ENTORNOS DE DESARROLLO

IES Santiago Hernández
Curso 2017-2018
Ignacio Agudo Sancho

12. Código Limpio

- Introducción
- Prácticas de código limpio

12. Código Limpio

- Introducción
- Prácticas de código limpio

Código Limpio

- Es una filosofía que agrupa un conjunto de ideas cuyo objetivo es hacer código más fácil de leer, mantener y extender, y menos propenso a errores.
- La responsabilidad del código limpio es 100% responsabilidad del desarrollador.
- Un código limpio se caracteriza por ser elegante, eficaz, legible, mínimo, hacer solo una cosa bien de una única manera y tener pruebas unitarias.

Código Limpio

- Teoría de las ventanas rotas:
  - “Consideren un edificio con una ventana rota. Si la ventana no se repara, los vándalos tenderán a romper unas cuantas más. Finalmente, quizás hasta irrumpan en el edificio; y, si está abandonado, es posible que lo ocupen ellos y que prendan fuego dentro.”
- El código incorrecto es el principio del desastre.
- Regla del BoyScout:
  - Dejar el código más limpio de lo que lo encontré.

12. Código Limpio

- Introducción
- Prácticas de código limpio

Código Limpio

- 1. Usa nombres con significado
  - Una de las ideas que se repite mucho es que el código tiene que poder leerse como si fuera un libro. Para ello, los nombres de variables, métodos, clases tienen que seleccionarse con cuidado para que den significado a lo que estamos intentando "contar" en nuestro programa.
  - Los nombres de las estructuras públicas (clases, funciones...) deben ser concretos y claros. Sin embargo, los privados pueden ser todo lo largos y descriptivos que necesitemos, de tal forma que expliquen claramente lo que hacen.

Código Limpio

- 2. Haz unidades de código pequeñas
  - Las clases y métodos cortos simplifican la comprensión del código y lo hacen más mantenible
  - Las funciones o clases que son muy largos se vuelven casi imposibles de entender si no estudiamos lo que hacen línea a línea, y eso alargará muchísimo el tiempo de comprensión.
  - Por el contrario, si son cortos y, combinado con el punto anterior, les damos mucho significado, la lectura será directa.
  - Se suele además recomendar la limitación del número de líneas de clases y funciones, con el fin de obligarnos a cumplir este punto. Cuánto quieras limitarlo es decisión tuya, pero como una norma general, 10-20 líneas por método y unas 500 por clase puede ser un buen baremo.
  - Se recomienda además no superar más de un nivel de anidamiento. Cada una de las ramas de un if/else por ejemplo, debería estar formado por una única línea que llame a una función.
  - La idea detrás de esto es que las bifurcaciones de código se vuelven difíciles de seguir. Si lo extraes a una función, le puedes dar un nombre con significado que explique lo que se hace en ese posible camino.

Código Limpio

- 3. Las unidades de código deben hacer una única cosa
  - Muy relacionado con el punto anterior, si nuestras estructuras de código son demasiado grandes, es porque probablemente estén haciendo más de una cosa. Es lo que se conoce como el Principio de Responsabilidad Unica, uno de los 5 principios SOLID, muy presentes en las ideas del código limpio.
  - ¿Por qué es importante este punto? Centrándonos en una clase por ejemplo, si esta hace varias cosas, tarde o temprano nos encontraremos con los siguientes problemas:
  - Esta clase será muy propensa a cambios: Aumentamos el número de razones por el que necesitaremos modificar esa clase, y por ello es más fácil que introduzcamos errores y que se vuelva excesivamente compleja.
  - Nos cuesta más testearla: al no tener bien definido lo que hace, los tests serán muy largos y seguramente poco eficaces. Además nos tocará cambiarlos muy a menudo cuando esa clase se modifique.
  - El código crecerá de forma descontrolada, y nos será muy difícil añadir nueva funcionalidad.

