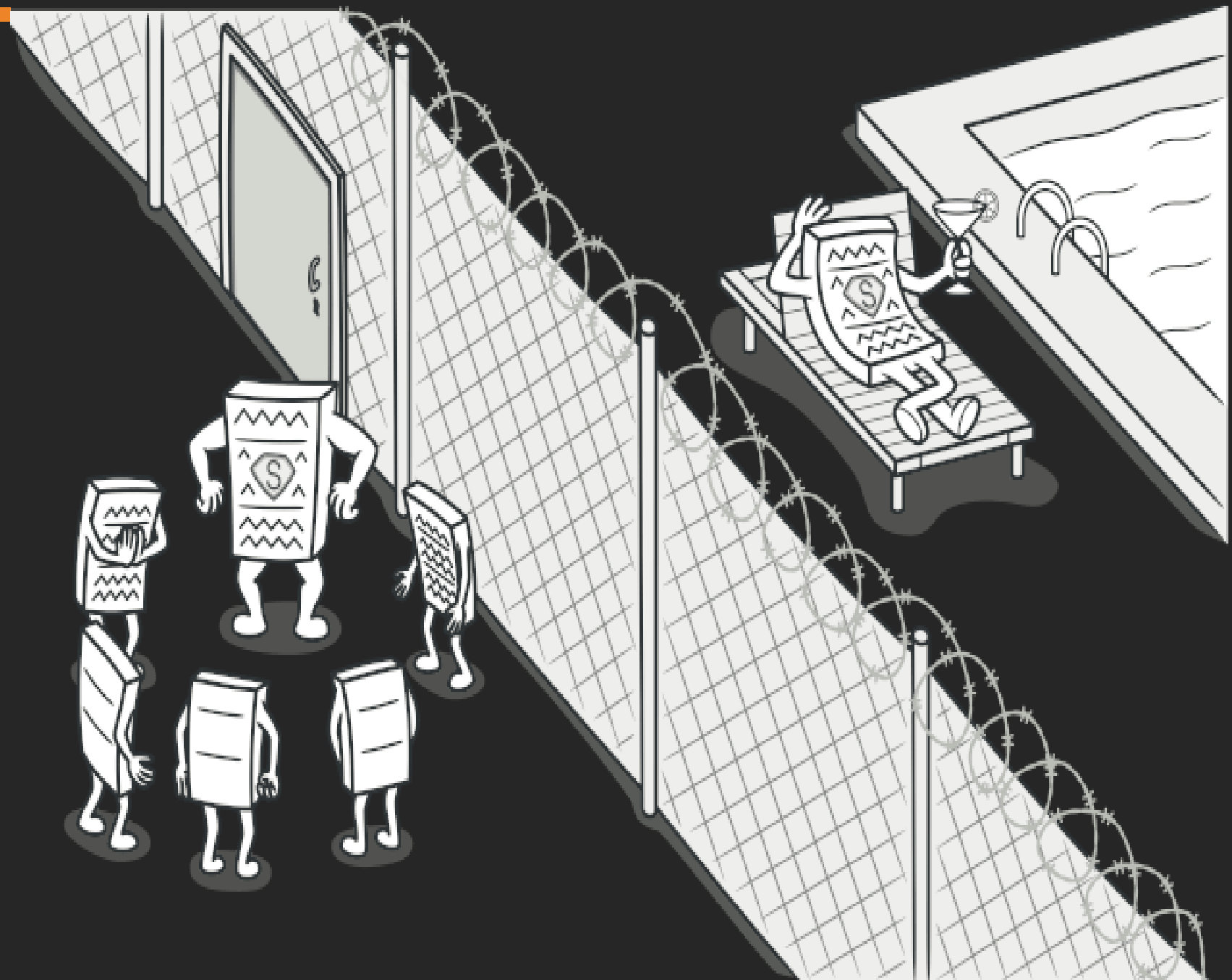
PROXY DESIGN PATTERN

Erick O. Escarcega Ramirez



Que es Proxy?

