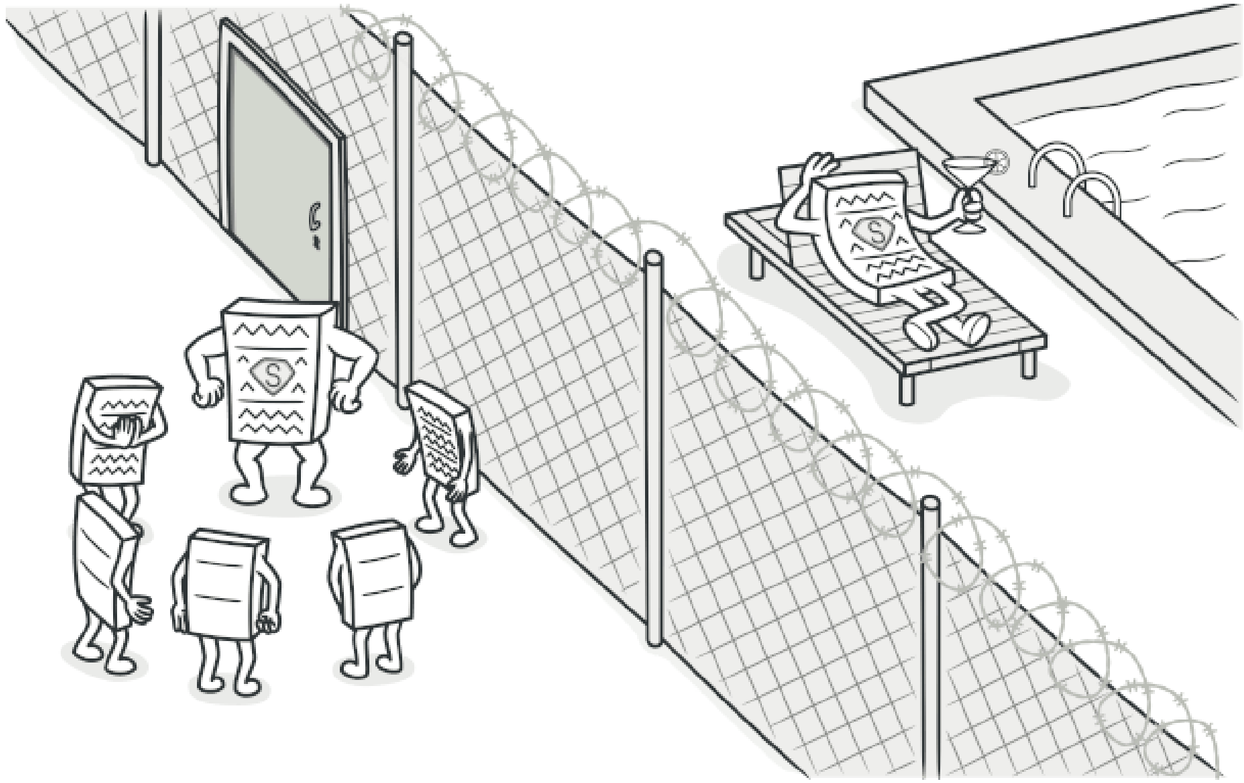
PROXY

Esteban Castañeda Blanco
Daniel Lizano Morales

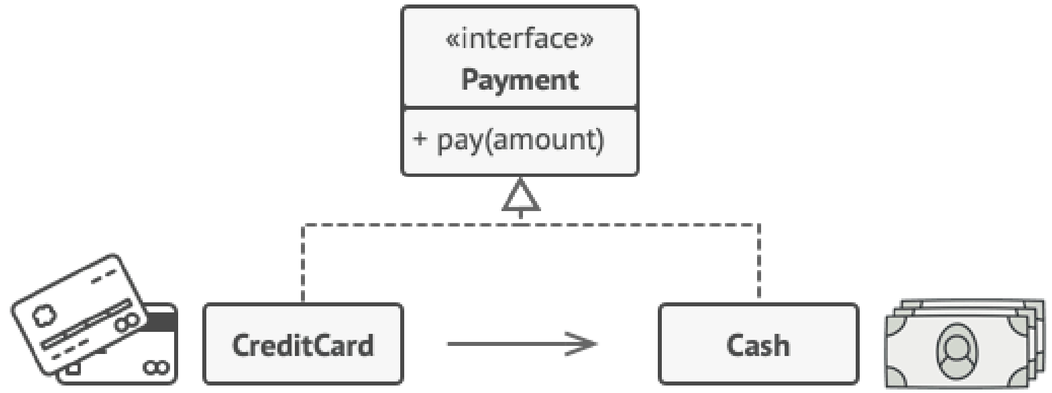
¿Que es Proxy?

