PUI PUI

Estándares de programación en .Net para el desarrollo del proyecto

Versión 2.0

Grupo Integrador
Gonzales Jonathan
Gutiérrez Javier
Piñango Hernán

Pui Pui es un sistema orientado a mejorar la experiencia de los clientes de un gimnasio. Desde la gestión de clases y rutinas de ejercicio en las que un cliente puede participar hasta la gestión de clases e instructores.

TABLA DE CONTENIDO

I. Introducción

El siguiente documento describe las reglas y recomendaciones para el desarrollo de aplicaciones utilizando C# más específicamente el desarrollo de la aplicación "PUI PUI" su objetivo es definir los lineamientos y estándares necesarios, con la finalidad de ayudar a los desarrolladores involucrados a evitar errores comunes así como problemas de integración.

Este documento cubre convenciones de nombres, estilos de codificación, uso de lenguaje, mejores prácticas para aplicaciones, entre otros.

Se utilizaron como fuentes para el desarrollo de este documento los siguientes documentos:

Estándares y Codificación en C# y buenas prácticas de programación.	http://www.canaldenegocio.com/docs/MejoresPracticasDotNet.pdf
C# Coding Standards for .NET	http://www.lance-hunt.net

II. Convenciones del documento

Para la comprensión de este documento, es necesario definir estándares para asegurar la claridad y facilitar la lectura al momento de indicar las reglas y líneas guías. Ciertas condiciones son usadas a través de este documento para añadir énfasis. A continuación, algunos de los estándares para este documento:

Color y énfasis:

- Azul: El texto coloreado en azul indica palabras reservadas por el lenguaje .NET
- **Negrita**: Texto con énfasis adicional a resaltar.

Palabras claves:

- **Siempre**: Enfatiza que la regla debe ser cumplir.
- **Nunca**: Enfatiza que la acción nunca se debe de ejecutar.
- **Evitar**: Enfatiza que la acción debe de ser prevenida, pero algunas excepciones pueden existir.
- **Tratar**: Enfatiza que la regla debe de cumplirse siempre y cuando sea posible y sea adecuado.

- **Ejemplo**: Precede el texto usado para ilustrar la regla o recomendación.
- **Razón**: Explica el propósito detrás de la regla o recomendación.

III. Terminología y definiciones

Notación Camell:

Forma de escribir una palabra donde el primer carácter de todas las palabras, excepto de la primera palabra se escribe en Mayúsculas y los otros caracteres en minúsculas.

Ejemplo:

```
colorDeFondo.
```

Notación Pascal:

Forma de escribir una palabra donde el primer carácter de todas las palabras se escribe en mayúsculas y los otros caracteres en minúsculas.

Ejemplo:

```
ColorDeFondo.
```

Identificador:

El desarrollador define un nombre único para declarar un objeto o instancia de alguno.

Número Mágico:

Cualquier número usado con una expresión (o inicialización de alguna variable) que no contiene en si un significado obvio o conocido. Este usualmente excluye los enteros 0 y 1 y cualquier otro equivalente numérico que se evalúe como cero.

IV. Convenciones y estándares de nombres

Esta sección se explica cómo nombrar el proyecto, archivos del código fuente, e identificadores incluyendo campos, variables, propiedades, métodos, parámetros, clases, interfaces y espacios de nombres (Namespaces).

1. Usar notación Pascal para el nombre de las Clases.

Ejemplo:

```
public class HolaMundo
{
…
}
```

2. Usar notación Pacal para el nombre de los Métodos.

Ejemplo:

```
void DiHola (string nombre)
{
…
}
```

3. Usar notación de Camell para variables y parámetros de los métodos.

Ejemplo:

```
int cuentaTotal = 0;
void DiHola(string nombre)
{
string mensajeCompleto = “Hola “ + nombre;
…
}
```

4. **Evitar** nombres totalmente en MAYÚSCULAS o en minúsculas.

5. Usar palabras entendibles y descriptivas para nombrar a las variables. No usar abreviaciones.

6. No usar palabras reservadas para nombres de variables.

7. Usar el prefijo “Es” ó “Is” para variables de tipo boolean o prefijos similares.

Ejemplo:

```
private bool EsValido;

private bool IsActivo;
```

8. El nombre de los archivos debe coincidir con el nombre de la clase. Por ejemplo, para la clase HolaMundo el nombre del archivo debe ser HolaMundo.cs. Y usar notación Pascal para el nombre de los archivos.

9. **Nunca** crear declaraciones del mismo tipo (espacios de nombres, clases, métodos, propiedades, campos o parámetros) que varíen solo por su capitalización (Mayusculas).

