Proxy Design Pattern

Proxy Design Pattern: The *Proxy Design Pattern* is a *structural design pattern* that provides a *placeholder* or *representative* for another object to control access to it.

In simple terms: Instead of *interacting* with the *real object* directly, the client interacts with a *proxy object*. The proxy internally manages *how* and *when* to *communicate* with the real object.

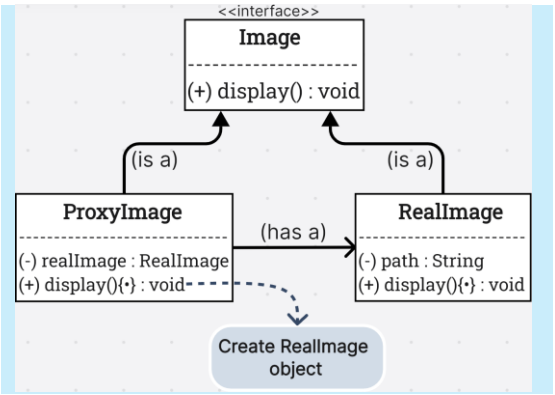
Types of Proxy:

1. *Virtual Proxy* – *Controls access* to *resource-intensive* objects, creating them only when necessary (*lazy initialization*).
   Example: Loading an image only when it's actually needed.
2. *Remote Proxy* – Represents an *object* that *exists* in a *different address space* (e.g., *another* machine or *server*).
   Example: RMI (Remote Method Invocation) in Java.
3. *Protection Proxy* – *Controls access* to the *real object* based on *access rights*.
   Example: Restricting certain operations based on user roles.
4. *Cache Proxy* – Provides *temporary storage* to reduce *expensive operations* (like repeated *network/database* calls).

1. *Virtual Proxy*

Problem:

Loading a *large image file* from disk is *expensive* and *time-consuming*. If you have multiple images in a gallery, you don't want to load them all at once, only when the user actually views one.



Solution (Virtual Proxy):

- RealImage (Real Subject): Represents the *actual heavy image* that takes time to load.
- ProxyImage (Virtual Proxy): Represents a *lightweight placeholder* that only loads the real image when needed.
- Client: Works with Image interface and doesn't care whether it's a proxy or the real image.

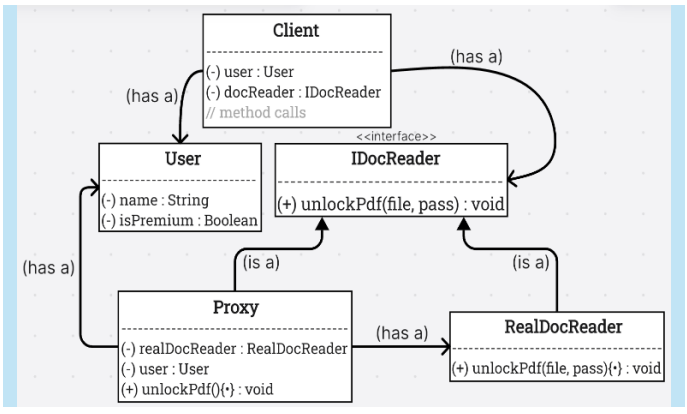
Flow:

1. Client calls *display()* on *ProxyImage*.
2. *ProxyImage* checks if *RealImage* is already loaded.
   - If not → *load* it from disk (*expensive operation*).
   - If yes → just *display* it.
3. This way, image is loaded *only when required* (lazy loading).

2. *Protection Proxy*

Problem:

You have a *locked PDF document*. Only *premium users* are allowed to *remove* the *password* and view the document freely. *Normal users* can view it only in *read-only mode*.



Solution (Protection Proxy):

- RealDocument (RealSubject): Represents the *actual PDF* and has the method to *remove* the *password*.
- ProxyDocument (Protection Proxy): Controls access to the *unlockPdf()* feature based on *user type*.
- Client: Works with Document interface and doesn't need to know whether it's talking to the *real* document or the *proxy*.