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



Analogía de la tarjeta de crédito

Una tarjeta de crédito es un representante de una cuenta bancaria, que es un representante de un paquete de efectivo. Ambos implementan la misma interfaz: se pueden utilizar para realizar un pago. Un consumidor se siente muy bien porque no hay necesidad de llevar un montón de dinero en efectivo. El dueño de una tienda también está feliz ya que los ingresos de una transacción se agregan electrónicamente a la cuenta bancaria de la tienda sin el riesgo de perder el depósito o que le roben en el camino al banco.



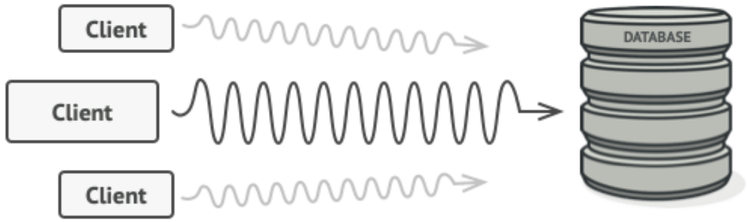
Las tarjetas de crédito se pueden utilizar para pagos de la misma manera que en efectivo.

Problema

Hay objeto masivo que consume una gran cantidad de recursos del sistema. Lo necesitas de vez en cuando, pero no siempre.

Podría implementar la inicialización diferida: cree este objeto solo cuando sea realmente necesario. Todos los clientes del objeto necesitarían ejecutar algún código de inicialización diferida. Desafortunadamente, esto probablemente causaría mucha duplicación de código.

En un mundo ideal, nos gustaría poner este código directamente en la clase de nuestro objeto, pero eso no siempre es posible. Por ejemplo, la clase puede ser parte de una biblioteca de terceros cerrada.

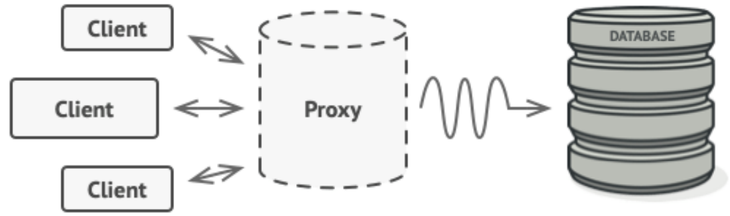


Las consultas a la base de datos pueden ser muy lentas.

Solución

El patrón Proxy sugiere que cree una nueva clase de proxy con la misma interfaz que un objeto de servicio original. Luego, actualiza su 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.

Si necesita ejecutar algo antes o después de la lógica principal de la clase, el proxy le permite hacerlo sin cambiar esa clase. Dado que el proxy implementa la misma interfaz que la clase original, se puede pasar a cualquier cliente que espere un objeto de servicio real.



El proxy se disfraza como un objeto de base de datos. Puede manejar la inicialización diferida y el almacenamiento en caché de resultados sin que el cliente o el objeto de la base de datos real lo sepan.

Ventajas

PUEDE CONTROLAR EL OBJETO DE SERVICIO SIN QUE LOS CLIENTES LO SEPAN

PUEDE ADMINISTRAR 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 DISPONIBLE

PRINCIPIO ABIERTO/CERRADO . PUEDE INTRODUCIR NUEVOS PROXIES SIN CAMBIAR EL SERVICIO O LOS CLIENTES

Desventajas

EL CÓDIGO PUEDE VOLVERSE MÁS COMPLICADO YA QUE NECESITA INTRODUCIR MUCHAS CLASES NUEVAS

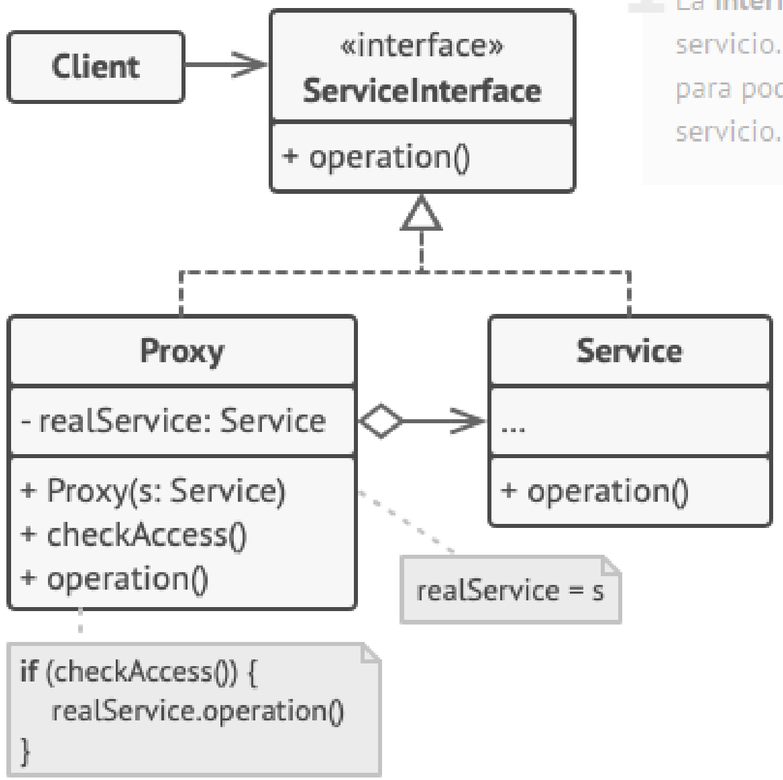
LA RESPUESTA DEL SERVICIO PUEDE RETRASARSE

