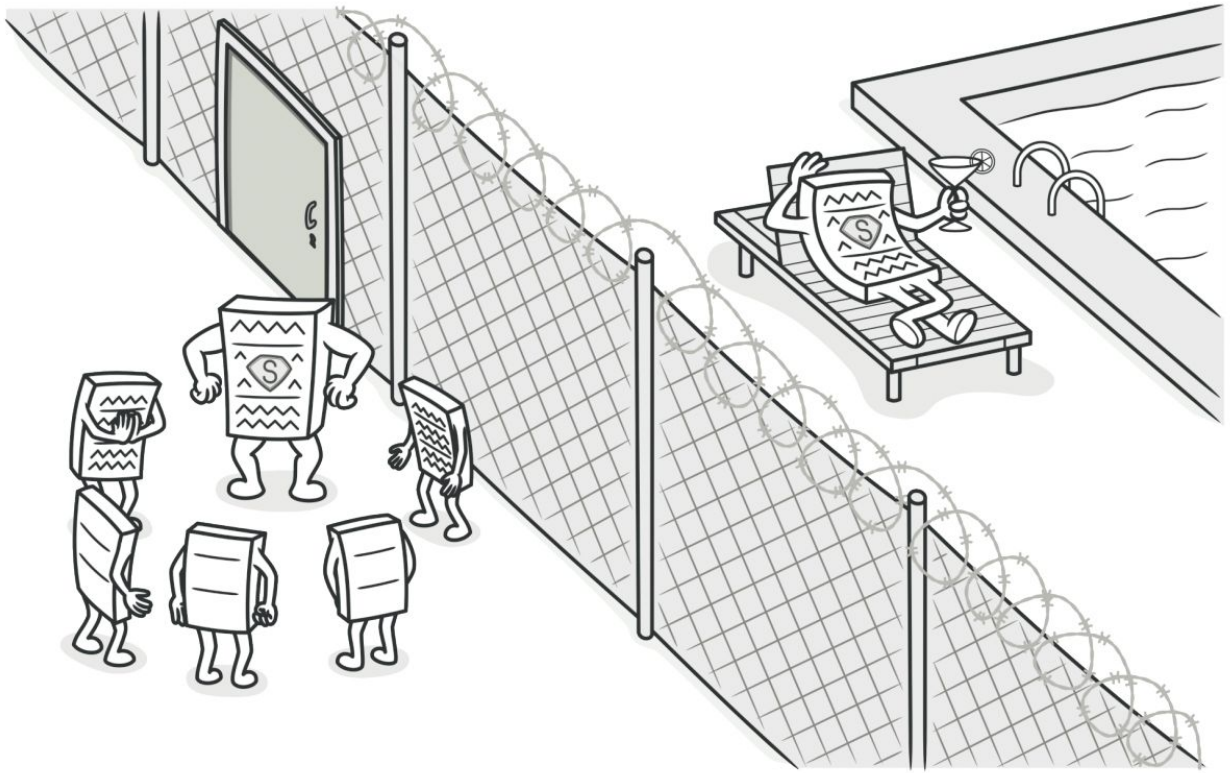
Lecture X: Design Patterns Proxy

...

Proxy



Proxy

Proxy is a structural design pattern that lets you provide a substitute or placeholder for another object.

A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

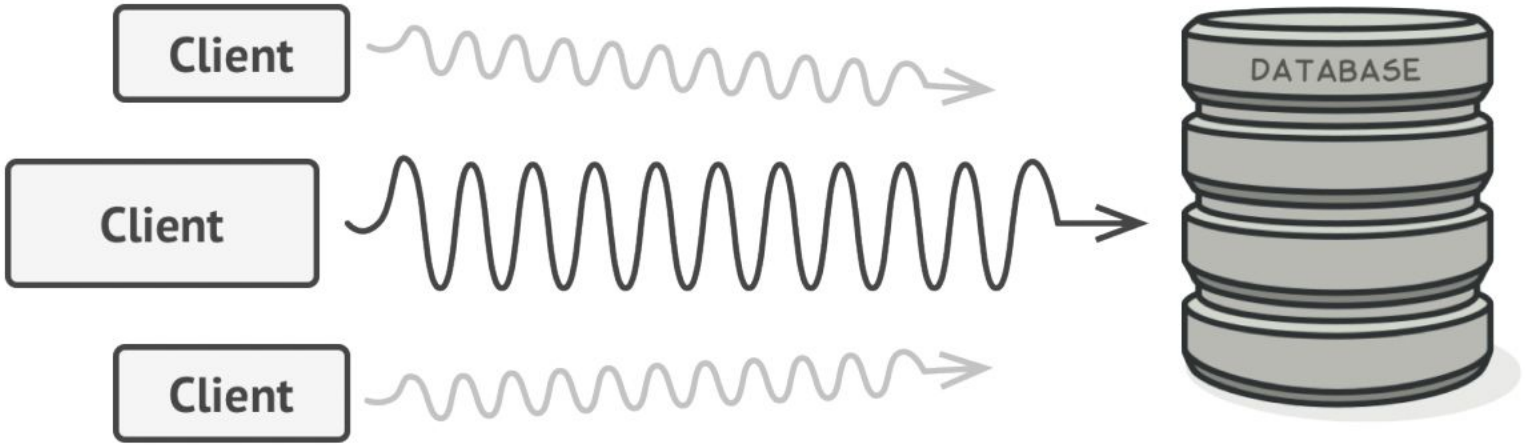
Proxy

Why would you want to control access to an object?