10. Variables y propiedades deben describir la entidad que representan no el tipo o tamaño.

11. Nunca usar nombres que comiencen con caracteres numéricos.

12. Nunca usar notación Hungara.

Ejemplo:

```
string m_sNombre;
int nEdad;
```

13. **Evitar** el uso de abreviaturas al menos que el nombre sea excesivo.

14**. Evitar** abreviaturas que su longitud sea mayor a 5 caracteres.

15. Cualquier abreviatura debe ser totalmente conocida y aceptada por el grupo de desarrollo.

16. Usar mayúsculas en caso de abreviaciones de 2 letras, y estilo de escritura notación Pascal para abreviaturas más largas.

17**. Nunca** usar palabras reservadas para nombres.

18. **Evitar** conflictos de nombres con los espacios de nombres o tipos existentes en el .NET.

19. **Evitar** añadir redundancia en prefijos o sufijos de los identificadores.

20. **Siempre** usar el prefijo “I” con notación Camell para interfaces (interface)

Ejemplo: `IEstudiante`.

21. **Siempre** usar palabras significativas y descriptivas para nombrar variables. No usar abreviaciones.

Forma Correcta	Forma incorrecta

`string nombre`
`string direccion`
`int salario`
	`string nom`
	`string dir`
	`int sal`

22. **Nunca** usar caracteres simples para nombrar variables como i, n, s etc. Usar nombres como index, temp

Una excepción en este caso serían las variables usadas para iteraciones en un ciclo:
Bien:

```
for (int i = 0; i < count; i++)
{
      ...
}
```

23. No usar underscores (_) para nombres de variables locales.

24. Todas las variables globales deben estar precedidas con underscore (_) para que puedan diferenciarse de las variables locales.

V. Sangría y Espaciamiento

1. Usa TAB para la sangría. No uses ESPACIOS. Define el tamaño del Tab de 4 espacios.

2. Los comentarios deben estar al mismo nivel que el código (usar el mismo nivel de sangría).

```
//Formatea un mensaje y lo despliega

string mensajeCompleto = “Hola “ + nombre;

DateTime horaActual = DateTime.Now;

string mensaje = mensajeCompleto + “, la hora es: “ +

horaActual.ToShortTimeString();

MessageBox.Show(mensaje);
```

3. Las llaves ({}) deben estar en el mismo nivel que el código fuera de las llaves.

```
if ( … )
```

```
{
	//Haz algo
	// …
	return false;
}
```

4. Usar una línea en blanco para separar un grupo lógico de código

```
bool DiHola (string nombre)
{
	string mensajeCompleto = “Hola “ + nombre;
	DateTime horaActual = DateTime.Now;
	string mensaje = mensajeCompleto + “, la hora es: “ +
	horaActual.ToShortTimeString();

	MessageBox.Show(mensaje);
	if ( … )
	{
		//Haz algo
		// …
		return false;
	}
	return true;
}
```

5. Debe haber una y solo una línea en blanco entre cada método dentro de las Clases.

6. Las llaves ({}) deben estar en una línea separada y no en la misma línea del if, for etc.

```
if ( … )
{
	// Haz algo
```

```
}
```

7. **Siempre** usar un espacio simple antes y después de cada operador.

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

8. Mantener variables, propiedades y métodos **private**, en la parte superior del archivo y lo que sea **public** después.

VI. Buenas Practicas en la Programación

1. Evitar escribir métodos muy largos. Un método debe típicamente tener entre 1 a 30 líneas de código. Si un método tiene más de 30 líneas de código, se debe considerar refactorizarlo en métodos separados.

2. El nombre de los métodos debe decir lo que hace. No usar nombres engañosos. Si el nombre del método es obvio, no hay necesidad de documentación que explique qué hace el método.

```
void GuardaNumeroTelefonico (string numeroTelefono)
{
}
```

3. Un método debe tener solo 'una tarea'. No combines más de una tarea en un solo método, aún si esas tareas son pequeñas.

4. Convertir las cadenas de texto a minúsculas o mayúsculas antes de compararlas. Esto asegurará que la cadena coincida.

```
if (nombre.ToLower() == "juan")
{
// …
}
```

5. Usar String.Empty en vez de "".

```
if (nombre == String.Empty)
```

```
{
      // haz algo
}
```

6. Las rutinas que controlan los eventos (event handlers) no deben contener el código que ejecuta la acción requerida. En vez de ello, llamar a otro método desde la rutina controladora.

```
ButtonVentas_Click(Object sender, EventArgs e) Handles Button1.Click
{
      GetVentasPorMes();
}
```

7. Nunca incrustar en el código rutas o letras de dispositivos. Obtén la ruta de la aplicación programáticamente y usa rutas relativas a ella.

8. Nunca asumir que el código se ejecutará desde el disco "C:". No puedes saber si algunas usuarios lo ejecutan desde la red o desde "Z:".