Código Limpio

- 4. Las funciones deben tener un número limitado de argumentos
  - Preferir las funciones sin parámetros sobre el resto. Un máximo de 3 parámetros es un buen límite.
  - Los parámetros añaden una gran cantidad de carga conceptual que dificulta la lectura. Es mucho más sencillo entender qué hace una función sin argumentos que una a la que le pasamos un parámetro, porque eso significa que el resultado va a ser diferente en función de la entrada.
  - Además dificultan el testing. Imagina el número de combinaciones posibles que existen para una función con 4 parámetros. Y para estar seguro de que funciona en cada caso, necesitarías probarlas todas.
  - Más de tres parámetros deberían estar muy justificados, y es mejor evitarlos.
  - Una alternativa sería sustituir todos (o algunos de) esos parámetros por un objeto que modele la entrada. No elimina complejidad, pero al menos será más fácil de comprender lo que se está haciendo con esos argumentos, sin necesidad de ver qué hace el código.

Código Limpio

- 5. Sigue el principio DRY: Don't Repeat Yourself
  - Repetir código es una acción bastante peligrosa que nos causará problemas tarde o temprano, y seguir este principio de diseño de software te evitará algunos quebraderos de cabeza.
  - Algunos de los puntos que surgen cuando repetimos código son:
  - Nos obliga a testear lo mismo varias veces, pues al estar en distintos lugares del programa, tendremos que validar su funcionamiento en todos ellos.
  - Cuando modifiquemos algo en un bloque, tendremos que acordarnos de cambiarlo en el resto. Obviamente, lo más probable es que nos olvidemos de ellos.
  - Amplían la cantidad de código a mantener, y eso siempre es un problema.
  - Sería tan fácil como extraerlo a una nueva clase o función, y utilizarlo desde los lugares que nos haga falta.
  - El problema de esto es que a veces la duplicidad de código no es tan evidente como podría parecer, así que tendremos que estar atentos y refactorizar cuando detectemos estos casos.

Código Limpio

- 6. Evita utilizar comentarios siempre que sea posible.
  - "Un comentario es un síntoma de no haber conseguido escribir un código claro", Robert C.Martin
  - Si necesitas añadir un comentario, es porque el código no es autoexplicativo, lo que iría en contra del primer punto.
  - Siempre que te veas en la necesidad de escribir un comentario, intenta reescribir el código o nombrar las cosas de otra forma, de tal forma que el comentario se vuelva irrelevante.
  - El problema de los comentarios es que tiene algunos problemas similares a los del código repetido. Muchas veces modificamos el código pero nos olvidamos de hacer lo propio con los comentarios. Además, el cerebro del programador es muy hábil para ignorarlos, incluso aunque estén ahí.
  - En cualquier caso, siempre habrá situaciones en las que un comentario sea mucho mejor que dejar un código complejo o algún "hack" sin explicación. Cuando trabajamos con otras librerías, APIs, Frameworks, siempre surgirán situaciones en las que necesitemos hacer algo que tengamos que explicar con palabras.

Código Limpio

- 7. Utiliza un formato único en tu código
  - Cualquier formato, por extraño que sea, es mejor que ninguno en absoluto. Dará coherencia y estructura al código, y unas pautas a seguir por todos los desarrolladores. Algunos de ellos ya los hemos mencionado, como el tamaño de las funcione y las clases. Algunas otras cosas que puedes acordar:
  - Densidad vertical: los conceptos que están relacionados deberían aparecer juntos verticalmente. Necesitarás reglas que decidan cuándo se debe dejar una línea en blanco y cuándo no.
  - Ubicación de los componentes: normalmente los campos de una clase se suelen poner al principio de la misma. También hay que decidir dónde situar las clases internas o las interfaces, en el caso de que permitáis usarlas.
  - El número de caracteres por línea: Normalmente los editores vienen marcados con una línea vertical que suele rondar los 120 caracteres. Es una buena regla a seguir.
  - Y así un largo etcétera. Cuando veas un conflicto en la forma de definir el formato de algo específico en vuestro equipo, lo mejor es definir una regla para que todo el mundo lo haga de la misma forma.