@Sibasundarj8

Flow:

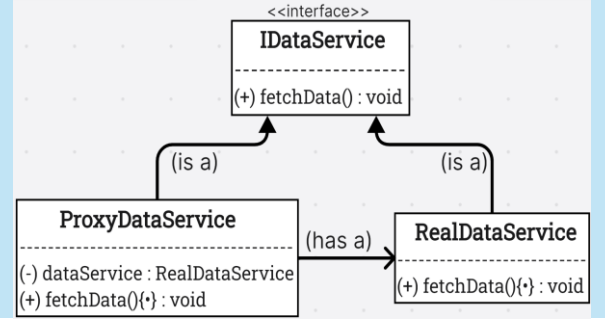
1. Client calls ***unlockPdf()*** on ***ProxyDocument***.
2. ***ProxyDocument*** checks the user type:
    - If ***premium*** → forward request to ***RealDocument*** to remove the password.
    - If ***normal*** → ***deny access*** or show a message "Upgrade to premium to remove password."
3. ***Reading/viewing*** the PDF can be allowed for everyone, but the sensitive feature (password removal) is protected.

3. *Remote Proxy*

Problem:

A client wants some ***data*** that is hosted on an ***online server***.
But the client:

- Doesn't know ***how to connect*** to the ***server***.
- Doesn't care about ***network setup***, ***authentication***, or ***protocols***.



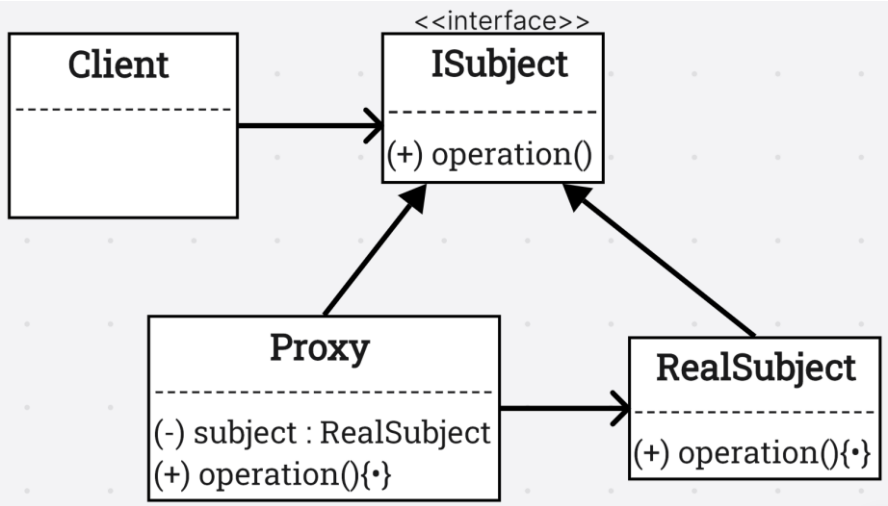
Solution (Remote Proxy):

- ***RealDataService*** (RealSubject): Runs on the ***server*** and knows ***how to fetch*** the data.
- ***ProxyDataService*** (Remote Proxy): Lives on the ***client side***. It looks like the real service but internally manages the ***connection*** to the remote server.
- ***Client***: Just calls ***getData()*** on the ***proxy object*** as if it were local.

Flow:

- Client calls ***getData()*** on ***ProxyDataService***.
- ***ProxyDataService lazily*** establishes the ***network connection*** with the ***remote server*** (only when needed).
- ***Proxy*** forwards the request to the ***RealDataService*** on the server.
- ***Server*** responds → ***Proxy*** returns the ***data*** back to the client.

Standard UML:



Standard Defⁿ:

The proxy pattern provides a surrogate or placeholder for another object to control access to it.

- Remote
- Virtual
- Protection

Code link: *https://github.com/sibasundarj8/-System-Design-/tree/main/Codes/21___Proxy%20Design%20Pattern%20code*