You have a massive object that consumes a vast amount of system resources.

You need it from time to time, but not always.

Proxy



Database queries can be really slow.

Proxy

You could implement lazy initialization: create this object only when it's actually needed. All of the object's clients would need to execute some deferred initialization code. Unfortunately, this would probably cause a lot of code duplication.

In an ideal world, we'd want to put this code directly into our object's class, but that isn't always possible. For instance, the class may be part of a closed 3rd-party library.

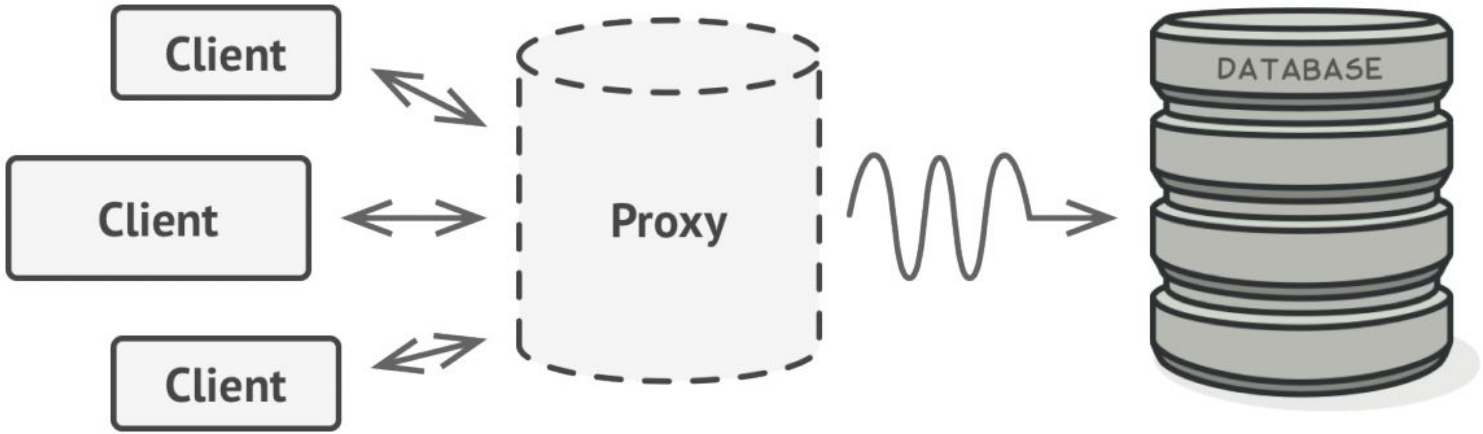
Proxy: Solution

The Proxy pattern suggests that you create a new proxy class with the same interface as an original service object

update your app so that it passes the proxy object to all of the original object's clients.

Upon receiving a request from a client, the proxy creates a real service object and delegates all the work to it.

Proxy: Solution



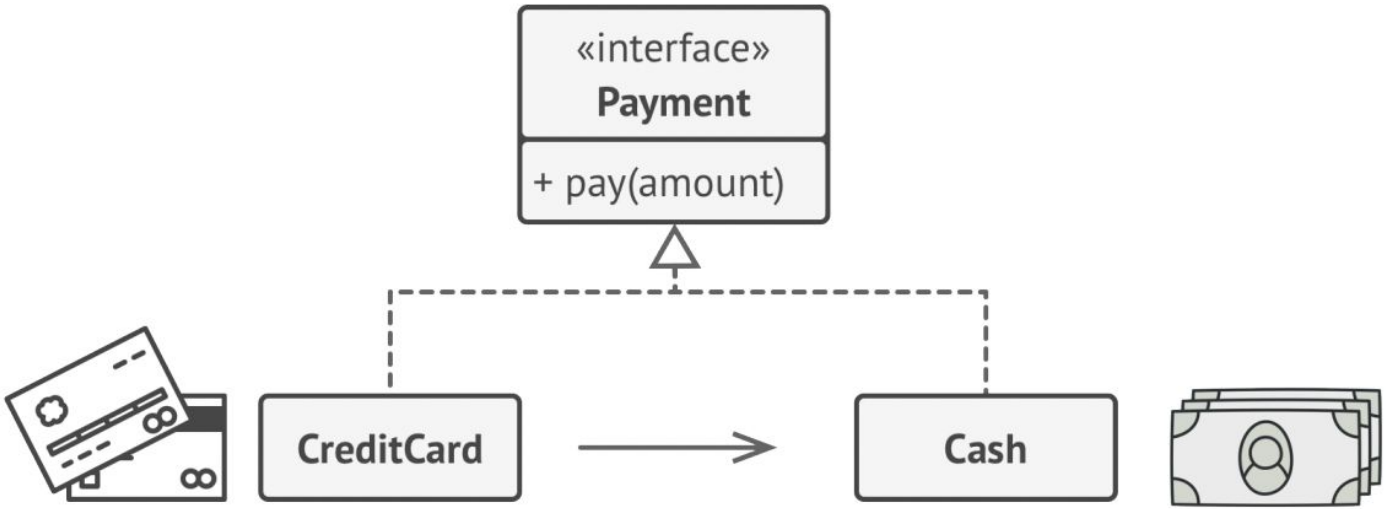
The proxy disguises itself as a database object. It can handle lazy initialization and result caching without the client or the real database object even knowing.

