

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

“Código Limpio”

“PROGRAMACION ORIENTADA A OBJETOS”

Autores:

Alcántara Cuzco, Ernesto

Bogatich Huaranga Jheyckoff

Chino Rojas, Diego

Jara Acuña, Alessandro

Asesor:

### Eric Gustavo Coronel Castillo

LIMA – PERÚ

2017

INTRODUCCIÓN

El hecho de que conozcas muchos de los conceptos de algunos paradigmas de programación como la orientación a objetos, por ejemplo; y aunque escribas miles de lí­neas de código que cumplen con un cometido, no te hacen un buen programador si es que no aplicaste buenas prácticas de desarrollo, simplemente porque tu código no será un código limpio.

CODIGO LIMPIO

Concepto según Bjarne Stroustrup (creador de c++ y autor de the c++ programming Language)

“Es aquel codigo donde logica debe ser directa para evitar errores ocultos, las dependencias deben ser minimas par facilitar el mantenimiento , el procesamiento de erroes completo y sujeto a una estrategia articulada, y el rendimiento deber ser optimo para que los usuarios no tiendan a estropear el codigo con optimizaciones sin sentido.”

Según Grady Booch , (autor de object oriented Anaysis and design with applications)

“El codigo limpio es simple y directo. El codigo limpio se lee como un texto bien escrito. El codigo limpio no oculta la intencion del diseñador sino que muestra nitidas abstracciones y lineas directas de control”

Por ultimo

Concepto seguin Dave thomas ( fundador de OTI , padrino de la estrategia eclipse)

“El codigo limpio se puede leer y mejorar por parte de un programador que no sea su autor original . Tiene pruebas de unidad y de aceptacion . Tiene nombres con sentido. Ofrece una y no varias formas de hacer algo . Sus dependencias son minimas , se definen de forma explicita y ofrece y ofrece una API clara y minima. El codigo debe ser oculto en funcion del lenguaje, ya que no toda la informacion necesaria se puede expresar de forma clara en el codigo.”

Principios a tener para el desarrollo de un código limpio

**Nombres con sentido:**

Usar nombres que revelan las intenciones, es fácil afirmar que los nombres deben revelar nuestras intenciones. Lo que queremos recalcar es la importancia de hacerlo. Elegir nombres correctos lleva tiempo pero también ahorra trabajo. Por ello, preste atención a los nombres y cámbielos cuando encuentre otros mejores. Todo el que lea su código se lo agradecerá.

El nombre de una variable , función o clase debe responder una serie de cuestiones básicas. Debe indicar porque existe, que hace y como se usa. Si un nombre requiere un comentario, significa que revela su cometido ejemplo:

Int d ; // tiempo transcurrido en días.

El nombre d no revela nada. No evoca una sensación de tiempo transcurrido, ni de días. Debe elegir un nombre que especifique lo que se mide y la unidad de dicha medida.

Nota: la elección de nombres que revelen intenciones facilita considerablemente la comprensión y la modificación el código.

**Evitar la desinformación**

Los programadores deben evitar dejar pistas falsas que dificulten el significado del código. Debemos evitar palabras cuyo significado se aleje del que pretendemos. Por ejemplo: hp, aix, y sco son nombres de variables pobres ya que son los nombres de plataformas o variantes de Unix. Aunque se trate del código de una hipotenusa y hp parezca la abreviatura correcta, puede no serlo.

No haga referencia a un grupo de cuentas como accountList a menos que realmente sea una lista (List). La palabra lista tiene un significado concreto para los programadores. Si el contenedor de las cuentas no es realmente una lista, puede provocar falsas conclusiones.

**Realizar distinciones con sentido**

Los programadores se crean un problema al crear código únicamente dirigido a un compilador o interprete. Por ejemplo, como se puede usar el mismo nombre para hacer referencia a dos elementos distintos en el mismo ámbito, puede verse tentado a cambiar un nombre de forma arbitraria. En ocasiones se hace escribiéndolo incorrectamente, lo que provoca que los errores ortográficos impidan la compilación.

No basta con añadir series de números o palabras adicionales, aunque eso satisfaga al compilador. Si los nombres tienen que ser distintos, también deben tener un significado diferente.

Los nombre de series (a1, a2, a3…an) son lo contrario a los nombres intencionados .No desinforman, simplemente no ofrecen información; son una pista sobre la intensión del autor.

**Usar nombres que se puedan pronunciar**

A los humanos se nos dan bien las palabras. Gran parte de nuestro cerebro se dedica al concepto de palabras. Y, por definición, las palabras son pronunciables. Sería una pena malgastar esa parte de nuestro cerebro dedicada al lenguaje hablado. Por tanto cree nombres pronunciables. Si no lo puede pronunciar, no podrá explicarlo sin parecer tonto.

