Patrones de Diseño

¿Qué es un patrón?

Es un modelo que sirve de muestra para sacar otra cosa igual.

A continuación, una breve descripción de cada patrón según su tipo.

Creacionales

Singleton:

Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia. Restringe la instanciación de una clase o valor de un tipo a un solo objeto. Se lo considera un patrón fácil, pero intrusivo porque ensucia la clase.

Multiton:

Garantiza que una clase solo tenga varias instancias conocidas, y proporciona un punto de acceso global a ellas. Se implementan igual al singleton, pero con un mapa (identificador, instancia) en vez de con un atributo. El método getInstancia recibe el nombre de la instancia.

Factory Method:

Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística, es decir, la diversidad de casos particulares que se pueden prever, para elegir el subtipo que crear. Parte del principio de que las subclases determinan la clase a implementar.

Abstract Factory:

Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. El problema a solucionar por este patrón es el de crear diferentes familias de objetos, como, por ejemplo, la creación de interfaces gráficas de distintos tipos (ventana, menú, botón, etc.).

Organización del trabajo:

Command:

Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de peticiones, y poder deshacer las operaciones. Desacopla el código que solicita un servicio del que lo presta.

Control de Acceso:

Proxy:

Proporciona un representante, o sustito de otro objeto para controlar el acceso a este, es decir, es una clase que funciona como una interfaz para otra cosa. Un proxy es un contenedor o un objeto de agente que el cliente está llamando para acceder al objeto de servicio real detrás de escena.

Facade (Fachada):

Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Viene motivado por la necesidad de estructurar un entorno de programación y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos.

Variación de servicios:

Strategy:

Determina cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea. El patrón estrategia permite mantener un conjunto de algoritmos de entre los cuales el objeto cliente puede elegir aquel que le conviene e intercambiarlo dinámicamente según sus necesidades.

Template:

Define el esqueleto de programa de un algoritmo en un método, llamado método de plantilla, el cual difiere algunos pasos a las subclases. Permite redefinir ciertos pasos seguros de un algoritmo sin cambiar la estructura del algoritmo.

State:

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto. Localiza el compor-

tamiento dependiendo del estado y divide dicho comportamiento en diferentes estados. Hace explicitas las transiciones entre los estados.

Extensión de servicios:

Decorator:

Asigna responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender la funcionalidad. Mas flexibilidad que la herencia estática. Evitar clases cargadas de funciones en la parte de arriba de la jerarquía.

Descomposición Estructural:

Composite:

Compone objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos. Defines jerarquías de clases formadas por objetos primitivos y compuestos. Facilita añadir nuevos tipos de componentes.