**FUNDACION UNIVERSITARIA TECNOLOGICO COMFENALCO**

**DESARROLLO DE SOFTWARE II**

**PRESENTADO POR:**

**OSCAR DAVID CASTILLO CABALLERO**

**PRESENTADO A:  
  
RONALD CARRASCAL CARREAZO  
  
  
  
  
  
  
SEMESTRE:**

**5**

**2024**

**FORMATO**

Principalmente este debe ser claro, el formato de un código es de demasiada importancia como para ignorarlo. Este se basa en la comunicación la cual deber ser el pilar de un desarrollador profesional. El estilo del código y su legibilidad establecen los precedentes que afectan a la capacidad de mantenimiento y ampliación mucho después de que el código cambie.  
  
**Formato Vertical**  
El tamaño de los archivos está relacionado con el tamaño de las clases

**Metáfora Del Periódico**  
Un archivo de código debe ser como un artículo de periódico. El nombre debe ser sencillo, pero claro. Por sí mismo, debe bastar para indicarnos si estamos o no en el módulo correcto. Los elementos superiores del archivo deben proporcionar conceptos y algoritmos de nivel superior. Los detalles deben aumentar según avanzamos, hasta que en la parte final encontremos las funciones de nivel inferior del archivo.

**Apertura Vertical Entre Conceptos**

La mayor parte del código se lee de izquierda a derecha y de arriba hacia abajo. Cada línea contiene una expresión o cláusula, y las agrupaciones de líneas forman ideas completas. Estas ideas deben separarse con líneas en blanco.

Las líneas en blanco separan la declaración del paquete, las importaciones y las funciones, ayudando a organizar visualmente el código. Estas líneas actúan como señales que indican el inicio de un nuevo concepto o sección en el código.

El efecto aumenta todavía más si no centramos la vista, La diferencia entre ambos listados es una ligera apertura vertical.  
  
**Densidad Vertical**

Si la apertura separa los conceptos, la densidad vertical implica asociaciones. Por tanto, las líneas de código con una relación directa deben aparecer verticalmente densas.

**Distancia Vertical**

Los conceptos relacionados deben mantenerse juntos en el mismo archivo para que sean fáciles de entender. Evita dividirlos entre archivos a menos que sea realmente necesario. La separación vertical dentro de un archivo debe basarse en la importancia y la legibilidad, para que el lector no tenga que saltar entre archivos o clases para comprender el código.

**Declaraciones De Variables**

Las variables deben declararse de la forma más aproximada a su uso. Como las funciones son muy breves, las variables locales deben aparecer en la parte superior de cada función.

Se debe tener en cuenta Las variables de control de bucles deben declararse en la instrucción del bucle. En casos excepcionales, una variable puede declararse en la parte superior de un bloque o antes de un bucle en una función extensa.

**Variables De Instancia**

Las variables de instancia deben declararse en la parte superior de la clase, ya que suelen ser utilizadas en muchos o todos los métodos de la clase. Aunque hay diferencias en los estilos de programación, como en C++ donde a veces se colocan al final, en Java es común ubicarlas al principio. Lo esencial es que se declaren en un lugar fijo y predecible para que cualquier persona pueda encontrarlas fácilmente.

**Funciones Dependientes**

Si una función invoca otra, deben estar verticalmente próximas, y la función de invocación debe estar por encima de la invocada siempre que sea posible. De este modo el programa fluye con normalidad. Si la convención se sigue de forma fiable, los lectores sabrán que las definiciones de función aparecen después de su uso

**Afinidad Conceptual**

Los conceptos de código que están estrechamente relacionados deben estar cerca unos de otros. Si tienen una fuerte relación, como cuando una función llama a otra o usa una variable, deben estar menos separados verticalmente. La afinidad también puede surgir si un grupo de funciones realiza operaciones similares, lo que justifica mantenerlas juntas para mayor claridad.

**Orden Vertical**

En general, las funciones que son llamadas por otras funciones deben estar situadas debajo de las que las invocan. Esto ayuda a que el código se lea de manera más fluida, de lo general a lo específico. Así como en un periódico se presentan primero los conceptos más importantes y luego los detalles, en el código se deberían colocar las funciones principales antes que las que proporcionan detalles adicionales. De esta forma, es más fácil entender el propósito general del código sin tener que profundizar en los detalles.

**Formato Horizontal**

Esto sugiere que debemos intentar reducir las líneas de código

**Apertura y Densidad Horizontal**

Se recomienda usar el espacio en blanco horizontal para asociar elementos directamente relacionados y separar otros con una relación menos estrecha. Las instrucciones de asignación tienen dos elementos principales: el lado izquierdo y el derecho. Los espacios acentúan esta separación. Por otra parte, no hemos incluido espacios entre los nombres de las funciones y el paréntesis de apertura, ya que la función y sus argumentos están estrechamente relacionados. Su separación los desconectaría. Separo los argumentos en los paréntesis de invocación de la función para acentuar la coma e indicar que los argumentos son independientes.

Los factores, que tienen una mayor precedencia en las operaciones, no deben tener espacios en blanco alrededor, mientras que los términos, como en la suma y la resta que tienen una precedencia menor, se separan con espacios en blanco. Sin embargo, muchas herramientas de formateo de código no respetan estas diferencias de precedencia y aplican un espaciado uniforme, lo que puede hacer que estas separaciones importantes se pierdan al ajustar el formato del código.

**Sangrado**

Un archivo de código tiene una estructura jerárquica que organiza la información en diferentes niveles, desde el archivo completo hasta sus partes más detalladas, como las clases, los métodos, y los bloques de código dentro de los métodos. Cada uno de estos niveles representa un ámbito donde se declaran variables, funciones y otras estructuras, y donde se interpretan y ejecutan las instrucciones del programa.

Para que esta jerarquía sea visualmente comprensible, usamos sangrías para mostrar la relación entre los distintos niveles. Por ejemplo, las declaraciones de clases se colocan sin sangría, lo que indica que están en el nivel más alto. Los métodos dentro de una clase se sangran un nivel hacia la derecha, lo que indica que están dentro de esa clase. A su vez, el código dentro de los métodos, como las implementaciones de las funciones o bloques condicionales, se sangra aún más, mostrando que está contenido dentro del método correspondiente.

Este sistema de sangrado es crucial para los programadores porque les permite ver de un vistazo cómo está estructurado el código. La alineación visual ayuda a identificar rápidamente las relaciones entre diferentes partes del código, como los métodos dentro de una clase o los bloques de código dentro de una función. Sin esta estructura de sangrado, sería mucho más difícil seguir el flujo y la organización del código, haciendo que el archivo fuera casi ilegible.

**Reglas De Equipo**

Un equipo de programadores debe acordar un único estilo de formato y todos los integrantes del equipo deben aplicarlo. El objetivo es que el software tenga un estilo coherente. un buen sistema de software se compone de una serie de documentos que se leen fácilmente. Deben tener un estilo coherente y dinámico. El lector debe confiar en que los formatos que ve en nuestro archivo de código significarán lo mismo para otros. Lo último que queremos es aumentar la complejidad del código creando una mezcla de estilos diferentes.