Código Limpio

- 8. Abstrae tus datos: no uses getters y setters indiscriminadamente
  - Una práctica muy común en lenguajes como Java es la de crear un objeto e inmediatamente autogenerar todos sus getters y sus setters para tener acceso y capacidad de modificación de su estado.
  - Pero las clases existen precisamente para encapsular nuestros datos y que sólo puedan ser modificados desde el exterior bajo ciertas reglas que nosotros impongamos. No nos interesa exponer los detalles de nuestros datos.
  - Aquí es importante identificar si nuestra clase es una simple estructura de datos o es un objeto con comportamiento. Los objetos esconderán sus datos mediante abstracciones y expondrán funciones para operar con ellos. Por el contrario las estructuras de datos exponen su contenido pero sus funciones no realizan ninguna operación relevante.
  - En este segundo caso sí que tendrá sentido exponer todos sus atributos con getters y setters, o incluso hacerlos públicos.

Código Limpio

- 9. ¿Conoces la Ley de Demeter?
  - Ley de Demeter: nuestro objeto no debería conocer las entrañas de otros objetos con los que interactúa
  - La Ley de Demeter es un mecanismo de detección de acoplamiento, y nos viene a decir que nuestro objeto no debería conocer las entrañas de otros objetos con los que interactúa. Si queremos que haga algo, debemos pedírselo directamente en vez de navegar por su estructura.
  - Esta ley se refiere a situaciones como esta:
    - getX().getY().getZ().doSomething()
  - El problema con este tipo de código es, primero, que necesitamos conocer las estructura de las clases para realizar una operación que debería ser directa en el punto del código en el que nos encontramos.
  - Y segundo, que este código será muy propenso a modificaciones, pues esa línea depende de 3 clases que pueden ser cambiadas en cualquier momento.
  - Normalmente la solución a la violación de esta ley suele pasar por reestructurar nuestro código, no exclusivamente crear métodos que oculten el problema. Para ello hará falta identificar quién es responsable de cada una de las acciones que se realizan y delegar en ellas le ejecución de esa tarea.

Código Limpio

- 10. Lanza excepciones en lugar de devolver códigos de retorno
  - De esta forma, se consigue un efecto doble:
    - Por un lado, no tienes que recordar operar con ese código de error y decidir qué camino tomar, pues las excepciones lo harán por ti.
    - Por otro lado, se separa fácilmente la lógica del "camino feliz" de la de errores, ya que los errores serán manejados en sus catch correspondientes.
  - Lo idóneo es crearse excepciones propias que den un significado más semántico al error de un solo vistazo, y proveerlas de mensajes de error claros y descriptivos.

Código Limpio

- 11. Establece fronteras
  - Las fronteras nos permiten establecer límites entre nuestro código y ese código que no controlamos
  - Existen muchos casos en nuestros software en los que no tenemos control sobre el código que ejecutamos, ya sea por ejemplo cuando utilizamos librerías de terceros o frameworks.
  - Las fronteras nos permiten establecer límites entre nuestro código y ese código que no controlamos, de tal forma que podamos acotar la interacción con él, y que a su vez nos permita sustituirlo sin problemas en caso de necesidad.
  - Las fronteras también nos pueden ayudar en los casos en los que tenemos que trabajar con código que aún no existe. Podemos declarar una serie de interfaces que inicialmente implementaremos con datos falsos que nos sirvan de sustituto hasta que el código definitivo esté listo.

Código Limpio

- 12. Escribe tests
  - Los tests ayudan a definir y validar la funcionalidad del software que estés implementando.
  - Hay mucha literatura sobre tests, pero un buen lugar para empezar son los tests unitarios. Suelen ser los más sencillos por los que empezar, y además deberían representar el mayor porcentaje en cuanto a cantidad. Son rápidos de ejecutar y validan los detalles de la implementación.
  - Como siempre, todas estas ideas hay que estudiarlas y utilizarlas cuando tenga sentido. Pero si empiezas a aplicarlas, conseguirás que tu código sea más fácil de leer y de mantener.

12. Código Limpio

- Introducción
- Prácticas de código limpio