Price = price;`
`}`
	`public Item(int id, string category, int price){`
	`Id = id;`
	`Category = category;`
	`Price = price;`
	`}`

Tabla 1. Estilos de código fuente: Sangría

3.2 Espacios en blanco

Los espacios en blanco serán utilizados en las siguientes situaciones y condiciones:

- Tras una palabra reservada y su paréntesis asociado.
- Nunca deben separar operadores unarios como “**++**” y “**--**” a partir de sus operandos.
- Tras signos cómo: coma, operador menor que o mayor que, punto y coma, igual.

Ejemplo correcto	Ejemplo de violación a la regla

`for (int i = 0; i < 10; i++)`
`{`
`  declaración`
`}`
	`for(int i=0;i<10;i+ +){`
	`  declaración`
	`}`

Tabla 2. Estilos de código fuente: espacios en blanco

3.3 Llaves de apertura y cierra

La llave inicial debe estar en la siguiente línea de la estructura al igual que la llave de cierre, con la misma sangría para que coincida con su declaración de apertura correspondiente.

Ejemplo correcto	Ejemplo de violación a la regla

`public Item(int id, string category, int price)`
`{`
`  Id = id;`
`  Category = category;`
`  Price = price;`
`}`
	`public Item(int id, string category, int price){`
	`  Id = id;`
	`  Category = category;`
	`  Price = price;`
	`}`

Tabla 3. Estilos de código fuente: llaves de apertura y cierre

3.4 Instrucciones por la línea de código

En cada línea de código sólo irá una instrucción para así evitar que la comprensión del código sea dificultosa.

Ejemplo correcto

`if (expresion)`
`{`
`  i++;`
`  Console.Write(i);`
`}`
`else`
`{`
`  i--;`
`  Console.Write(i);`
`}`

Tabla 4. Estilos de código: instrucciones por línea de código

3.5 Comentarios de implementación

Los códigos deben tener comentarios de implementación delimitados por /*…*/ o //. Para comentar el código de debe usar una barra doble, es decir, //, al igual que para comentarios de una o varias líneas.

Los comentarios se realizarán por bloques, minimizando los comentarios por líneas, salvo aclaraciones en un código complejo o para reflejar la importancia de una línea dentro de la ejecución del código. Estos comentarios se harán al final de la línea.

Evitar comentarios que expliquen cosas obvias, en la mayoría de las cosas el código debe ser auto explicativo. Un buen código con buenas prácticas de nombrado no debe ser comentado.

Ejemplo correcto	Ejemplo de violación a la regla

`//------------------------------------------------------------------------------`
`// <auto-generated>`
`//`
`//  Este código se generó a partir de una plantilla.`
`//`
`// </auto-generated>`
`//------------------------------------------------------------------------------`
	`/*`
	`------------------------------------------------------------------------------`
	`<auto-generated>`
	`  Este código se generó a partir de una plantilla.`
	`</auto-generated>`
	`------------------------------------------------------------------------------`
	`*/`

Tabla 5. Estilos de código: comentarios de implementación

3.6 Estructuras de control

3.6.1 Declaraciones if

La palabra clave 'if' y la expresión condicional deben colocarse en la misma línea. Las declaraciones if deben siempre usar llaves de apertura y cierre.

Ejemplo correcto

`if (expresion)`
`{`
`  instrucciones;`
`}`
`else`
`{`
`  instrucciones;`
`}`

Tabla 6. Estructuras de control: declaraciones if

3.6.2 Declaraciones for

Una declaración for debe tener la siguiente forma:

Ejemplo correcto

`for (int I = 0; i < 10; i++)`
`{`
`  Console.WriteLine(i);`
`}`

Tabla 7. Estructuras de control: declaraciones for

La palabra clave 'for' y el paréntesis deben estar separados por un espacio y las expresiones en una declaración deben ser separadas por espacio en blanco como se indica en el apartado 3.2. El bloque de instrucciones se coloca en la siguiente línea.

3.6.3 Declaraciones while

La sentencia 'while' usa el mismo formato de diseño que la construcción 'if'. Debería aparecer la palabra clave "while" en su propia línea, seguida inmediatamente por la expresión condicional. La palabra clave ‘while’ y los paréntesis deben estar separados por un espacio. El bloque de instrucciones se coloca en la siguiente línea.

