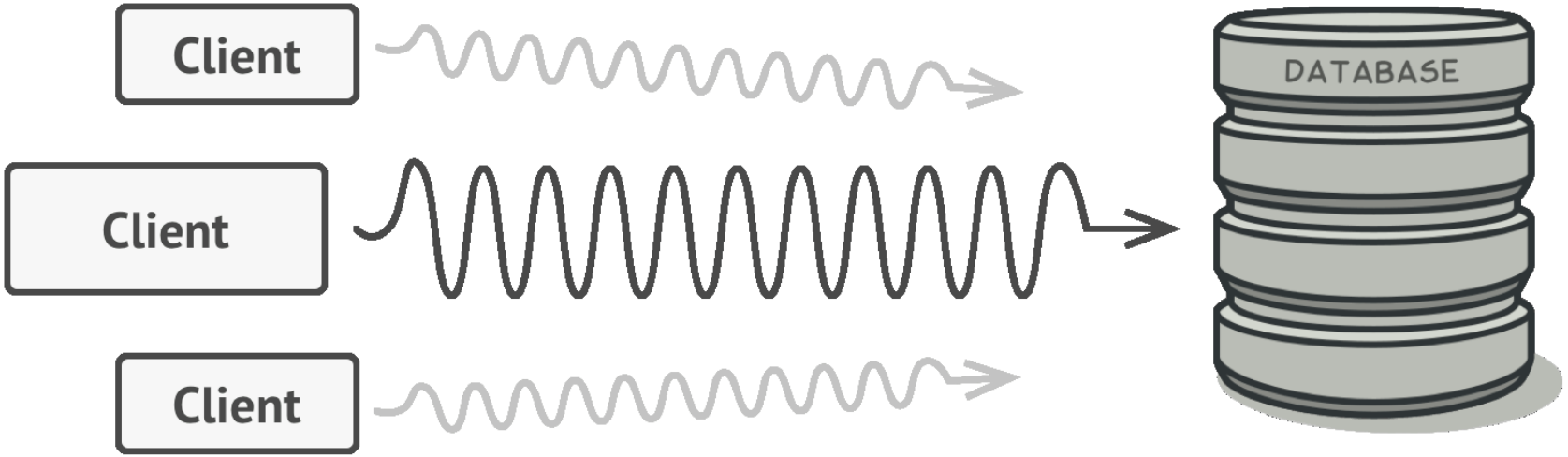
Proxy

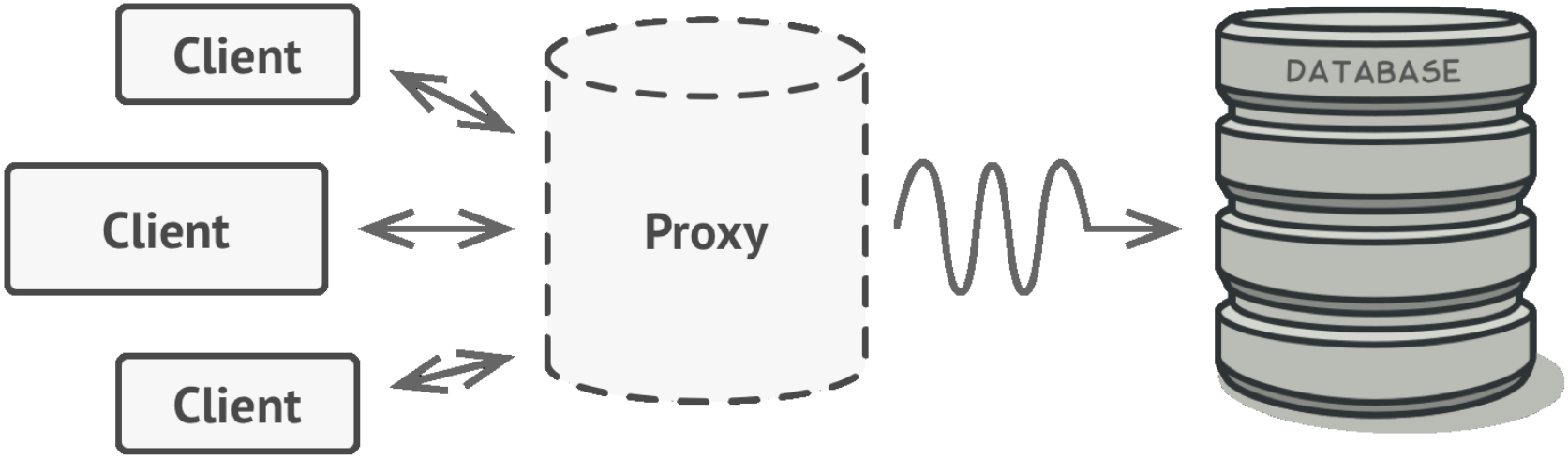
Proxy is a structural design pattern that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

