Contenido

Introducción

El mundo de la programación es cada vez más complejo y competitivo. Nuevos lenguajes programáticos nacen a medida que las tecnologías mejoran y los equipos se sofistican. Por ello, no está de más echar un vistazo a una serie de buenas prácticas para programadores.

Las buenas prácticas de programación son un conjunto de normas y pautas que los desarrolladores siguen para escribir código más legible, mantenible y eficiente. Estas prácticas no solo benefician al propio programador, sino que también hacen que el código sea más fácil de entender y modificar para otros miembros del equipo.

1. Clean Code

El término clean code se atribuye al ingeniero de software Robert Cecil Martin, que lo utilizó en su libro Clean Code: Refactoring, Patterns, Testing and Techniques for Clean Code para referirse al código limpio.

El clean code no es un conjunto de reglas estrictas, sino una serie de principios que ayudan a producir código intuitivo y fácil de modificar. En este contexto, intuitivo significa que cualquier desarrollador profesional pueda entenderlo de inmediato. Un código fácilmente adaptable tiene las siguientes características:

- La secuencia de ejecución de todo el programa sigue una lógica y tiene una estructura sencilla.
- La relación entre las diferentes partes del código es claramente visible.
- La tarea o función de cada clase, función, método y variable es comprensible a primera vista.

Un código se considera fácil de modificar cuando es flexible y ampliable, lo que también ayuda a corregir los posibles errores que pueda tener. Por todo ello, el código limpio es muy fácil de mantener y presenta las siguientes propiedades:

- Las clases y los métodos son reducidos y, si es posible, tienen una sola tarea clara.
- Las clases y los métodos son predecibles, funcionan como se espera y son de acceso público a través de API (interfaces) bien documentadas.
- El código ha sido sometido a pruebas unitarias.

Nombres descriptivos

La idea de este principio es que las variables, funciones, clases, métodos, etc... tienen que tener un nombre que exprese su intención. Es decir, solo con leer el nombre de un objeto deberíamos saber cuál es su propósito. Puede parecer algo insignificante, pero es muy importante para entender el código. Un ejemplo muy sencillo, pero con el que se entiende bien el concepto es el siguiente: si vamos a declarar una variable para almacenar el campo ZLSCH (vía de pago) de la tabla BSID, no lo llamemos lv_zlsch sino lv_viaPago.

Si, por ejemplo, una función necesita tener un nombre largo para que se entienda su función, adelante con ese nombre. No hay problema porque un objeto tenga un nombre largo, porque así evitaremos tener que adentrarnos en la función para saber qué es lo que hace.

Para la nomenclatura de los objetos, podemos usar la práctica de escritura CamelCase. Consiste en unir dos o más palabras sin dejar espacios entre ellas y diferenciándolas en que la primera letra de cada palabra la ponemos en mayúscula (excepto la primera).

Funciones

Una regla fundamental es que una función realice una sola cosa, y como hemos dicho anteriormente, el nombre de dicha función tiene que indicar cuál es esa cosa. Con ello generamos que cualquier programador pueda saber lo que realiza la función sin tener que mirar su código.

Tiene que ser lo más pequeña posible. Si una función es más grande de lo que debería, lo ideal es generar funciones que hagan distintas partes de ese código. Gracias a ello conseguimos que el código sea más reutilizable.

También es aconsejable usar el menor número posible de argumentos de entrada/salida, facilitando así entender lo que hace una función y también provocará que el tiempo usado para probarla sea menor porque las combinaciones posibles también son menores.

Estructura del código

Estructurar de manera clara el código provoca que sea más fácil de leer y por tanto de mantener. A continuación, expongo ideas de cómo estructurar el código:

- Nuestras diferentes variables deben declararse todas juntas al principio de la función, método, etc...
- Cada sentencia del código debe ir en una línea.
- Por lo tanto, las sentencias relacionadas deben estar en líneas consecutivas para separarlas del resto de código.
- Cada grupo de sentencias relacionadas deben separarse por líneas en blanco.
- Aunque no haya establecido un límite máximo de caracteres por línea, no es aconsejable escribir líneas de código muy largas porque dificulta la lectura rápida del código.
- Es importante tabular el código correctamente.