Ejemplo correcto

`while (expresión)`
`{`
`  instrucciones;`
`}`

Tabla 8. Estructuras de control: declaraciones while

3.6.4 Declaraciones do-while

La forma do-while de la construcción while debería aparecer como se muestra a continuación:

Ejemplo correcto

`do`
`{`
`  instrucciones;`
`}`
`while (expresión);`

Tabla 9. Estructuras de control: declaraciones do-while

3.6.5 Declaraciones try-catch

En la construcción try-catch, la palabra clave 'try' debe ir seguida de la llave abierta en su propia línea, luego deberá ir el cuerpo de la declaración y la llave de cierre en su propia línea. Seguido se coloca el bloque 'catch' con las sentencias en su cuerpo;

- No debe haber ningún bloque try-catch vacío.
- Ordenar la captura de excepciones (catch) siempre en orden descendente desde la más particular hasta la más genérica.
- Evitar anidar bloques try/catch

Ejemplo correcto

```
public static void Main()
{
      string s = null; // For demonstration purposes.

      try
      {
            ProcessString(s);
      }
      catch (Exception e)
      {
            Console.WriteLine("{0} Exception caught.", e);
      }
}
```

Tabla 10. Estructuras de control: declaraciones try-catch

4. Estándares para métodos

4.1 Métodos de denominación

Los métodos deberán ser nombrados en inglés, para la primera palabra del método deberá de estar escrita en letra mayúscula y debe ser un verbo, si el método está compuesto por dos palabras, la segunda comenzará con la primera letra mayúscula y el resto en minúsculas, esto con el fin de determinar su función con solo ver su nombre.

También es recomendable utilizar sustantivos o frases nominales para nombrar un metodo

Ejemplo correcto	Ejemplo de violación a la regla

`public void PrintWelcomeMessage()`
`{`
`  Console.Write("Welcome :)");`
`}`
	`public void message()`
	`{`
	`  Console.Write("Welcome :)");`
	`}`

Tabla 11. Estándares para métodos: métodos de denominación

5. Estándares para la convención de nomenclatura

En cuanto a los nombres de las variables, las reglas del lenguaje C# son muy extensas y permiten mucha libertad, pero acostumbrarse a seguir ciertas normas facilitan la lectura y mantenimiento de procedimientos. Ya hemos dicho que los nombres en C# son sensibles a mayúsculas y minúsculas. Por ejemplo, podemos tener en un programa las variables "height", "Height" y "HEIGHT", y serán tres variables diferentes.

Se recomiendan emplear las siguientes reglas:

- En un nombre que consta de varias palabras, se recomienda colocar una por una, escribiendo en mayúscula la primera letra de cada palabra, exceptuando la primera palabra que deberá empezar en minúscula, siguiendo el lower camel case.
- Los nombres de clases e interfaces siempre comienzan con una letra mayúscula. En el caso de las interfaces, siempre deben empezar con la letra I.
- Las variables de miembro privada no deben de ser precedidas por un guion bajo.
- Los nombres de las variables deben ser descriptivas.

- Evitar el uso de tipos de datos del sistema, es preferible usar tipos de datos predefinidos:

Correcto	Incorrecto
int employeeId; string employeeName; bool isActive;	Int32 employeeId; String employeeName; Boolean isActive;

- Las variables deben estar declaradas lo más cerca posible del fragmento de código donde se usarán.
- Usar abreviaciones**.** Excepciones: abreviaciones usadas comúnmente como nombres, por ejemplo: **Id, Xml, Ftp, Uri.** Así es consistente con .NET y evita abreviaciones inconsistentes.

Correcto	Incorrecto	Excepción
public DateTime fechaModificacion; public TimeSpan tiempoTranscurrido;	public DateTime fecha_Modificacion; public TimeSpan tiempo_Transcurrido;	private DateTime _fechaCreacion;

6. Estándares para XAML

Para establecer la propiedad name a los elementos depende del tipo de elemento para su terminación. Ejemplo:

Elemento	Correcto	Incorrecto
TextBlock	SearchResultMessageTextBlock	SearchResultMessage
Grid	FoodRecipeTableGrid	FoodRecipeTable
Border	SecondLayerFilterBorder	SecondLayerFilter
RadioButton	CookFilterRadioButton	CookFilter
ComboBox	IngredientsComboBox	Ingredients

- Las etiquetas tienen que abrir y cerrar adecuadamente. Ejemplo:

```
<StackPanel>
    <Button>This example</Button>
        <StackPanel.Resources>
            <SolidColorBrush x:Key="BlueBrush" Color="Blue"/>
        </StackPanel.Resources>
    <Button>... is illegal XAML</Button>
</StackPanel>
```

- Cerrar los controles en una línea cuando no vayamos a tener propiedades dentro.

```
<TextBlock Text=”Hello world”/>
<TextBlock Text=”Hello world”></TextBlock>
```

- Uso de Grid como panel siempre que sea posible.
- Ocupar Visibility para ocultar elementos y no Opacity, un elemento con Opacity a cero sigue estando en la página, ocupa espacio en la memoria. Si ponemos Visibility=”Collapsed” el elemnto saldrá del UI.
- Uso de comentarios en código cuando sea necesario <!–Tu comentario–>