Problem



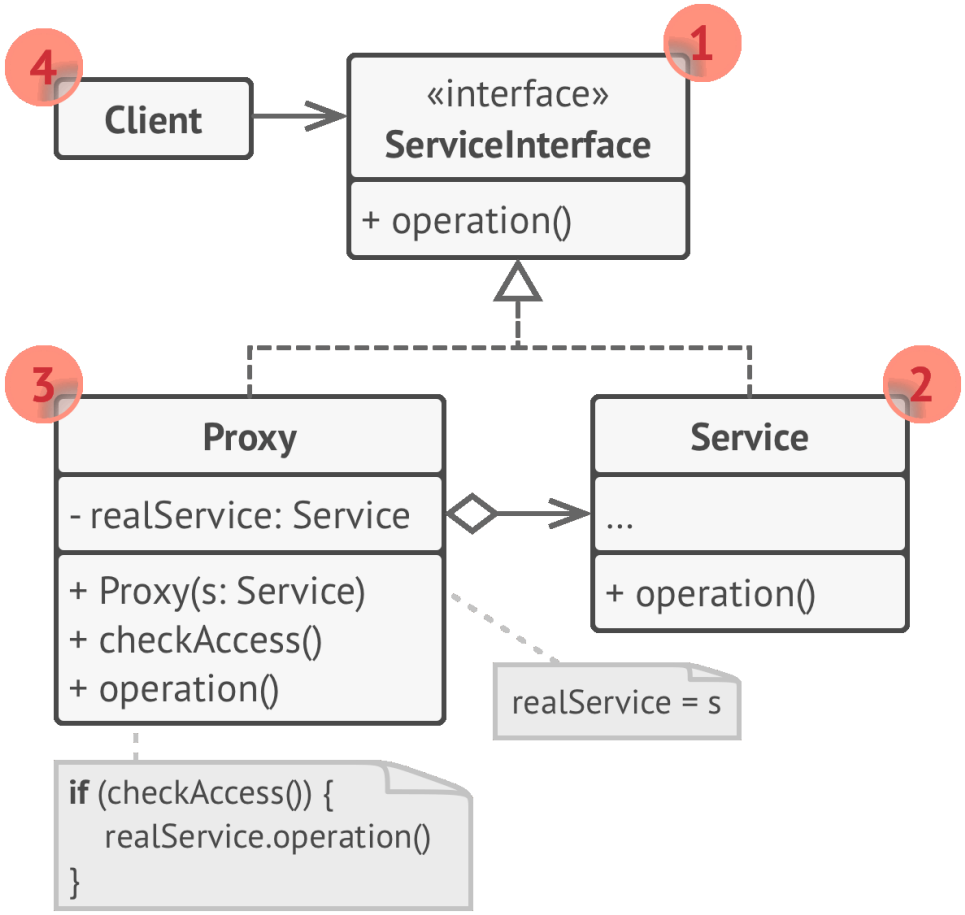
Database queries can be really slow

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.