Ventajas de usar Clean Code

Son múltiples las ventajas de usar estas técnicas. A continuación, comento algunas de ellas:

- El código es más fácil de leer para cualquier persona, sobre todo para aquella que no es quien lo ha desarrollo, es muy útil cuando se trabaja en equipo.
- El código es más fácil de mejorar y por tanto de mantener.
- Una situación bastante común es, debido a no entender bien el código ya desarrollado, implementar un nuevo cambio añadiendo código al final del objeto. Esto se evita utilizando estas técnicas, ya que el tiempo empleado en ver el punto a tocar es menor si se usan estas técnicas. De esta manera, añadimos el nuevo código en el punto correcto y en poco tiempo.
- La calidad del código aumenta considerablemente y, por tanto, la calidad de nuestro trabajo.
- Usando estas técnicas, se benefician no solo los demás, sino también nosotros mismos. Piensa que puedes ser tú mismo quien tenga que modificar un objeto (por ejemplo) un año después de la última modificación. ¿Crees que te acordaras de porque escribiste esas líneas de código? De esta manera, solo tendrás que leer el código para entenderlo y podrás realizar las modificaciones sin ningún problema.
- En definitiva, una ventaja esencial es que con estas prácticas generamos código limpio para todo el mundo.

2. Principio KISS

KISS es un principio de diseño muy popular tanto en el desarrollo de software como en otros muchos ámbitos. Es el acrónimo de «Keep It Simple, Stupid», y es una forma coloquial de decir que las cosas sencillas funcionan mejor.

Es uno de los mejores consejos se le puede dar a cualquier persona que diseñe o cree algo, ya sea relacionado con el desarrollo del software o no: hazlo sencillo, evita complicaciones innecesarias.

Y ya centrándonos en el desarrollo de software, evitar aumentar el nivel de complejidad cuando no es estrictamente necesario trae consigo múltiples ventajas, entre las que destacar la mejora de la comprensión y el mantenimiento, sin olvidar la sensación de satisfacción de haber escrito un código elegante.

3. Principio DRY

DRY hace énfasis en la importancia de evitar la duplicidad en el código fuente, siendo el acrónimo de «Don't repeat yourself» y estando sustentado por la afirmación de que cada pedazo de código debe tener una representación única, inequívoca y autoritaria en el sistema.

El principio Don't Repeat Yourself (DRY) es de naturaleza muy similar al principio KISS. También tiene un significado bastante simple pero amplio.

DRY nos recuerda que cada comportamiento repetible en el código debe estar aislado (por ejemplo, separado en una función) para su reutilización. Cuando tienes exactamente las mismas dos piezas de código en tu base de código, eso no es bueno, debido a que a menudo conduce a la de sincronización y otros errores, sin mencionar el hecho de que aumenta el tamaño del programa.

4. Principio YAGNI

El principio YAGNI (you aren't gonna need it o "no lo vas a necesitar") del clean code se basa en la siguiente idea: un desarrollador solo debe añadir funciones al código cuando sea estrictamente necesario. YAGNI está íntimamente relacionado con los métodos del desarrollo ágil de software. De acuerdo con este principio, en lugar de comenzar a programar partiendo de un concepto general, la arquitectura del software se desarrolla paso a paso para poder reaccionar a cada problema de forma dinámica. En otras palabras, se crea clean code cuando los problemas subyacentes se resuelven de la manera más eficiente posible.

Aplicar YAGNI de manera estricta en los proyectos de desarrollo de software aportará numerosos beneficios en nuestro trabajo del día a día, los proyectos de software y en el equipo de desarrollo como:

- Ahorro de tiempo y recursos
- Simplificación del código
- Mejora de la mantenibilidad del proyecto
- Reducción de errores y bugs

5. Estándares de C#

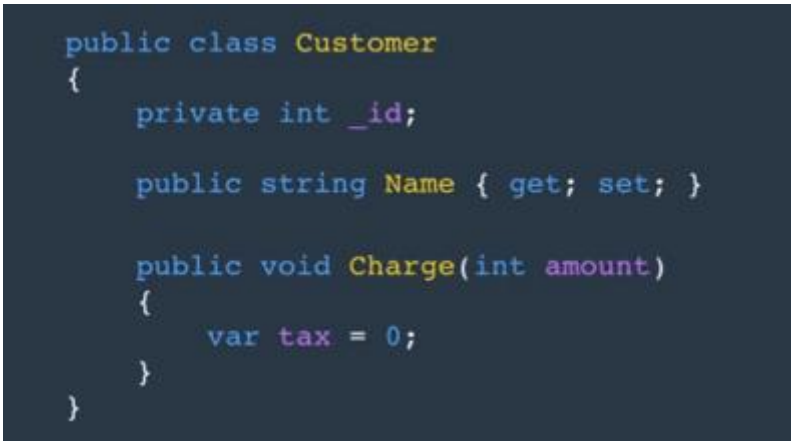
En las secciones siguientes se describen las prácticas que sigue el equipo de documentación de .NET para preparar las muestras y ejemplos de código. En general, siga estos procedimientos:

