EDVIN ALESSANDRO SOLIS MELGAR

BUENAS MALAS PRACTICAS EN LA PROGRAMACIÓN ORIENTADA A OBJETOS

MALAS PRACTICAS DE LA PROGRAMACIÓN EN LA PROGRAMACIÓN ORIENTADA A OBJETOS

1.

- Mal uso de herencia: La herencia puede ser una herramienta útil en la programación orientada a objetos, pero su mal uso puede llevar a una jerarquía de clases compleja e inflexible. Por ejemplo, una jerarquía de clases muy profunda o una sobreutilización de clases base pueden hacer que el código sea difícil de mantener.

2.

- Acoplamiento fuerte: El acoplamiento es la medida de cómo de cerca relacionados están dos componentes de un sistema. Un acoplamiento fuerte entre componentes significa que cambios en un componente pueden tener un gran impacto en otro. Un acoplamiento fuerte puede hacer que el código sea difícil de entender y de mantener.

3.

- Código duplicado: La duplicación de código puede hacer que el código sea difícil de mantener. La programación orientada a objetos ofrece herramientas como la herencia y la composición para reutilizar código, por lo que la duplicación de código debería evitarse.

4.

- Abuso de patrones de diseño: Los patrones de diseño pueden ser útiles para resolver problemas comunes de programación, pero su abuso puede llevar a código complejo y difícil de entender. Es importante usar patrones de diseño de manera juiciosa y solo cuando sean necesarios.

5.

- Falta de encapsulamiento: El encapsulamiento es el principio de la programación orientada a objetos que establece que los datos y el comportamiento de un objeto deben estar contenidos dentro del objeto mismo. Una falta de encapsulamiento puede llevar a código difícil de entender y de mantener.

6.

- Clases demasiado grandes: Las clases grandes pueden ser difíciles de entender y mantener. Es mejor dividir una clase grande en clases más pequeñas y cohesivas.

7.

- Código no reutilizable: La reutilización de código es uno de los principales beneficios de la programación orientada a objetos. No reutilizar el código puede llevar a código duplicado y difícil de mantener.

8.

- Sobreingeniería: La sobreingeniería se produce cuando se agrega complejidad innecesaria al código. Esto puede hacer que el código sea difícil de entender y de mantener. Es importante recordar que la simplicidad es una virtud en la programación orientada a objetos.

BUENAS PRACTICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

1.

- Diseño orientado a objetos: El diseño orientado a objetos es un proceso que implica pensar en objetos y sus relaciones antes de comenzar a escribir código. Este proceso ayuda a crear un diseño más limpio y cohesivo.

2.

- Principios SOLID: Los principios SOLID son un conjunto de principios de diseño orientado a objetos que ayudan a crear código modular y mantenible. Estos principios incluyen la responsabilidad única, abierto/cerrado, sustitución de Liskov, segregación de la interfaz y inversión de dependencia.

3.

- Patrones de diseño: Los patrones de diseño son soluciones comunes a problemas de programación que se han demostrado útiles en la práctica. El uso juicioso de patrones de diseño puede ayudar a crear código limpio y fácil de entender.

4.

- Acoplamiento débil: El acoplamiento débil es la medida de cómo de independientes están dos componentes de un sistema. Un acoplamiento débil puede hacer que el código sea más fácil de entender y mantener.

5.

- Encapsulamiento: El encapsulamiento es el principio de la programación orientada a objetos que establece que los datos y el comportamiento de un objeto deben estar contenidos dentro del objeto mismo. El uso adecuado del encapsulamiento puede ayudar a crear código limpio y fácil de entender.

6.

- Clases pequeñas y cohesivas: Las clases pequeñas y cohesivas son más fáciles de entender y mantener que las clases grandes y complejas. Es importante dividir una clase grande en clases más pequeñas y cohesivas.

7.

- Reutilización de código: La reutilización de código es uno de los principales beneficios de la programación orientada a objetos. El uso juicioso de la herencia y la composición puede ayudar a crear código más limpio y fácil de entender.

8.

- Pruebas unitarias: Las pruebas unitarias son pruebas automatizadas que se ejecutan en el código para garantizar que funcione según lo previsto. Las pruebas unitarias pueden ayudar a detectar problemas en el código temprano y facilitar su corrección.