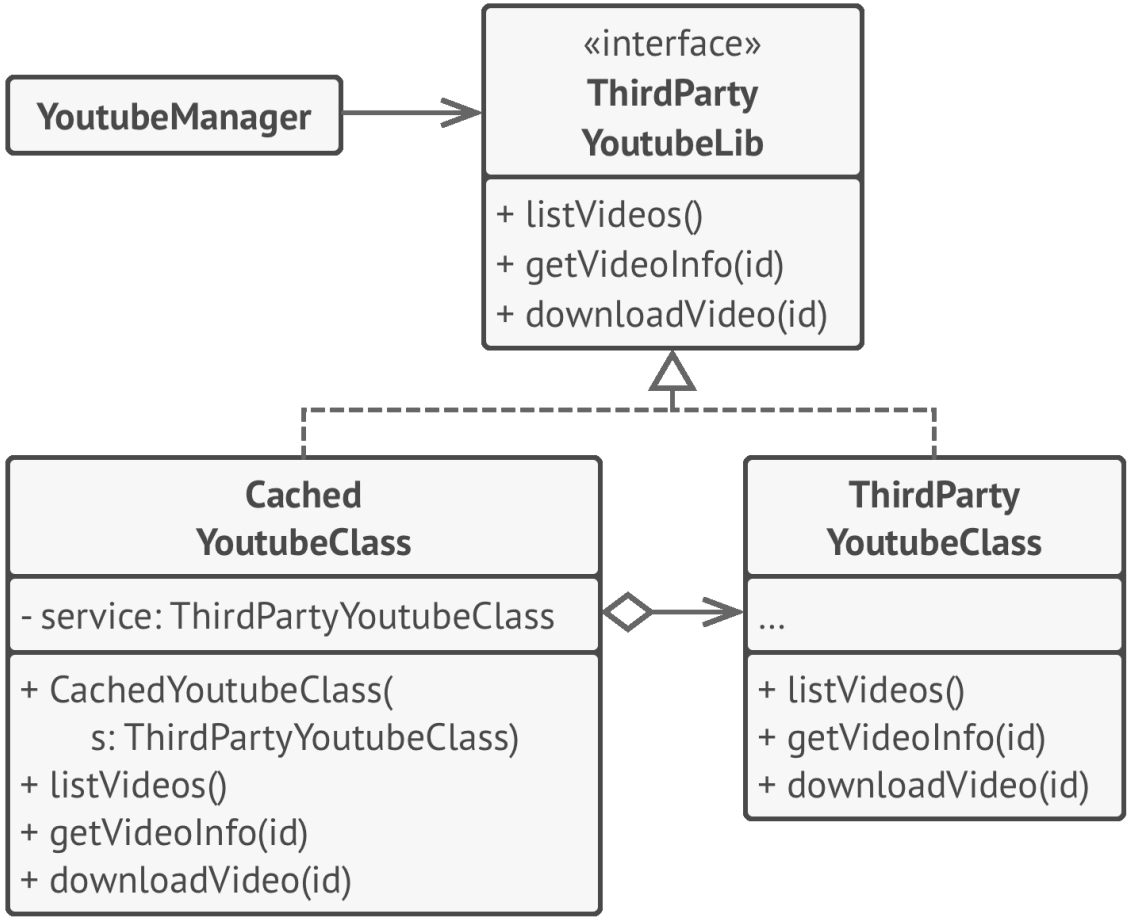
Structure


4
Client
1
«interface»
ServiceInterface
+ operation()
3
Proxy
- realService: Service
+ Proxy(s: Service)
+ checkAccess()
+ operation()
2
Service
...
+ operation()
realService = s
if (checkAccess()) {
realService.operation()
}

Contd.

- The **Service Interface** declares the interface of the Service.
- The **Service** is a class that provides some useful business logic
- The **Proxy** class has a reference field that points to a service object.
- The **Client** should work with both services and proxies via the same interface.

Implementation



Caching results of a service with a proxy.

Application

- Lazy initialization (virtual proxy)
- Local execution of a remote service (remote proxy)
- Caching request results (caching proxy)

Pros and Cons

- The service objects can be controlled without clients knowing about it.
- The lifecycle of the service object can be managed 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*. Introduction to new proxies can be done without changing the service or clients.
- The code may become more complicated as it is needed to introduce a lot of new classes.
- The response from the service might get delayed.