Proxy Pattern

> Provide a surrogate or placeholder for another object to control access to it.

- “Control Access”
    - Not just controlling access, but sometimes also providing ease of access.
- There are several different versions of Proxy Pattern. Can be said that this is a Pattern of Patterns.
- A **remote** proxy is useful to talk to an external system, giving an illusion of a local access.
    - A remote proxy gives you the illusion that you’re talking to a local object when you are really talking to a remote object.
    - Further, this proxy could be extended to cache the data. If the data isn’t changing very frequently, you can work with the data locally that it cached and save a round trip to the server.
- A **virtual** proxy can help by postponing the creation of expensive resources until needed.
- A spy used in testing is an example of proxy.
- A proxy encapsulates the interface of the object it represents.
- Example that would benefit from Proxy Pattern?
    - An application needs to communicate with a remote subsystem to obtain some critical information.
    - The application may have code all over that deals with communication logic and data.
    - How to minimise the code for communication logic?
    - What if not all data on the remote subsystem is changing dynamically?

When to use Proxy Pattern?

- When a more sophisticated reference than a simple pointer is needed.
- Remote proxy provides local representation for object in different address space.
- Virtual proxy created expensive objects on demand.
- Protection proxy controls access to original object.
- Copy-on-modify proxy guts of data not copied until if and when data modification starts.
- Smarts pointers (in C++) are needed to
    - manage object lifetime
    - loading object into memory when first referenced
- Local management for synchronisation.

Consequences of using Proxy

- Introduces a level of indirection in accessing objects.
    - Indirection provides flexibility
- Incurred cost on additional object and computations.

Proxy vs. Other Patterns

- Adapter changes interface. Proxy provides the same interface.
- Protection proxy implements subset of interface — may deny access to certain functions.
- Decorator adds functionality (or enhances it). Proxy controls functionality.