Patrones de diseño

(Hecho por Jonatan Bas Hidalgo)

Índice:

1)Introducción.

Los patrones de diseño o *design patterns*, son una solución general, reutilizable y aplicable a diferentes problemas de diseño de software. Se trata de plantillas que identifican problemas en el sistema y proporcionan soluciones apropiadas a problemas generales a los que se han enfrentado los desarrolladores durante un largo periodo de tiempo, a través de prueba y error.

Los patrones de diseño se han convertido en un objeto de cierta controversia en el mundo de la programación en los últimos tiempos, en gran parte debido a su "uso excesivo" percibido, lo que lleva a un código que puede ser más difícil de entender y administrar.

2)Modelo.

En 1994, cuatro autores Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, a los que llamaron *Gang of Four (GoF)*, publicaron un libro titulado *Design Patterns*, elementos de software orientado a objetos reutilizables. Con este trabajo se inició el concepto de patrón de diseño en el desarrollo de software y recoge 23 patrones de diseño comunes. Cada uno de ellos define la solución para resolver un determinado problema, facilitando además la reutilización del código fuente.

Existen muchísimos patrones de diseño de software e irán apareciendo cada vez más. En este post hablaremos de los más conocidos o los llamados patrones clásicos.

3)¿Por qué usar patrones de diseño?.

El gran crecimiento del sector de las tecnologías de la información ha hecho que las prácticas de desarrollo de software evolucionen. Antes se requería completar todo el software antes de realizar pruebas, lo que suponía encontrarse con problemas. Para

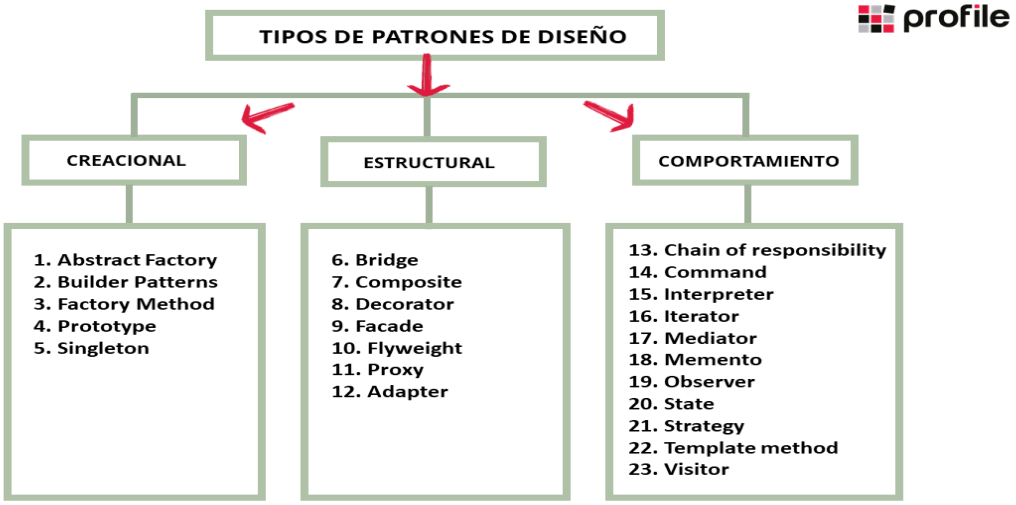
ahorrar tiempo y evitar volver a la etapa de desarrollo una vez que este ha finalizado, se introdujo una práctica de prueba durante la fase de desarrollo.

Esta práctica se usa para identificar condiciones de error y problemas en el código que pueden no ser evidentes en ese momento. En definitiva, los patrones de diseño te ayudan a estar seguro de la validez de tu código, ya que son soluciones que funcionan y han sido probados por muchísimos desarrolladores siendo menos propensos a errores.

4)Tipos de patrones de diseño.

Los patrones de diseño más utilizados se clasifican en tres categorías principales, cada patrón de diseño individual conforma un total de 23 patrones de diseño. Las cuatro categorías principales son:

- Patrones creacionales
- Patrones estructurales
- Patrones de comportamiento



5)Principales patrones de diseño y su explicación.

5.1. Factory method.

Una fábrica normal produce bienes; una fábrica de software produce objetos. Y no solo eso, lo hace sin especificar la clase exacta del objeto que se creará. Para lograr esto, los objetos se crean llamando a un método de fábrica en lugar de llamar a un constructor.

5.2.Singleton.

El patrón singleton se utiliza para limitar la creación de una clase a un solo objeto. Esto es beneficioso cuando se necesita un objeto (y solo uno) para coordinar acciones en todo el sistema. Hay varios ejemplos en dónde debería existir una única instancia de una clase, incluidos cachés, grupos de subprocesos y registros.

