Mantenga su código coherente

por Grzegorz Ziemoński MVB · Feb. 02, 17 · Zona Java

Navegue por el laberinto de la experiencia del usuario final y recoja esta guía esencial de APM, presentada en asociación con CA Technologies .

Hay un montón de cosas en la programación que tienen aparentemente poco o ningún significado. Algunas de esas cosas, a pesar de su "aparentemente" poco significado causa una gran cantidad de discusiones sin sentido, por ejemplo, *pestañas vs espacios* (siempre y cuando se utilizan 4 espacios) o *llaves vs no llaves* (siempre y cuando se utilizan llaves). Yo diría que no importa cuál usted elija, pero hay una cosa importante a recordar. Una vez que hayas tomado una decisión, **¡tienes que ser consistente!**

¿Por qué eso importa?

Cuando pensé por primera vez en escribir este artículo, pensé en poner un argumento como: *Mira. Pasas mucho más tiempo leyendo código que escribiéndolo. Así que cuando hablamos de consistencia, estamos más preocupados con la lectura. Imagine que está leyendo un libro que tiene diferentes fuentes en cada página, utiliza diferentes espaciados y puntuación o incluso el estilo de escritura cambia cada otro párrafo. ¿Leería bien?* Ahora, tal argumento sería cojo, como la analogía no necesariamente tiene que celebrar, pero parece que lo hice de todos modos;).

Después de haberle debilitado con un tiro de mi cañón lame, ahora puedo tratar de convencer a mantener consistente con otros argumentos:

- **La fusión de cambios es más fácil** : Cualquiera que haya intentado resolver un conflicto SCM con un tipo que usa un formateador diferente está sonriendo ahora mismo. Algunos problemas de consistencia te hacen perder aún más tiempo en el agradable proceso de combinar cambios.
- **El código es más predecible** : si ves una construcción en el código, hay una mayor probabilidad de que sepas qué es, por qué está ahí, dónde buscar más, y así sucesivamente, ya que podría haber similares en la base de código ya.
- **Algunos errores pueden ser más fáciles de detectar o evitar** : El ejemplo más simple es estar siempre usando llaves o siempre haciendo el cuerpo de condicionales y bucles de una línea.
- **Las personas extrañas como yo son más felices** : algunos de nosotros sólo necesitamos que las cosas estén en perfecta simetría, bien ordenadas, etc. De lo contrario, nuestro cerebro levanta una alerta y nos hacemos menos efectivos.

¿Qué mantener constante?

Espacios en blanco

Los espacios en blanco le ayudan a organizar bien su código. Ellos le permiten mantener las cosas cerca que deben permanecer cerca y separar las cosas que merecen ser separados. Convenciones con respecto a los espacios en blanco puede ahorrar un montón de tiempo cuando la fusión, por lo que es bueno comenzar con éste.

Cuando se trata de ejemplos, como ya se mencionó, una de las opciones sería *tabs vs spaces* . Si los espacios, *¿cuántos espacios? ¿Cuántas pestañas o espacios hay al romper una línea? ¿Dónde pones nuevas líneas y cuántas?* ¿Deberías ponerlos entre campos? ¿Campos anotados? ¿Métodos? Antes y después de bucles? Partes de sus exámenes? Podría seguir y seguir, pero lo más probable es obtener el punto ya.

Orden de Miembros

Ordenar es otro punto muy importante cuando la fusión como SCM se pierden cuando se mezclan las cosas en un archivo de origen. También afecta a la previsibilidad, ya que desea saber si debe buscar o no el archivo de origen de la cosa que desea encontrar.

Para este punto, hay dos opciones particularmente importantes:

1. *¿Qué orden tienen los diferentes constructos de lenguaje en una clase?* Lo más probable es que elija algo como: constantes, campos, constructores, métodos, clases internas.
2. *¿Qué orden deben ir los métodos en una clase?* Aquí yo esperaría algo como la regla del step-

down.

Tirantes

Aparatos no son un tema candente en Java cuando se trata de su colocación en una línea, pero seguramente lo son cuando se trata de una línea condicional y bucles. Algunos argumentan que el uso de llaves siempre te hace más seguro porque nunca te olvidarás de añadirlas a una declaración existente al extenderlo. Otros sostienen que esas declaraciones deben contener una línea de código de todos modos y por lo tanto los apoyos no son necesarios. En otros idiomas, también se discute la colocación de llaves en una línea, porque algunos afirman que una>

Habiendo dicho todo lo anterior, decida con su equipo: *¿Debe usar frenos con condicionales y bucles?* Y, si es necesario: *¿Dónde pones las llaves en una línea?*

Nombre

Nombrar es muy importante en el desarrollo de software. Supongo que nadie cuestiona eso. Mientras que, en general, nos esforzamos por "buenos nombres", hay algunas cosas que no pueden ser evaluadas como mejores o peores. Sólo tenemos que estar de acuerdo con ellos.

El primer y muy importante ejemplo que me viene a la mente es: *¿Cómo debería nombrar sus exámenes?* Dependiendo del lenguaje y el marco utilizado, puede haber un montón de opciones diferentes. En general, desea especificar la funcionalidad que se está probando, el estado de sistema o objeto asumido y el comportamiento esperado. Otra cosa a decidir sería *cuando para capitalizar una letra en un nombre?* Es común ver un proyecto en el que algunas abreviaturas se mueven a los nombres tal cual, mientras que otros sólo tienen la primera letra en mayúscula. Y tal vez una inconsistencia más que me encontré recientemente: *¿Dónde poner prefijos como "Prueba" en un nombre?*

Estructura del paquete

Los diferentes módulos / componentes de su aplicación podrían tener bloques de construcción similares en el nivel inferior, por ejemplo entidades, configuraciones de contexto, repositorios, etc. Mantener la estructura interna del paquete coherente en la aplicación podría hacer que el proyecto sea más navegable y, por tanto, más comprensible al costo De tener algunos paquetes con sólo una o dos clases dentro.

Soluciones a problemas similares

Este punto se refiere menos a las convenciones y más a la codificación del sentido. En general, queremos utilizar soluciones similares a problemas similares, especialmente dentro de un único archivo de origen. Por ejemplo, si alguien está usando una `Stream` para filtrar una colección, no escribe un `for` bucle con una `if` declaración dentro de unas pocas líneas a menos que tenga una buena razón para hacerlo. Si todo el proyecto utiliza una clase de biblioteca para construir URL, no hace `String` escapes y concatenación sólo porque sabe cómo hacerlo. Si el proyecto utiliza las afirmaciones de JUnit para las pruebas de unidad, no añade AssertJ a los proyectos simplemente porque lo prefiera. Estos son sólo algunos ejemplos que me molestó en el pasado, pero usted consigue el punto.

¡Tu turno!

La lista de arriba es seguramente incompleta - estas son sólo las cosas que se me ocurrió a través de mi mente en el momento de escribir este artículo. Si tiene alguna idea para que otras cosas se mantengan consistentes, por favor, póngalas en los comentarios para que todos podamos beneficiarnos!

Temas: CÓDIGO LIMPIO, CONVENCIONES DE CÓDIGO, JAVA, COHERENCIA

Publicado en DZone con el permiso de Grzegorz Ziemoński , DZone MVB . Vea el artículo original aquí.