- Use las características modernas del lenguaje y las versiones de C# siempre que sea posible.
- Evite construcciones de lenguaje obsoletas.
- Solo detecta excepciones que se pueden controlar correctamente; evite detectar excepciones generales. Por ejemplo, el código de ejemplo no debe detectar el tipo de System.Exception sin un filtro de excepción.
- Use tipos de excepción específicos para proporcionar mensajes de error significativos.
- Use consultas y métodos LINQ para la manipulación de recopilación para mejorar la legibilidad del código.
- Use una programación asincrónica con async y await para operaciones enlazadas a E/S.
- Tenga cuidado con los interbloqueos y use Task.ConfigureAwait cuando corresponda.
- Use las palabras clave del lenguaje para los tipos de datos en lugar de los tipos de tiempo de ejecución. Por ejemplo, use string en vez de System.String o int en lugar de System.Int32. Esta recomendación incluye el uso de los tipos nint y nuint.
- Utilice int en lugar de tipos sin signo. El uso de int es común en todo C#, y es más fácil interactuar con otras bibliotecas cuando se usa int. Las excepciones son para la documentación específica de los tipos de datos sin firmar.
- Use var solo cuando un lector pueda deducir el tipo de la expresión. Los lectores ven nuestros ejemplos en la plataforma de documentos. No tienen sugerencias pasando el ratón por encima ni las herramientas que muestran el tipo de variables.
- Escriba el código pensando en su claridad y simplicidad.
- Evite la lógica de código demasiado compleja y enrevesada.

Convención de nombres

Muchos piensan que nombrar cosas es difícil, lo es. Cada uno de los lenguajes tienen sus propias convenciones de nombres, por ejemplo, la de java es muy similar a la de C#, pero, no es exactamente la misma. En C# tenemos básicamente 3 que son las más populares, esto no quita que cada equipo pueda hacer las suyas, pero la comunidad en la mayoría de los casos sigue estas 3:

- PascalCase: se utiliza para nombres de archivo, espacios de nombres, clases, métodos y miembros públicos.
- camelCase: utilizada para miembros no públicos o privados.
- UPPER_CASE: se utiliza para nombrar variables constantes o enumeraciones.



6. Fuentes de información

BillWagner. (s/f). Convenciones de código de .NET - C#. Microsoft.com. Recuperado el 12 de marzo de 2025, de https://learn.microsoft.com/es-es/dotnet/csharp/fundamentals/coding-style/coding-conventions

Cano, I. R. (2022, noviembre 29). Clean Code: ¿Qué es y por dónde empezar? Viewnext.com. https://www.viewnext.com/clean-code-que-es-y-por-donde-empezar/

Clean code: principios, ventajas y ejemplos. (s/f). IONOS Digital Guide. Recuperado el 12 de marzo de 2025, de https://www.ionos.mx/digitalguide/paginas-web/desarrollo-web/clean-code-que-es-el-codigo-limpio/

El principio DRY No te repitas! (s/f). Codeyourapps.com. Recuperado el 12 de marzo de 2025, de https://codeyourapps.com/el-principio-dry-no-te-repitas/

Kurth, M. (2024, febrero 14). Buenas prácticas de la programación. Instituto Profesional Providencia. https://ipp.cl/tecnologia-y-desarrollo/buenas-practicas-de-la-programacion/

Principio YAGNI: evita la sobreingeniería. (s/f). Arsys. Recuperado el 12 de marzo de 2025, de https://www.arsys.es/blog/principio-yagni-evita-la-sobreingenieria

Sonego, F. (2022, enero 22). Estándares que todo desarrollador de C# debe conocer. Without Debugger. https://withoutdebugger.com/2022/01/22/estandares-que-todo-desarrollador-de-c-debe-conocer/