**Propósito**

**Proxy** es un patrón de diseño estructural que te permite proporcionar un sustituto o marcador de posición para otro objeto. Un proxy controla el acceso al objeto original, permitiéndote hacer algo antes o después de que la solicitud llegue al objeto original.

**Problema**

¿Por qué querrías controlar el acceso a un objeto? Imagina que tienes un objeto enorme que consume una gran cantidad de recursos del sistema. Lo necesitas de vez en cuando, pero no siempre.

Puedes llevar a cabo una implementación diferida, es decir, crear este objeto sólo cuando sea realmente necesario. Todos los clientes del objeto tendrán que ejecutar algún código de inicialización diferida. Lamentablemente, esto seguramente generará una gran cantidad de código duplicado.

En un mundo ideal, querríamos meter este código directamente dentro de la clase de nuestro objeto, pero eso no siempre es posible. Por ejemplo, la clase puede ser parte de una biblioteca cerrada de un tercero.

**Solución**

El patrón Proxy sugiere que crees una nueva clase proxy con la misma interfaz que un objeto de servicio original. Después actualizas tu aplicación para que pase el objeto proxy a todos los clientes del objeto original. Al recibir una solicitud de un cliente, el proxy crea un objeto de servicio real y le delega todo el trabajo.

Pero, ¿cuál es la ventaja? Si necesitas ejecutar algo antes o después de la lógica primaria de la clase, el proxy te permite hacerlo sin cambiar esa clase. Ya que el proxy implementa la misma interfaz que la clase original, puede pasarse a cualquier cliente que espere un objeto de servicio real.

**Aplicabilidad**

Hay decenas de formas de utilizar el patrón Proxy. Repasemos los usos más populares.

 Inicialización diferida (proxy virtual). Es cuando tienes un objeto de servicio muy pesado que utiliza muchos recursos del sistema al estar siempre funcionando, aunque solo lo necesites de vez en cuando.

 En lugar de crear el objeto cuando se lanza la aplicación, puedes retrasar la inicialización del objeto a un momento en que sea realmente necesario.

 Control de acceso (proxy de protección). Es cuando quieres que únicamente clientes específicos sean capaces de utilizar el objeto de servicio, por ejemplo, cuando tus objetos son partes fundamentales de un sistema operativo y los clientes son varias aplicaciones lanzadas (incluyendo maliciosas).

 El proxy puede pasar la solicitud al objeto de servicio tan sólo si las credenciales del cliente cumplen ciertos criterios.

 Ejecución local de un servicio remoto (proxy remoto). Es cuando el objeto de servicio se ubica en un servidor remoto.

 En este caso, el proxy pasa la solicitud del cliente por la red, gestionando todos los detalles desagradables de trabajar con la red.

 Solicitudes de registro (proxy de registro). Es cuando quieres mantener un historial de solicitudes al objeto de servicio.

 El proxy puede registrar cada solicitud antes de pasarla al servicio.

 Resultados de solicitudes en caché (proxy de caché). Es cuando necesitas guardar en caché resultados de solicitudes de clientes y gestionar el ciclo de vida de ese caché, especialmente si los resultados son muchos.

 El proxy puede implementar el caché para solicitudes recurrentes que siempre dan los mismos resultados. El proxy puede utilizar los parámetros de las solicitudes como claves de caché.

 Referencia inteligente. Es cuando debes ser capaz de desechar un objeto pesado una vez que no haya clientes que lo utilicen.

 El proxy puede rastrear los clientes que obtuvieron una referencia del objeto de servicio o sus resultados. De vez en cuando, el proxy puede recorrer los clientes y comprobar si siguen activos. Si la lista del cliente se vacía, el proxy puede desechar el objeto de servicio y liberar los recursos subyacentes del sistema.

El proxy también puede rastrear si el cliente ha modificado el objeto de servicio. Después, los objetos sin cambios pueden ser reutilizados por otros clientes.

**Pros**

* Puedes controlar el objeto de servicio sin que los clientes lo sepan.
* Puedes gestionar el ciclo de vida del objeto de servicio cuando a los clientes no les importa.
* El proxy funciona incluso si el objeto de servicio no está listo o no está disponible.
* *Principio de abierto/cerrado*. Puedes introducir nuevos proxies sin cambiar el servicio o los clientes.

## Contras

* El código puede complicarse ya que debes introducir gran cantidad de clases nuevas.
* La respuesta del servicio puede retrasarse.