9. En el arranque de la aplicación, ejecutar una clase de "auto verificación" y asegurarse que todos los archivos requeridos y las dependencias están disponibles en las ubicaciones esperadas. Verificar las conexiones a la base de datos en el arranque, si es requerido. Enviar un mensaje amigable al usuario en caso de algún problema.

10. Si el archivo de configuración requerido no se encuentra, la aplicación debe ser capaz de crear uno con valores predeterminados.

11. Los mensajes de error deben ayudar al usuario a resolver el problema. Nunca mostrar mensajes de error como "Error en la Aplicación", "Hay un error…", entre otros. En vez de ello dar mensajes específicos como "Fallo al actualizar la base de datos", sugerir lo que el usuario debe realizar: "Fallo al actualizar la base de datos. Por favor asegurarse de que la cuenta y la contraseña sean correctos".

12. Al desplegar mensajes de error, adicionalmente al decirle al usuario qué está mal, el mensaje debe también decirle lo que el usuario debe hacer para resolver el problema. En vez de un mensaje como "Fallo al actualizar la base de datos.", sugiere lo que el usuario debe hacer: "Fallo al actualizar la base de datos. Por favor asegúrate de que la cuenta y la contraseña sean correctos.".

13. Muestra mensajes cortos y amigables al usuario. Pero registrar el error actual con toda la información posible. Esto ayudará mucho a diagnosticar problemas.

14. No guardes más de una clase en un solo archivo.

15. Evitar tener archivos muy grandes. Si un solo archivo tiene más de 1000 líneas de código, es un buen candidato para refactorizar. Dividirlo lógicamente en dos o más clases.

16. Evitar pasar muchos parámetros a un método. Si tienes más de 4~5 parámetros, es un buen candidato para definir una clase o una estructura. Lo contrario destroza el consumo en memoria, más facilidad de corrupción de datos, más castigo a los ciclos del procesador... entre otros.

17. **Tratar** de usar los tipos específicos de c# (alias), en lugar de los tipos definidos en System namespace.

Ejemplo:

```
int age; (no usar Int16)
string name; (no usar String)
object contactInfo; (no usar Object)
```

18. Vigilar **Siempre** los valores inesperados o errores. Por ejemplo, si se usa parámetros con 2 posibles valores nunca asumir que si uno no coincide entonces la única posibilidad es el otro valor.

```
if ( tipoMiembro == eTipoMiembro.Registrado )
{
// Usuarios registrados hacen algo…
}
else if ( tipoMiembro == eTipoMiembro.Invitado )
{
// Usuario invitado hace algo…
}
else
{
// Un tipo de usuario inesperado. Throw an exception
throw new Exception (“Valor inesperado “ + tipoMiembro.ToString() + “’.”)
// Si introducimos un Nuevo tipo de usuario en el futuro
// podemos encontrar fácilmente el problema aquí.
}
```

19. **Nunca** usar variables de clases globales como public or protected. Mantenerlas como private.

20. **Tratar** de no pasar muchos parámetros a un método. **Evitar** pasar más de 4 parámetros.

VII. ASP.NET

1. Siempre usar css para controlar el look and feel de las páginas. Nunca especificar tipos de fuente ni tamaños dentro de las páginas. Esto ayudará a cambiar la interfaz fácilmente en el futuro si se desea.

2. Para el nombre de las páginas utilizar nomenclatura Pascal case. Además utilizar nombres que indiquen que actividad se lleva a cabo en la página y la gestión sobre la que se está trabajando. Ejemplo ConsultarInventario.aspx

VIII. COMMENTARIOS

Los buenos comentarios permitirán la mantenibilidad del sistema, sin embargo, hay algunos puntos a considerar:

1. No escribir comentarios para cada línea de código y cada variable declarada.

2. Usar // o /// para comentar. Evitar usar /* … */

3. Escribir comentarios donde sea requerido. Pero un código entendible requerirá muchos menos comentarios. Si todos los nombres de variables y métodos son significativos el código será más entendible y necesitará menos comentarios.

4. No escribir muchos comentarios si el código es fácilmente entendible. Esto debido a que si posteriormente se cambia el código y no se cambian los comentarios, esto puede generar confusión

5. Pocas líneas de comentario harán el código más elegante. Pero si el código no es entendible y hay pocos comentarios no será de beneficio.

6. Si inicializan una variable numérica con un valor por ejemplo con 0, -1 etc, documenten la razón de esta elección.

IX. MANEJO DE EXCEPCIONES

1. Nunca hacer 'catch’ a una excepción y no hacer nada. Si esconden una excepción nunca se sabrá si ocurrió o no.

2. En el caso de las excepciones, dar un mensaje amigable para el usuario, sino registrar el error real con todos los detalles posibles sobre el error, incluyendo el momento en que ocurrió, el método y nombre de la clase, etc.