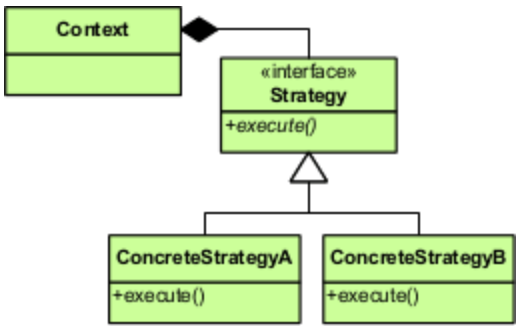
# PATRONES DE DISEÑO

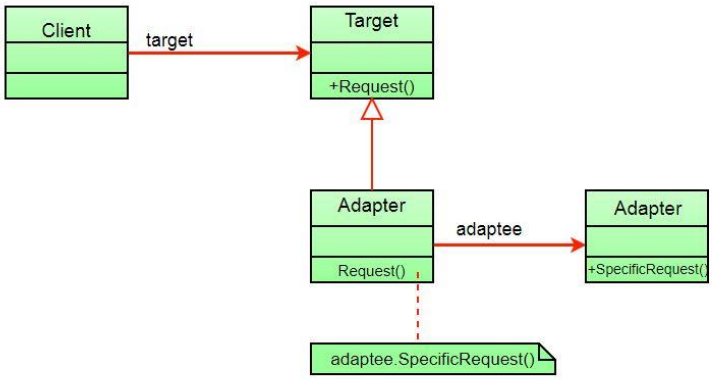
Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software.

Jethran Enrique
Gómez San Gabriel

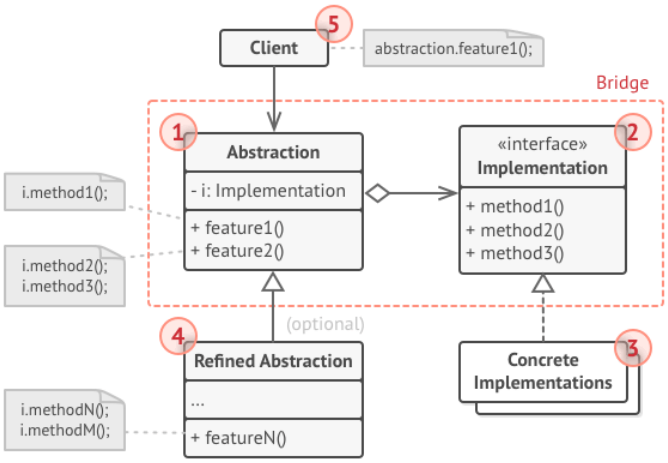
El patrón **strategy** se puede definir como un conjunto de algoritmos encapsulados y que son intercambiables. Este patrón nos permite variar el algoritmo de manera independiente al cliente que lo esté utilizado. Es muy útil cuando se quiere utilizar una familia de algoritmos en diferentes contextos. Por ejemplo, se tiene algunas clases en las que cada una se implementa un algoritmo diferente y se desea que todas utilicen una interfaz en común.



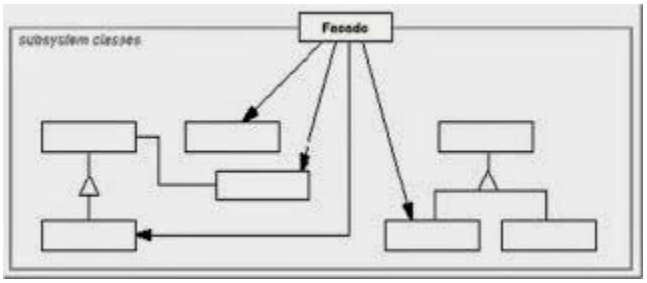

El patrón **adapter** convierte la interfaz de una clase en otra interfaz. Permite que las clases trabajen juntas y que no podrían hacerlo otra manera debido a interfaces incompatibles. Los adapter (adaptadores) se utilizan cuando tenemos una clase (cliente) que espera algún tipo de objeto y tenemos un objeto (Adaptee) que ofrece las mismas características pero que expone una interfaz diferente. El cliente solo ve la interfaz de destino y no el adaptador. El adaptador implementa la interfaz de destino.



