Otra guía de estilo para C#

¿Por qué?

Seguramente te estarás preguntando para qué necesitas una guía de estilo si tu ya sabes programar. Cierto, pero déjame decirte algo esperando no sonar ofensivo: **Cualquiera puede programar**. Cualquiera con unos cuantos meses de experiencia, alguno que otro curso o vídeo en internet puede escribir una aplicación funcional en poco tiempo. Hacer código que funcione es sencillo, hacer un buen código que funcione es lo complicado.

El Buen Código

Respecto al buen código, todos pueden tener una opinión distinta, sin embargo hay puntos en los que coincide cualquier definición de El Buen Código, basándonos en eso me atrevo a decir decir que es:

- Eficiente
- Mantenible
- Organizado
- Legible

Escribir El Buen Código es un arte que tu debes aprender, practicar y fomentar. Adoptar un estilo (o crear uno propio) te puede ayudar a conseguirlo.

Sobre las guías de estilo

A diferencia de lo que podría parecer, esta guía (y todas las demás que te encuentres) no son mandamientos que debes seguir estrictamente, porque tal vez haya aspectos que no se puedan integrar a tu organización por cuestiones filosóficas de la empresa, compatibilidad con sistemas antiguos o complicados errores en la Matrix. Así que toma estas guías como sugerencias que si son adoptadas por tu organización te ayudarán a mantener un código estandarizado, que sea fácilmente legible por todos miembros del equipo de desarrollo y, en caso de ser requerido, sea mantenible.

La guía

Te voy a cambiar el nombre…

Increíblemente, escoger buenos nombres para tus variables de programa, métodos y clases puede ser la diferencia entre querer revisar un código viejo o rehacer todo de nuevo. Tal vez no quieras buscar para qué usas esa variable `c` dentro del método `compraalgo` y prefieras escribir el método `compraalgo2`. No sé, piénsalo.

Recomendaciones sobre los nombres

- Usa nombres comprensibles que se autodocumenten para clases, métodos, variables, etc.

- En caso de propiedades de clase, a menos que se requiera otra lógica, procura siempre usar las propiedades autoimplementadas.
```
int Property { get; set; }
```

- En caso de ser necesarias, utiliza un _ antepuesto al nombre de una variable miembro, escrita con *camel casing*.
```
private int _clientId
private string _name;
```

- Para el caso de variables tipo booleanas, además del usa un prefijo como “can”(puede), “is”(es) o “has”(tiene).
```
public bool IsCool { get; set; }
private bool _hasChildren;
```

- No uses nombres de variables que se puedan parecer a palabras reservadas de C# (o de cualquier otro lenguaje).

- Usa nombres completos siempre que sea posible, es decir, evita abreviaturas. Si se está trabajando con siglas o acrónimos debes mantener todos sus caracteres en mayúsculas.
```
string addr; // incorrecto
string address; // correcto
void PublishHtml() { ... // correcto
void PublishHypertextTrans... // incorrecto
```

- Usa nombres de variables de una sola letra solo en caso de bucles, siempre y cuando no se use más que para contador de iteraciones.

```
for(int i = 0; i < value; i++) {
```

- Usa *pascal casing* para nombrar clases, métodos y estructuras.

```
class SharpGuide { ...
void FixMatrix(int matrixId) { ...
```

- Usa *camel casing* para los nombres de variables y parámetros de método.

```
string awesomeProgrammingLanguage = "C#";
void SetLanguage(string someLanguage) { ...
```

- Para las interfaces antepón una `I` al nombre que debe estar escrito usando *camel casing*
  `interface IStyleGuide { ...`

- En cuanto a los espacios del nombre, la convención es bastante clara, utiliza un formato como este `<company name>.<product name>.<top module>.<bottom module>`

- Los archivos deben llamarse como la clase o interfaz pública que contienen.

¿Comentarios? ¿en serio?

Tal vez acabas de escribir una poesía con tu código (y los nombres de variables y métodos) pero aun así tu programa no sea tan entendible debido a alguna lógica loca del negocio o algún truco para eficientar la ejecución, ese es el momento de usar los comentarios. O tal vez la

compañía en la que trabajas requiere que documentes todos los métodos que creaste. He aquí algunas recomendaciones para los comentarios:

- No narres tu código a través de los comentarios, o en otras palabras, no escribas un comentario por cada línea de código que escribas.

