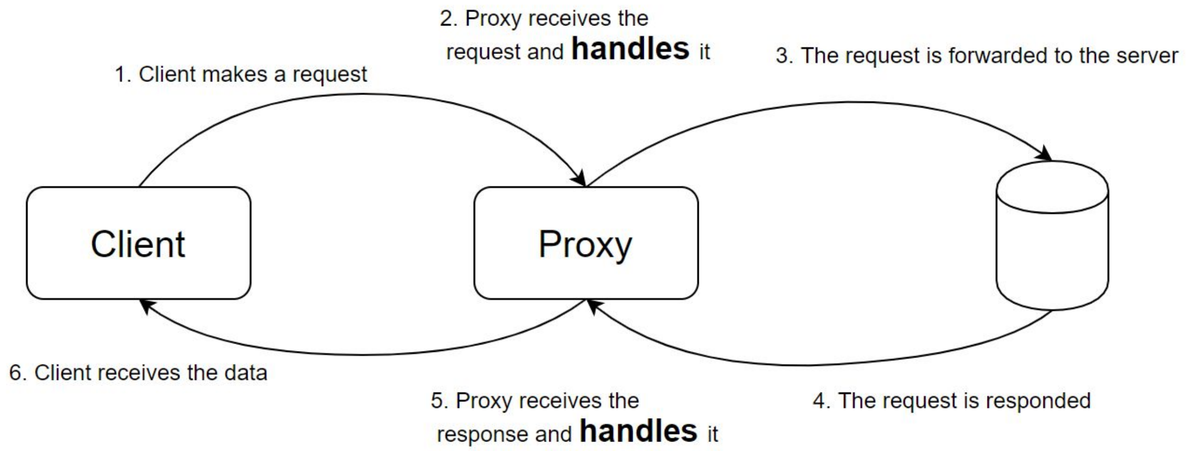
Proxy

Structural pattern

Why proxy?


2. Proxy receives the request and **handles** it
3. The request is forwarded to the server
1. Client makes a request
Client
Proxy
6. Client receives the data
5. Proxy receives the response and **handles** it
4. The request is responded

General scenario

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.

What does “handle” mean in our case?

- **Access control** (protection proxy). The proxy can filter requests from unauthorized clients. The proxy can pass the request to the service object only if the client’s credentials match some criteria.

- **Logging requests** (logging proxy). This is when you want to keep a history of requests to the service object.

- **Caching request results** (caching proxy). This is when you need to cache results of client requests and manage the life cycle of this cache, especially if results are quite large. The proxy can implement caching for recurring requests that always yield the same results. The proxy may use the parameters of requests as the cache keys.

Interface example

```
public interface Internet
{
    public void connectTo(String serverhost) throws
Exception;
}
```

```
public class ProxyInternet implements Internet {
    private Internet internet = new RealInternet();
    private static List<String> bannedSites;
    static{
        bannedSites = new ArrayList<String>();
        bannedSites.add("abc.com");
        bannedSites.add("def.com");
        bannedSites.add("ijk.com");
        bannedSites.add("lnm.com");
    }
    @Override
    public void connectTo(String serverhost) throws
Exception {

if(bannedSites.contains(serverhost.toLowerCase())) {
            throw new Exception("Access Denied");
        }
        internet.connectTo(serverhost);
    }
}
```

```
public class RealInternet implements Internet {
    @Override
    public void connectTo(String serverhost) {
        System.out.println("Connecting to "+ serverhost);
    }
}
```

Client example

```
public class Client
{
    public static void main (String[] args) {
        Internet internet = new ProxyInternet();
        try{

            internet.connectTo("geeksforgeeks.org");
            internet.connectTo("abc.com");

        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
}
```

Pros & 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.
- *Open/Closed Principle*. You can introduce new proxies without changing the service or clients.

- The response from the service might get delayed.