Se le denomina Proxy a una persona autorizada para actuar en nombre de otra

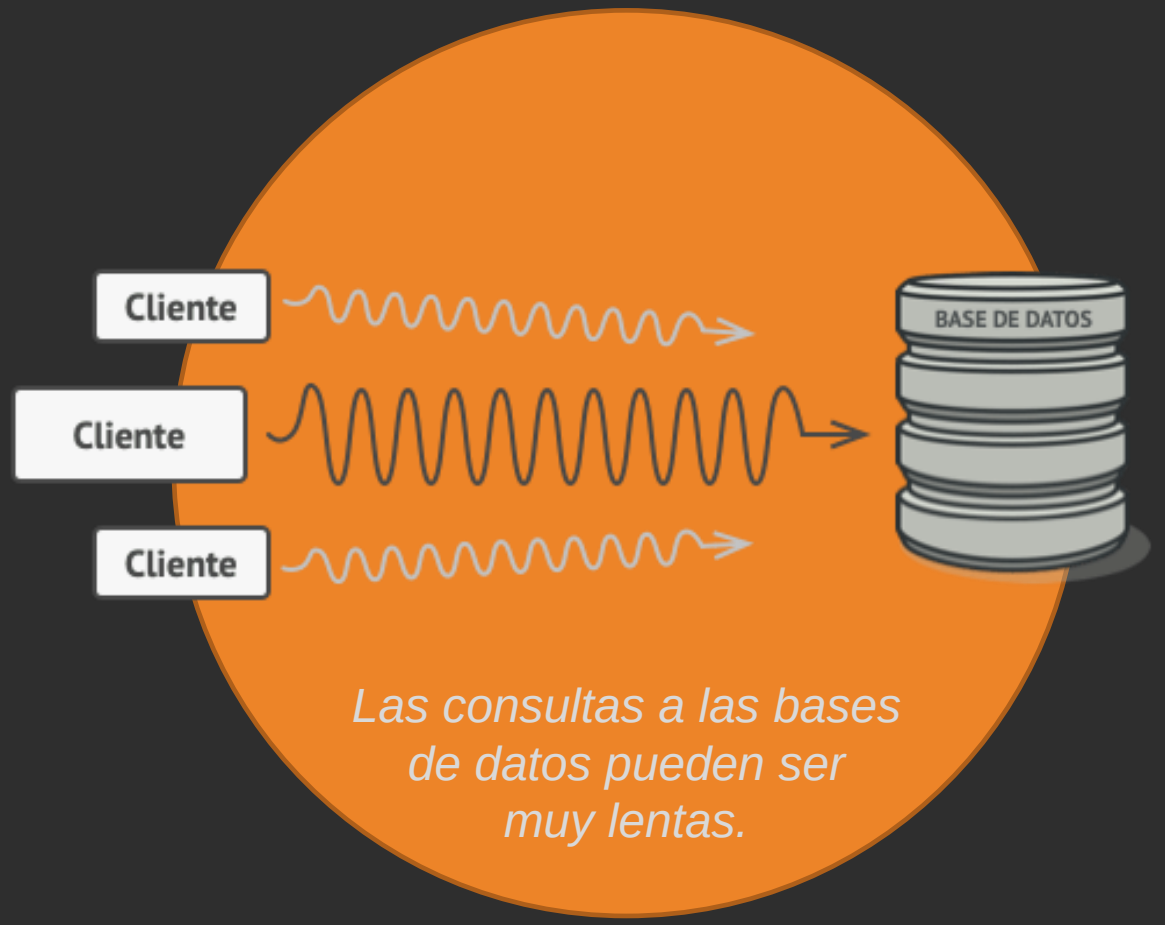
Representante - Sustituto

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.

PROBLEMÁTICA

Imagina que tienes un objeto enorme que consume una gran cantidad de recursos del sistema. Lo necesitas de vez en cuando, pero no siempre.

