Buenas prácticas de Java

Clean code: ¿qué es?

Son una serie de principios que ayudan a producir código intuitivo y fácil de modificar. 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.**

Las ventajas de este tipo de programación son obvias: el clean code se vuelve **independiente del desarrollador que lo ha creado**. En principio, cualquier programador puede trabajar con él, lo que evita problemas como los que conlleva el código heredado. El **mantenimiento del software también se simplifica,** porque los bugs son más fáciles de buscar y corregir.

Sus principios solucionan con eficacia uno de los principales problemas que gran parte de los proyectos de sistemas enfrentan: la manutención.

K.I.S.S.

1. K.I.S.S (keep it simple, stupid)

El código debe ser lo mas sencillo posible, evitando cualquier complejidad innecesaria. Por lo tanto, es importante preguntarse si hay una solución mas simple de resolver un problema en particular.

Cosas para tener en cuenta

- Separar cada clase por responsabilidad.
- Agrupamos cada clase en su paquete.
- No sobre cargar funciones.
- Evitamos la sobre información o código basura.

Nombramiento

2. Nombramiento (los nombres importan)

Nos debe facilitar la lectura a nosotros y a otras personas que puedan acceder a nuestro código.

Cosas para tener en cuenta

- Usamos Camel Case.
- Es descriptivo.
- Entregamos la idea central de su función (aplica a funciones, métodos o clases).
- No nos importa el tamaño

Criterios del Nombramiento

- **Proyectos:**

  En minúsculas, separando el nombre con guion bajo o medio (- _)

- **Paquetes:**

  En minúsculas, separando el nombre con punto cuando lo requiera (.)
- **Clases:**

  Primer letra en mayúsculas, aplicar camelCase desde la segunda palabra en adelante.
- **Clases de Servicio:**

  Como una clase concatenado con el apellido “implements o service”.
- **Métodos:**

  En camelCase, descriptivos, generalmente con el verbo que describe lo que hace.
- **Variables:** En camelCase, descriptivas y fáciles de leer.

Funciones Simples

3. Funciones simples (simples, claras y pequeñas)

Basada en el concepto de atomicidad en el código. Siempre es preferible mayor cantidad de partes pequeñas que realicen pequeñas tareas.

Cosas para tener en cuenta

- Encargadas solo de una responsabilidad / función.
- Encapsuladas dentro de una variable a excepción de void. Es decir, aquellas funciones que retornan valores, debo almacenar la información en una variable para luego poder operar con ella de ser necesario.
- Contenidas dentro de otra función en algunas ocasiones.

D.R.Y. vs W.E.T.

4. D.R.Y Vs W.E.T (Don´t repeat yourself vs We enjoy typing)

Hay que evitar ser repetitivo. Cada función debe tener una r**epresentación única y, por lo tanto, inequívoca** dentro del sistema general del clean code. No deberían existir 2 partes de código que sean repetitivas o duplicadas de forma innecesaria. Por protocolo, no debería existir mas de un 10% de código duplicado en toda la aplicación.

Cosas para tener en cuenta

- Utilizar constantes.
- Revisar si existen métodos que pueda reutilizar.
- Abstracción y uso de patrones de diseño.

Comentarios y Documentación

5. Comentarios y documentación

Solo realizar comentarios de ser necesario. Generalmente, **los códigos son modificados, los comentarios no**. Estos son olvidados, y, por lo tanto, no retratan la funcionalidad real de los códigos. Es sumamente importante que, si comentamos o documentamos código, sea solamente lo necesario y que sea revisado en conjunto con la versión del código que lo acompaña.

Manejo de Errores

6. Manejo de errores

Debemos tratar como desarrolladores los errores de forma correcta. las cosas pueden salir mal; pero cuando esto ocurre, los programadores son los responsables por garantizar que el código continúe realizando lo que necesita.

- Cosas para tener en cuenta
- Usar excepciones en lugar de códigos de retorno.

- No devolver nulo.
- No pasar valores nulos, prohibiendo la introducción de valores nulos.

Test limpios

7. Test limpios

Realizar tests en el área de programación, es una etapa muy importante. Un código solo se considera limpio, después de ser válido a través de pruebas, que también deben ser limpias. Por esta razón, estos deben seguir algunas reglas, como:

- **Fast:**

  El test debe ser rápido, permitiendo que sea realizado muchísimas veces y en cualquier momento;
- **Independent:**

  Debe ser independiente, con el fin de evitar que cause efecto cascada cuando ocurra alguna falla, lo que dificulta el análisis de los problemas.
- **Repeatable:**

  Debe permitir la repetición del test, muchísimas veces y en diferentes ambientes;
- **Self-Validation:**

  Los tests bien escritos retornan con las respuestas true o false para que el error no sea subjetivo;
- **Timely:**

  Los tests deben seguir estrictamente el criterio de puntualidad. Además de esto, lo ideal es que sean escritos antes del propio código, pues evita que sea muy complejo para realizar el test

Análisis

Prueba analizar el siguiente código con tu equipo y realizarle todos los cambios necesarios para que cumpla con las reglas de código limpio

```
public class Ejemplo { // Método public static void m(int a, int b) { int c = a + b; System.out.println("El resultado es: " + c); } // Método public static void metodoLargo(int x, int y, int z) { for (int i = 0; i < x; i++) { if (i % 2 == 0) { System.out.println("El número " + i + " es par."); } else { System.out.println("El número " + i + " es impar."); } } int resultado = y * z; if (resultado > 100) { System.out.println("El resultado es mayor a 100."); } else { System.out.println("El resultado es menor o igual a 100."); } } public static void main(String[] args) { m(3, 4); metodoLargo(10, 5, 2); } }
```