Es trivial iniciar un objeto de una clase, pero ¿cómo nos aseguramos de que solo se cree un objeto? La respuesta es hacer que el constructor sea "privado" para la clase que pretendemos definir como singleton. De esa forma, solo los miembros de la clase pueden acceder al constructor privado y nadie más.

5.3.Observer.

Este patrón es una dependencia de uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, se notifica a todos sus dependientes. Normalmente, esto se hace llamando a uno de sus métodos.

En aras de la simplicidad, piense en lo que sucede cuando se sigue a alguien en Twitter. Básicamente, le estás pidiendo a Twitter que te envíe (el observador) actualizaciones de tweets de la persona (el sujeto) que seguiste. El patrón consta de dos actores, el observador que está interesado en las actualizaciones y el sujeto que genera las actualizaciones.

5.4.Strategy.

El patrón de estrategia permite agrupar algoritmos relacionados bajo una abstracción, lo que permite cambiar un algoritmo o política por otro sin modificar el cliente. En lugar de implementar directamente un solo algoritmo, el código recibe instrucciones en tiempo de ejecución que especifican cuál del grupo de algoritmos ejecutar.

5.5.Adapter.

Esto permite que las clases incompatibles trabajen juntas al convertir la interfaz de una clase en otra. Piense en ello como una especie de traductor: cuando dos jefes de estado que no hablan un idioma común se encuentran, generalmente un intérprete se sienta entre los dos y traduce la conversación, lo que permite la comunicación.

Si tienes dos aplicaciones, una de las cuales arroja la salida como XML y la otra requiere una entrada JSON, entonces necesitará un adaptador entre las dos para que funcionen sin problemas.

5.6.Builder.

Como su nombre lo indica, se utiliza un patrón de construcción para construir objetos. A veces, los objetos que creamos pueden ser complejos, estar formados por varios subobjetos o requerir un elaborado proceso de construcción. El ejercicio de crear tipos complejos se puede simplificar utilizando el patrón de construcción. Un objeto compuesto o agregado es lo que generalmente construye un constructor.

Consideración clave: el patrón Builder puede parecer similar al patrón de "fábrica abstracta", pero una diferencia es que el patrón Builder crea un objeto paso a paso, mientras que el patrón de fábrica abstracto devuelve el objeto de una vez.

5.7.State.

El patrón State encapsula los diversos estados en los que puede estar una máquina y permite que un objeto altere su comportamiento cuando cambia su estado interno. La máquina o el contexto, como se le llama en lenguaje de patrones, puede tener acciones que lo impulsen a diferentes estados. Sin el uso del patrón, el código se vuelve inflexible y está plagado de condicionales if-else.

6) Ventajas.

1-Es una plataforma conocida.

Proporcionan una estructura conocida por todos los programadores, de manera que la forma de trabajar no resulte distinta entre los mismos. Así la incorporación de un nuevo programador, no requerirá conocimiento de lo realizado anteriormente por otro. Permiten tener una estructura de código común a todos los proyectos que implemente una funcionalidad genérica. La utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software.

2-Mantenimiento.

El software construido es más fácil de comprender, mantener y extender. Existen versiones ya implementadas de esta funcionalidad común (frameworks) como Struts o Velocity y que nos permiten centrarnos en desarrollar sólo la funcionalidad específica requerida por cada aplicación. Además, da mejor imagen de profesionalidad y calidad.

3-Reutilización de código.

Contribuyen a reutilizar diseño gráfico, identificando aspectos claves de la estructura de un diseño que puede ser aplicado en una gran cantidad de situaciones. La importancia de la reutilización del diseño no es despreciable, ya que ésta nos provee de numerosas ventajas: reduce los esfuerzos de desarrollo y mantenimiento, mejora la Seguridad informática, eficiencia y consistencia de nuestros diseños, y nos proporciona un considerable ahorro en la inversión.

3-Adaptabilidad.

Mejoran la flexibilidad, modularidad y extensibilidad, factores internos e íntimamente relacionados con la calidad percibida por el usuario.

7) Bibliografía.

1. https://profile.es/blog/patrones-de-diseno-de-software/#%C2%BFPor_que_usar_patrones_de_diseno
2. https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o#Aplicaci%C3%B3n_en_%C3%A1mbitos_concretos
3. https://devexperto.com/patrones-de-diseno-software/
4. https://dev.to/gelopfalcon/los-7-patrones-de-diseno-de-software-mas-importantes-28l2
5. https://devexperto.com/patrones-de-diseno-software/