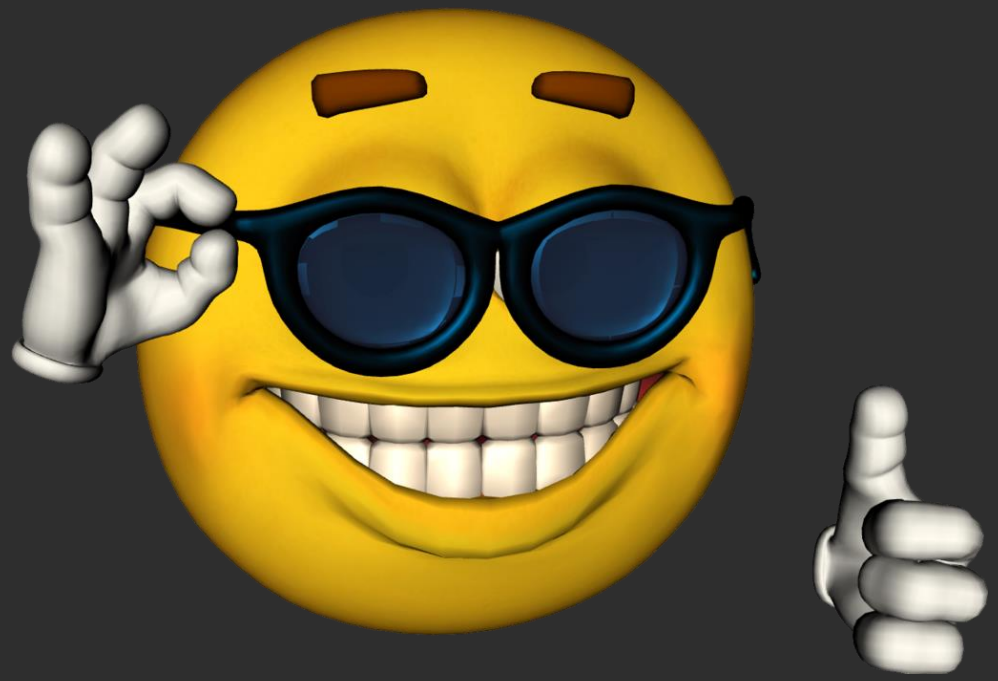


SOLUCIÓN

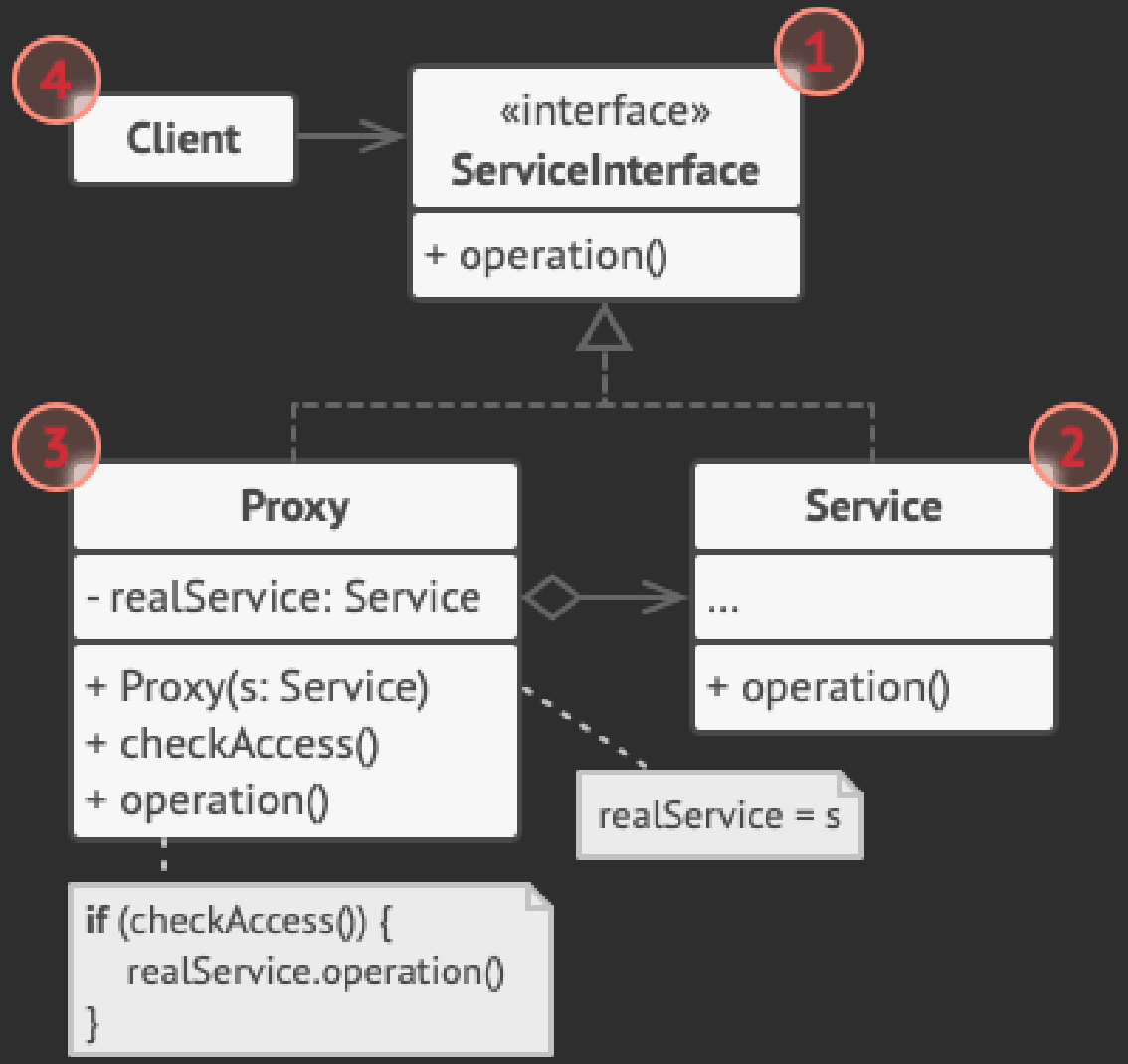
El patrón Proxy sugiere que crees una nueva clase proxy con la misma interfaz que un objeto de servicio original. Después actualizas tu aplicación para que pase el objeto proxy a todos los clientes del objeto original. Al recibir una solicitud de un cliente, el proxy crea un objeto de servicio real y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

El proxy se camufla como objeto de la base de datos. Puede gestionar la inicialización diferida y el caché de resultados sin que el cliente o el objeto real de la base de datos lo sepan.

