PATRONES DE DISEÑO DEL SOFTWARE

¿QUÉ ES UN PATRÓN DE DISEÑO?

TIPOS DE PATRONES DE DISEÑO

VENTAJAS DE LOS PATRONES DE DISEÑO

¿QUÉ ES UN PATRÓN DE DISEÑO?

Un **patrón de diseño** es una plantilla general, reutilizable para problemas de diseño de software. Estas tienen como objetivo identificar los problemas que podamos tener en nuestro programa, así como solucionar problemas a los que otros desarrolladores se han enfrentado antes.

En el pasado, se necesitaba completar todo el software antes de hacer pruebas, esto suponía encontrarse luego con muchos problemas. Entonces para ahorrar tiempo y no tener que volver a la parte de desarrollo después de finalizar, se introdujeron las **fases de desarrollo**, estas nos permiten identificar errores y problemas en el código que podrían no ser visibles de otra forma.

Además, los patrones de diseño te ayudarán a estar más seguro de que tu código está más libre de errores, ya que son soluciones que funcionan, ya que han sido probadas por otros desarrolladores con anterioridad.

TIPOS DE PATRONES DE DISEÑO

Los patrones de diseño más utilizados se pueden clasificar en tres categorías principalmente, que són:

Patrones creacionales

Estos proporcionan diversos mecanismos de creación de objetos, que trabajan en la reutilización del código existente de manera que se adaptan adecuadamente a la situación.

Algunos de los patrones que encontramos en esta categoría son:

- ★ Abstract Factory
- ★ **Builder Patterns**
- ★ **Factory Method**
- ★ Prototype
- ★ **Singleton**

Entre ellos destacaremos los tres más utilizados, los “builder patterns”, como su nombre indica, están hechos para construir objetos. El “factory method” proporciona una interfaz para crear objetos en una superclase, pero permite que las subclases alteren el tipo de los objetos que se van a crear. Y por último el patrón de diseño “Singleton”, que restringe la creación de instancias de una clase a un único objeto.

Patrones estructurales

Este tipo de patrones facilitan soluciones y estándares eficientes para las composiciones de las clases y las estructuras de los objetos. En estos, el concepto de herencia se utiliza para crear interfaces y definir formas de hacer objetos para conseguir nuevas funcionalidades.

Algunos de los patrones que encontramos en esta categoría son:

- ★ **Adapter**
- ★ Bridge
- ★ Composite
- ★ Decorator
- ★ Facade
- ★ Flyweight
- ★ Proxy

Entre todos el más usado sería el patrón de diseño “Adapter” que permite que dos clases incompatibles trabajen juntas al convertir la interfaz de una en la otra, esto nos permitirá utilizar las funcionalidades de ambas, lo cual no habría sido posible de otra forma.

Patrones de comportamiento

Esta clase de patrones se ocupan de la comunicación entre objetos de una clase, también pueden detectar otros patrones de comunicación ya presentes y manipularlos.

Algunos de los patrones que encontramos en esta categoría son:

- ★ Chain of responsibility
- ★ Interpreter
- ★ Iterator
- ★ Mediator
- ★ Memento
- ★ **Observer**
- ★ **State**
- ★ Template Method
- ★ Visitor

Entre ellos destacaremos dos, el patrón “Observer” es una dependencia de uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, se notifica a todos sus dependientes, de normal lo hará llamando a uno de sus métodos. Y el patrón “State” que encapsula los estados en los que puede estar una máquina y permite que un objeto modifique su comportamiento cuando cambia su estado interno.

VENTAJAS DE LOS PATRONES DE DISEÑO

Para terminar, los patrones de diseños tienen diversas funciones pero cuales son realmente algunas de las **ventajas** que ofrecen los patrones de diseño son:

- Es un extenso registro de problemas y soluciones.
- Crean un estándar para la resolución de problemas.
- Condensan y simplifican el aprendizaje de buenas prácticas.
- Proporcionan un lenguaje común a todos los desarrolladores.
- Evitan tener que reinventar cosas que ya han sido creadas.