Proxy Design Pattern:  
**Other name (if any)**

Proxy pattern is also known as **Surrogate or Placeholder.**

**What it does**

Simply, proxy means an object representing another object. It is a design pattern where clients do not interact directly with the real/actual object rather it interact with the object through other objects which are called proxy object. There could be multiple layer of proxy objects. For each layer, the object of next layer is considered to be the actual object. Proxy layers are not fully aware of that fact that in which layer the actual object is.

**Where to use**

1. Access restriction: It can be used in **Protective Proxy** scenario. The proxy layer has list of blocked users. Before interacting with the real object the proxy layer return the message access denied to the client. Use a proxy when you need to control and manage access to an object, ensuring that certain conditions or permissions are met before allowing clients to interact with the real object.  
2. Caching: If there is caching object in  the proxy layer, it will return the cached object instead of real object.  
3. Pre processing and post processing: Some processing will be occured in the proxy layer before interacting with the real object and/or after interacting with the real object while returning back to the client.  
4. It can be used in **Virtual Proxy** scenario. The real object gets created only when a client first requests/accesses the object and after that we can just refer to the proxy to reuse the object. This avoids duplication of the object and hence saving memory.  
5. It can be used in **Remote Proxy** scenario. The remote proxy provides a local representation of the object which is present in the different address location.  
6. It can be used in **Smart Proxy** scenario. A smart proxy provides additional layer of security by interposing specific actions when the object is accessed.  
7. One of the primary use cases for proxies is lazy loading. In situations where creating or initializing an object is resource-intensive, the proxy delays the creation of the real object until it is actually needed.  
8. Use a proxy when you want to postpone the creation of a resource-intensive object until it’s actually needed.  
9. Use a proxy to optimize the utilization of resources, such as caching results or storing previously fetched data.

**Steps**

1. ProxyReal Interface: It has all the methods the real and proxy object needs.  
2. Concrete Class For Real Object: It will implement the proxyReal interface.  
3. Concrete Class For Proxy Object: It will implement the proxyReal interface. It will contain an object of concrete class for real object. After do some processing it will return the method of the concrete class for real object.  
4. Client Code: It will create object using ProxyReal interface and concrete class for proxy object. And call necessary function of the concrete class for proxy object.

**Advantages**

1. It provides the protection to the original object from the outside world.  
2. The proxy delays the creation of the real object until it is actually needed. This can lead to improved performance by avoiding unnecessary resource allocation.  
3. By acting as a gatekeeper to the real object, proxies can restrict access based on certain conditions, providing security or permission checks.  
4. Protection proxies control access to a real object by adding an additional layer of security checks. They can ensure that the client code has the necessary permissions before allowing access to the real object.  
5. Proxies can implement caching mechanisms to store results or resources. It avoids redundant computations or data fetching.  
6. By intercepting method calls to the real object, proxies can log information, track usage, or measure performance without modifying the real object.

**Disadvantages**

1. Implementing proxies can add complexity to the codebase, especially when dealing with multiple layers of indirection.  
2. Adding proxies typically requires additional classes and interfaces, increasing the overall size of the codebase.  
3. Managing proxies alongside real subjects can make the codebase harder to maintain and understand, especially for developers new to the code.  
4. In multithreaded environments, ensuring proper synchronization between proxies and real subjects can be challenging.  
5. The use of proxies may introduce latency or delays, particularly if there are network calls or heavy computations involved in proxy operations.  
6. If the proxy fails, it can impact the entire system's functionality, especially if critical operations rely on the proxy. Then it becomes single point of failure.  
7. Managing dependencies between proxies and real subjects can become complex, especially in systems with multiple layers of abstraction.  
8. Testing code involving proxies may require additional effort, as both proxy behavior and real subject behavior need to be tested separately and in combination.  
9. In distributed systems, managing proxies and ensuring consistency between proxies and real subjects can be complex, especially in dynamic environments.  
10. Proxies may cache data or results, leading to stale or outdated information if not managed properly, impacting system correctness and performance.

**Code**

Tutorialspoint

**Difference with similar pattern**

**Decorator Pattern**: Like Proxy, it enhances the behavior of an object by wrapping it with another object. However, Decorator focuses on adding functionality dynamically, while Proxy focuses on controlling access to the real subject.

**Adapter Pattern**: Also known as the Wrapper Pattern, it converts the interface of a class into another interface that clients expect. While similar to Proxy in structure, Adapter focuses on making incompatible interfaces compatible, rather than controlling access to objects.  
May be flyweight and bridge design pattern but [[later]].  
These patterns share similarities with the Proxy Design Pattern in terms of structure or intent but differ in their specific use cases and implementations.

**Diagram**

Tutorialspoint