Es un factor importante, ya que la programación es una actividad social.

**Evitar codificaciones**

Ya tenemos suficientes codificaciones como para tener que añadir otras nuevas. Al codificar información de tipos o ámbitos en un nombre se dificulta la decodificación. No parece razonable que todos los nuevos empleados tengan que aprender otro lenguaje de codificación además del código con el que van a trabajar. Los nombres codificados resultan impronunciables y suelen escribirse de forma incorrecta.

**Prefijos de miembros**

Tampoco es necesario añadir m\_ como prefijo a los nombres de variables. Las clases y funciones tienen el tamaño necesario para no tener que hacerlo, y debe usar un entorno de edición que resalte o coloree los miembros para distinguirlos.

Ejemplo:

Public class part {

Private Strin m\_dsc;// la descripcion textual

Void setName (String name){

m\_dsc=name;

}

}

A diferencia de:

Public class part {

String description;

Void setDescripcion (String descripcion) {

}

}

Ademas, los usuarios aprenden rapidamente a ignorer el prefijo o sufijo y fijarse en la parte con sentido del nombre. Cuando mas codigo leemos, menos nos fijamos en los prefijos. En ultima instancia , los prefijos son un indicio de codigo antiguo.

Nombres de clases

Las clases y los objetos deben tener nombres o frases de nombres como Cliente, evite palabras como manager , processor, data o info en el nombre de una clase. El nombre de una clase no debe ser un verbo.

Nombres de métodos

Los métodos deben tener nombres de verbo como postpayment (pago posterior), borrar página o salvar. Los métodos de acceso, de modificación y los predicados deben tener como nombre su valor y usar como prefijo get , set e is de acuerdo al estandar de javabean.

No se exceda con el atractivo

Si los nombres son demasiados inteligentes, solo los recordaran los que compartan el sentido del humor de su autor, y solo mientras se acuerden del chiste. ¿Sabrán que significa la función Holyhandgrenade? Sin duda es atractiva pero en este caso puede que deleteItems fuera más indicado. Opte por la claridad antes que por el entretenimiento. En el código, el atractivo suele aparecer como formas coloquiales o jergas. Por ejemplo, no use whack () en lugar de kill (). No recurra a bromas culturales como eatMyShorths () si quiere decir abort ().

BLOQUES Y SANGRADO

Esto implica que los bloques en instrucciones if , else ,while y similares deben tener una línea de longitud que , seguramente , sea la invocación de una función. De esta forma, no solo se reduce el tamaño de la función, sino que también se añade valor documental ya que la función invocada desde el bloque puede tener un nombre descriptivo. Tambien implica que las funciones no deben tener un tamaño excesivo que albergue estructuras anidadas. Por tanto, el nivel de sangrado de una función no debe ser mayor de uno o dos. Evidentemente, de esta forma las funciones son más fáciles de leer y entender.

Comentarios

El uso correcto de los comentarios permite compensar nuestra capacidad para expresarnos en el código. He usado la palabra incapacidad, a propósito. Los comentarios siempre son fallos. Debemos usarlos porque no siempre sabemos cómo expresarnos si ellos pero su uso no es motivo de celebración.

Cuando tenga que escribir un comentario, piense si no existe otra forma de expresarse en el código. Siempre que se exprese en el código, debe felicitarse. Siempre que escriba un comentario, debe hacer un gesto de desaprobación y sentir su incapacidad para expresarse.

Los comentarios no compensan el código incorrecto. Una de las principales motivaciones para crear comentarios es el código incorrecto. Creamos un módulo y sabemos que es confuso y estas desorganizado. Sabemos que es un desastre y entonces decidimos comentarlo. ERROR, Mejor límpielo.

El código claro y expresivo sin apenas comentarios es muy superior al código enrevesado y complejo con multitud de comentarios. En lugar de perder tiempo escribiendo comentarios que expliquen el desastre cometido, dedíquelo a solucionarlo.

Explicarse en el código

En ocasiones el código es un pobre vehículo de expresión. Desafortunadamente, muchos programadores lo entienden como que el codigo no es un buen medio de expresion. Esto es falso. ¿Qué prefiere ver? Esto:

//comprobar si el empleado tiene derecho a todos los beneficios

If { (employee.flags & hourly\_flag) &&

(employee.age>65)}

O esto otro

If (employee.isEligibleforFullBenefits())

Apenas se tardan unos segundos en explicar nuestras intenciones en el codigo. En muchos casos, basta con crear una función que diga lo mismo que el comentario que pensaba escribir.

BIBLIOGRAFIA

* MARTIN, Robert. Manual de estilo para el desarrollo ágil de software, España: Anaya, 2012,463pp.
* ISBN : 8441532109, 9788441532106