Proxy: Benefits

If you need to execute something
either before or after the primary logic of the class, the proxy lets you do this without changing that class.

Since the proxy
implements the same interface as the original class, it can be passed to any client that expects a real service object.

Proxy: Example



Credit cards can be used for payments just the same as cash.

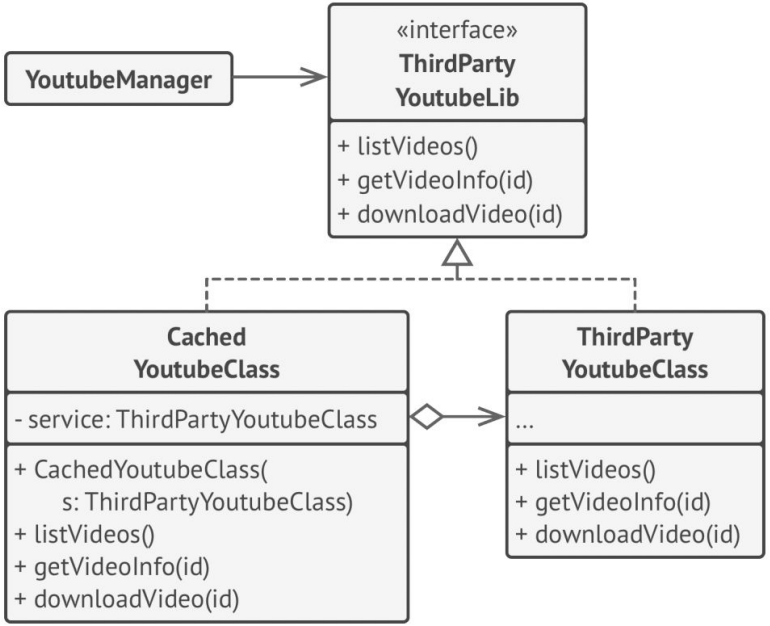
Proxy: Real-World Example

A credit card is a proxy for a bank account, which is a proxy for a bundle of cash. Both implement the same interface: they can be used for making a payment.

no need to carry loads of cash around.

shop owner is also happy since the income from a transaction gets added electronically to the shop's bank account without the risk of losing the deposit or getting robbed on the way to the bank.

Proxy: Real-World Example



Caching results of a service with a proxy.

Proxy: Real-World Example

The library provides us with the video downloading class. However, it's very inefficient. If the client application requests the same video multiple times, the library just downloads it

over and over, instead of caching and reusing the first downloaded file.

The proxy class implements the same interface as the original downloader and delegates it all the work. However, it keeps track of the downloaded files and returns the cached result when the app requests the same video multiple times.

Proxy: Pros and Cons

You can control the service object without clients knowing about it.
You can manage the lifecycle of the service object when clients don’t care about it.
The proxy works even if the service object isn’t ready or is not available.
Open/Closed Principle. You can introduce new proxies without changing the service or clients.

The response from the service might get delayed.

How to implement

1. If there's no pre-existing service interface, create one to make proxy and service objects interchangeable. Extracting the interface from the service class isn't always possible, because you'd need to change all of the service's clients to use that interface. Plan B is to make the proxy a subclass of the service class, and this way it'll inherit the interface of the service.

2. Create the proxy class. It should have a field for storing a reference to the service. Usually, proxies create and manage the whole life cycle of their servers. In rare occasions, a service is passed to the proxy via a constructor by the client.
3. Implement the proxy methods according to their purposes. In most cases, after doing some work, the proxy should delegate the work to the service object.
4. Consider introducing a creation method that decides whether the client gets a proxy or a real service. This can be a simple static method in the proxy class or a full-blown factory method.

5. Consider implementing lazy initialization for the service object.