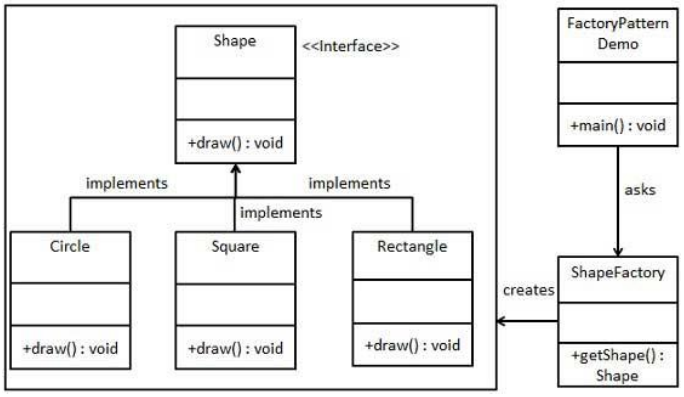

El patrón **Bridge** está diseñado para separar la interfaz de una clase de su implementación para que pueda variar o reemplazar la implementación sin cambiar el código del cliente. Patrón de diseño estructural que le permite dividir una clase gigante o un conjunto de clases estrechamente relacionas en dos jerarquías separada, abstracción e implementación, que se pueden desarrollar de manera independiente. Usa los términos de abstracción e implementación. Se ocupa para tener varias GUIs diferentes.



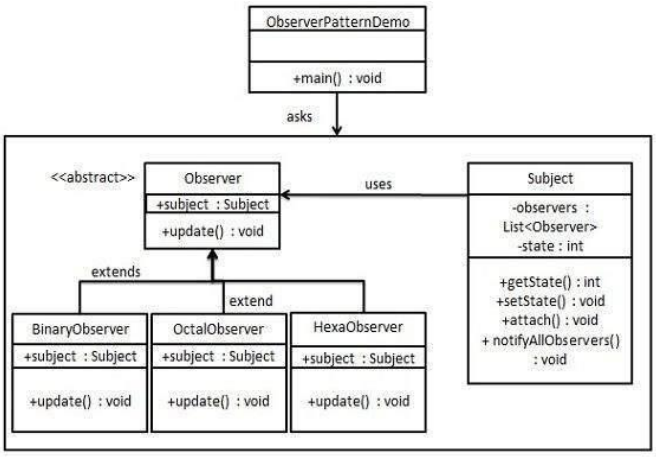

El patrón **facade** (fachada) permite simplificar la interfaz de comunicación entre dos objetos A y B de tal forma que para el objeto A sea más sencillo interactuar con el objeto B. Proporciona una interfaz unifica para un conjunto de interfaces de un sistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar. Proporciona una interfaz simplificada para un grupo de subsistemas o un sistema complejo.



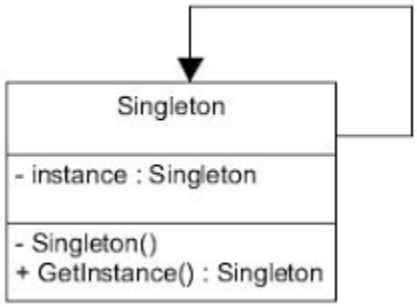

El patrón **factory method** sirve para crear un objeto sin exponer la lógica de creación al cliente y nos referimos a un objeto recién creado mediante una interfaz común. Define una interfaz para crear objetos, pero permite a las subclases decidir qué clase crear una instancia.



El patrón **observer** se utiliza cuando existe una relación de uno a muchos entre los objetos, por ejemplo, si un objeto se modifica, sus objetos dependientes se notificarán automáticamente. Cae bajo la categoría de patrón de comportamiento.




El patrón **singleton** garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a esta instancia. Se cuando existe una instancia de una clase y esta debe ser accesible a los clientes desde un punto de acceso conocido. La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de utilizar una instancia extendida sin modificar su código.




El patrón **template method** define un algoritmo en una clase base mediante operaciones abstractas que las subclases anulan para proporcionar un comportamiento correcto. Llama operaciones primitivas, así como operaciones definidas.