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.



ESTRUCTURA

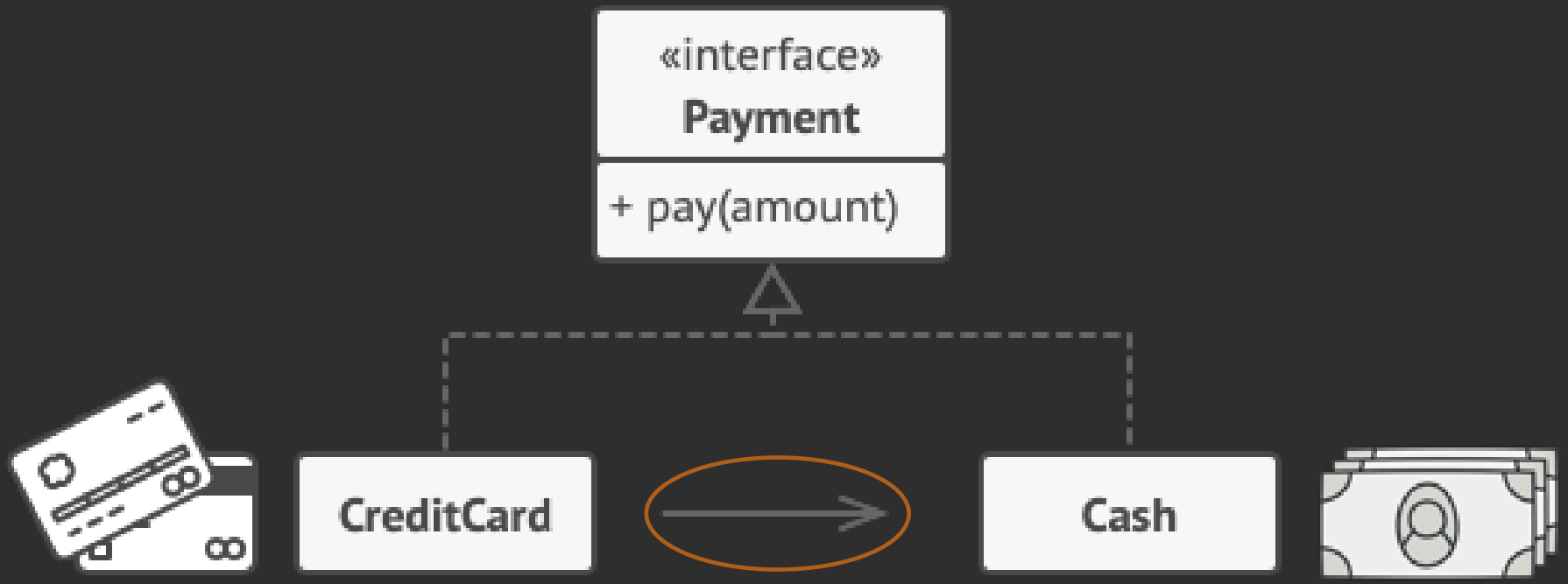


1. La **Interfaz de Servicio** declara la interfaz del Servicio. El proxy debe seguir esta interfaz para poder camuflarse como objeto de servicio.

2. Servicio es una clase que proporciona una lógica de negocio útil.

3. La clase **Proxy** tiene un campo de referencia que apunta a un objeto de servicio. Cuando el proxy finaliza su procesamiento (inicialización diferida, registro), pasa la solicitud al objeto de servicio.

4. El **Cliente** debe funcionar con servicios y proxies a través de la misma interfaz. De este modo puedes pasar un proxy a cualquier código que espere un objeto de servicio.


«interface»
Payment
+ pay(amount)
CreditCard
Cash

TIPOS DE PROXY

- 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.

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

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

- 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

PROS Y CONS

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

APLICABILIDAD

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

FIN.

REFERENCIAS

- https://refactoring.guru/design-patterns/proxy
- https://sourcemaking.com/design_patterns/proxy/python/1
- https://sodocumentation.net/es/python/topic/8056/patrones-de-diseno (code)
- Deign Patterns: Elements of Reusable object-oriented Software