# **Buenas Prácticas para Desarrollar Código en C# Basado en Clean Code**

## **1. Convención de Nombres (Naming Convention)**

Uno de los principios más importantes del *Clean Code* es que el código debe ser fácil de leer y entender. El uso correcto de las convenciones de nombres es esencial para cumplir con este principio. A continuación, se presentan algunas de las mejores prácticas al respecto.

### **1.1. Nombres de Variables**

* **Descriptivos y significativos**: Los nombres de las variables deben indicar claramente su propósito. Evita nombres genéricos o abreviaciones que puedan ser confusas para otros desarrolladores.
  + ❌ var a = 10;
  + ✔️ var numeroDeEstudiantes = 10;
* **Camel Case**: Utiliza *camelCase* para las variables locales y parámetros de método. Esto significa que la primera letra debe ser minúscula y cada palabra adicional comienza con mayúscula.
  + ✔️ int numeroDeProductos;
* **Evita nombres de una sola letra**: A menos que sea en bucles pequeños (por ejemplo, i en un for), evita nombres de una sola letra.
  + ❌ int x;
  + ✔️ int contadorDeUsuarios;

### **1.2. Nombres de Métodos**

* **Descriptivos y verbales**: Los nombres de los métodos deben describir la acción que realizan, utilizando verbos que indiquen claramente su comportamiento.
  + ❌ void Calculo();
  + ✔️ void CalcularSuma();
* **Pascal Case**: Utiliza *PascalCase* para los métodos, es decir, cada palabra debe comenzar con mayúscula.
  + ✔️ public void EnviarNotificacion();
* **Prefiere nombres de métodos cortos pero claros**: Los métodos no deben ser excesivamente largos, pero deben ser lo suficientemente descriptivos como para que quede claro lo que hacen.
  + ❌ public void EjecutarOperacionMatematicaSumaDeNumeros();
  + ✔️ public void SumarNumeros();

## **2. Declaración de Variables**

### **2.1. Inicialización en el momento adecuado**

* **Declara e inicializa variables solo cuando sea necesario**: No declares variables si no van a ser utilizadas de inmediato. Esto mejora la claridad y la gestión de memoria.
  + ❌ int resultado; // declarada pero no utilizada por varias líneas de código
  + ✔️ int resultado = SumarNumeros(); // declarada y usada inmediatamente

### **2.2. Constantes y Variables Inmutables**

* **Usa const o readonly cuando sea posible**: Si una variable no va a cambiar su valor después de ser inicializada, utiliza const (para valores conocidos en tiempo de compilación) o readonly (para valores conocidos en tiempo de ejecución).
  + ✔️ const int MaximoIntentos = 5;
  + ✔️ readonly DateTime fechaCreacion = DateTime.Now;

### **2.3. Usa tipos var cuando el tipo es obvio**

* **var es útil cuando el tipo es obvio y no compromete la claridad**: Sin embargo, no lo uses cuando el tipo no sea inmediatamente evidente.
  + ✔️ var usuarios = new List<Usuario>(); // Claramente es una lista de Usuario
  + ❌ var resultado = Procesar(); // ¿Qué tipo es resultado?

## **3. Declaración de Métodos**

### **3.1. Métodos pequeños y específicos**

* **Los métodos deben hacer una sola cosa**: Un método debe tener una responsabilidad clara y hacer solo una cosa. Si un método hace más de una cosa, divídelo en métodos más pequeños.
  + ❌ void ProcesarDatosYGuardarEnBaseDeDatos();
  + ✔️ void ProcesarDatos();
  + ✔️ void GuardarEnBaseDeDatos();

### **3.2. Evita parámetros excesivos**

* **Reduce la cantidad de parámetros en los métodos**: Si un método tiene más de tres parámetros, puede ser un indicativo de que debería ser refactorizado o que debes considerar agrupar los parámetros en un objeto.
  + ❌ void CrearUsuario(string nombre, string apellido, int edad, string direccion, string telefono, string correo);
  + ✔️ void CrearUsuario(Usuario nuevoUsuario);

### **3.3. Uso adecuado de los modificadores de acceso**

* **Privacidad primero**: Siempre que sea posible, declara métodos como private. Solo los métodos que realmente deban ser accesibles desde otras clases deben ser public.
  + ✔️ private void ValidarDatos();

### **3.4. Uso de métodos asíncronos**

* **Usa async y await cuando sea necesario**: Si tu método realiza operaciones asíncronas como llamadas a base de datos o a servicios externos, usa la programación asíncrona para no bloquear el hilo principal.
  + ✔️ public async Task GuardarDatosAsync() { await GuardarEnBaseDeDatos(); }

## **4. Principios Generales del Código Limpio**

### **4.1. Evita comentarios innecesarios**

* **El código debe ser autoexplicativo**: Si el código es claro y está bien nombrado, los comentarios deben ser mínimos. Los comentarios a menudo se utilizan como un indicativo de que el código no es lo suficientemente claro.
  + ❌ // Suma dos números int resultado = a + b;
  + ✔️ int sumaDeNumeros = a + b; // si el comentario aporta información adicional relevante

### **4.2. Código autoexplicable**

* **Usa nombres que hagan que el código sea autoexplicativo**: Los nombres deben describir claramente la intención. Asegúrate de que cualquier desarrollador que lea tu código pueda entender su propósito sin necesidad de demasiadas explicaciones.

### **4.3. DRY (Don't Repeat Yourself)**

* **Evita la duplicación de código**: Si ves que estás repitiendo código, considera extraerlo en un método o una clase reutilizable.