- Escribe los comentarios en líneas que no contengan código y siempre antes de la sección a la que te quieres referir. Una excepción son las declaraciones o inicializaciones de variables.

- Usa `//` o `///` para escribir comentarios. Evita `/* ... */` (y evita escribir comentarios de más de una línea).

- Tus comentarios son hermosos sin necesidad de que les hagas una cajita de `********` al rededor, así que también evita hacerlo.

- Revisa la ortografía de tus comentarios y asegúrate de que sean claros al momento de leerlos.

- Hay un comentario muy especial que debes usar cada vez que dejes código sin finalizar, ese es el comentario `// TODO: [descripción]` ya que el entorno de desarrollo, si usas Visual Studio, identificará ese comentario como una funcionalidad pendiente de acabar y esta aparecerá en la lista de tareas pendientes.

- En caso de que un método o clase no sea autodocumentable o necesites proveer de infromación específica para su uso, emplea los comentarios XML con `///` para que, en caso de que el IDE lo permita, esta documentación se muestre cuando alguien más (o tu mismo) haga uso del elemento documentado.

- Recuerda que C# permite crear documentación en XML, para lo cual es necesario que esté bien formado.

Estructura de un archivo y una clase

También hay recomendaciones con respecto a los archivos, su estructura y su contenido.

- Procura que un archivo contenga una sola clase, habrá casos en los que tal vez no se pueda, pero en general es posible.

- El archivo de una clase debe llamarse igual que la clase que contiene.

- Usa `tab` con tamaño de 4 para indentar el código en lugar de espacios

- Siempre escribe los corchetes `{ }` en líneas nuevas.

- Agrupa líneas de código de acuerdo a la labor que realizan, separa dichos grupos por una línea en blanco.

- Separa por una sola línea los métodos definidos en una clase.

- La estructura general de un archivo es:
    - Sentencias *using*
    - *namespace*
    - Declaración de la clase
    - Variables privadas
    - Propiedades privadas

    - Propiedades públicas
    - Constructores
    - Métodos públicos
    - Métodos privados
- La usa las directivas `#region` y `#endregion` para agrupar todas las categorías anteriores y sea fácil navegar entre ellas.

Algunas buenas prácticas

- Evita los métodos muy largos, procura conservar de 1 ~ 30 líneas de código. Si se pasa tal vez sea necesario revisarlo para ver si se puede separar en dos o varios métodos.

- Nuevamente, **autodocumenta** tus métodos, si el nombre es claro no hay necesidad de explicar demasiado mediante los comentarios.

- No mezcles varias tareas en un solo método. Un solo método debe hacer una sola acción.

- Prefiere el uso de los alias de tipos de dato conocidos para declarar variables. Es decir, prefiere `string awesome = "awesome!";` a `String awesome = "awesome!".`

- Por el contrario, utiliza los tipos definidos en `System` para hacer referencia a constantes o métodos de clase:
  `Int32.ParseInt("0");` mucho mejor que
  `int.ParseInt("0");`.

- No todo dentro de una clase tiene que ser público o privado, usa el modificador de acceso `internal` para propiedades y métodos si solo se accede a ellos dentro del mismo ensamblado.

- Si un método recibe más de 5 o 6 parámetros considera reemplazarlos por una estructura o una clase que contenga esos parámetros como propiedades.

- **NUNCA, nunca** *hardcodees*[1] números, cadenas o cualquier otro valor. Usa fuentes externas de datos (archivos, baess de datos, línea de comandos...).

- Utiliza `String.Empty` en lugar de las comillas vacías `""`.

- Cuando sea necesario comprobar si una cadena está vacía utiliza los métodos del framework: `String.IsNullOrEmpty` o `String.IsNullOrWhiteSpace`.

- Procura sobreescribir el método `Equals` en las clases definidas por el usuario.

Contribuciones

Soy solo un ser humano que gusta de programar, especialmente en C#, así que como bien dije, este no es un libro de reglas, solo algunas cuantas cosas que yo considero buenas prácticas. Si quieres hacer algún comentario, queja o sugerencia con respecto a esta guía, puedes hacerlo en https://github.com/fferegrino/cool-sharp/issues, o antonio.feregrino@hotmail.es

1. Mala práctica del en el desarrollo de software que consiste en incrustar datos relacionados con la lógica de la aplicación directamente en el código fuente de un programa.