¿Cuando aplicarlo?

- **Inicialización diferida (proxy virtual)**
- **Control de acceso (proxy de protección)**
- **Ejecución local de un servicio remoto (proxy remoto)**
- **Solicitudes de registro (proxy de registro)**
- **Almacenamiento en caché de los resultados de la solicitud (caching proxy)**

Estructura

1 La **interfaz de servicio** declara la interfaz del servicio. El proxy debe seguir esta interfaz para poder disfrazarse como un objeto de servicio.



2 El **Servicio** es una clase que proporciona una lógica empresarial útil.

3 La clase **Proxy** tiene un campo de referencia que apunta a un objeto de servicio. Una vez que el proxy finaliza su procesamiento (p. ej., inicialización diferida, registro, control de acceso, almacenamiento en caché, etc.), pasa la solicitud al objeto de servicio.

Por lo general, los proxies administran el ciclo de vida completo de sus objetos de servicio.

4 El **Cliente** debe trabajar con servicios y servidores proxy a través de la misma interfaz. De esta forma, puede pasar un proxy a cualquier código que espere un objeto de servicio.

Implementación

1. Si no existe una interfaz de servicio preexistente, cree una para que los objetos de proxy y de servicio sean intercambiables.
2. Cree la clase de proxy. Debe tener un campo para almacenar una referencia al servicio.
3. Implementar los métodos proxy de acuerdo a sus propósitos. En la mayoría de los casos, después de realizar algún trabajo, el proxy debe delegar el trabajo al objeto de servicio.
4. Considere introducir un método de creación que decida si el cliente obtiene un proxy o un servicio real.
5. Considere implementar la inicialización diferida para el objeto de servicio.

Ejemplo de código

```cpp
// Interfaz común para el sujeto real y el proxy
class Subject {
public:
    virtual void request() = 0;
};

// Sujeto real que realiza la operación
class RealSubject : public Subject {
public:
    void request() override {
        std::cout << "RealSubject: Manejando la solicitud." << std::endl;
    }
};

// Proxy que actúa como intermediario entre el cliente y el sujeto real
class Proxy : public Subject {
private:
    RealSubject* realSubject;

public:
    void request() override {
        if (realSubject == nullptr) {
            realSubject = new RealSubject();
        }

        preRequest();
        realSubject->request();
        postRequest();
    }

    void preRequest() {
        std::cout << "Proxy: Realizando tareas previas a la solicitud." << std::endl;
    }

    void postRequest() {
        std::cout << "Proxy: Realizando tareas posteriores a la solicitud." << std::endl;
    }
};
```

```
int main() {
    Proxy proxy;
    proxy.request();

    return 0;
}
```

Consecuencias

CONSECUENCIA #1

Un Proxy puede ocultar el hecho de que un objeto reside en un espacio de direcciones diferente

CONSECUENCIA #2

Puede llevar a cabo optimizaciones tales como crear un objeto por encargo

CONSECUENCIA #3

Permiten realizar tareas de mantenimiento adicionales cuando se accede a un objeto

Relación con otros patrones

ADAPTER

Adapter proporciona una interfaz diferente al objeto envuelto, Proxy le proporciona la misma interfaz.

FACADE

Ambos amortiguan una entidad compleja y la inicializan por sí sola. A diferencia de Facade , Proxy tiene la misma interfaz que su objeto de servicio, lo que los hace intercambiables.

DECORATOR

Ambos patrones se basan en el principio de composición. La diferencia es que un Proxy generalmente administra el ciclo de vida de su objeto de servicio por sí mismo, mientras que la composición de Decorators siempre la controla el cliente

Referencias bibliograficas

Refactoring.Guru. (2023b). Proxy. Refactoring.Guru. https://refactoring.guru/design-patterns/proxy