Definition

*The Proxy Pattern provides a surrogate or placeholder for another object to control access to it.*

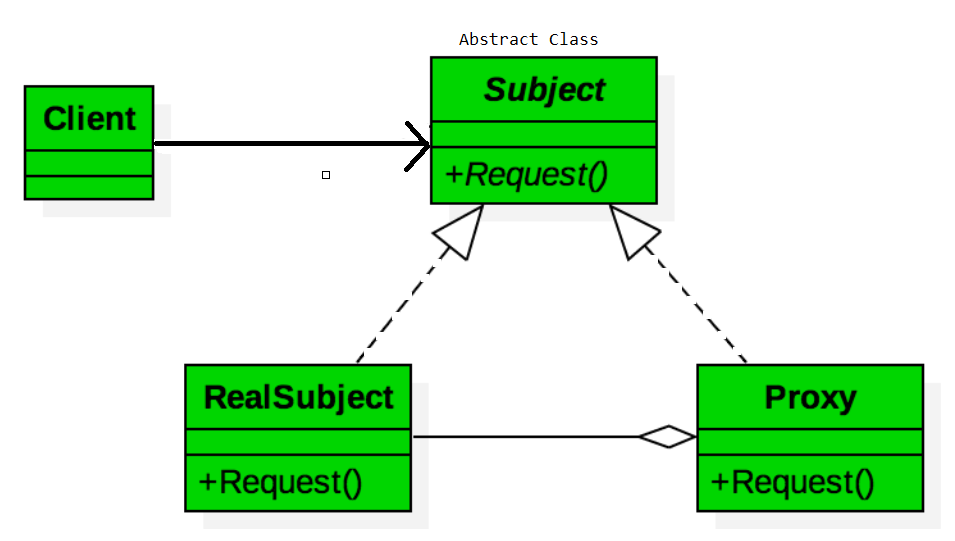
You need to support resource-hungry objects, and you do not want to instantiate such objects unless and until they are actually requested by the client.

For example, if we need to use only a few methods of some costly objects. We'll initialize those objects when we need them entirely. Until that point we can use some light objects exposing the same interface as the heavy objects.

These light objects are called proxies and they will instantiate those heavy objects when they are really needed.

Consider for example, an image viewer program.

An image viewer program must be able to list and display high resolution photo objects that are in a folder, but how often do someone open a folder and view all the images inside. Sometimes you will be looking for a particular photo, sometimes you will only want to see an image name. The image viewer must be able to list all photo objects, but the photo objects must not be loaded into memory until they are required to be rendered.



The above class diagram represents the Proxy Pattern in its simplest form. Our surrogate or place holder would be our ***Proxy*** object that controls the access to another object ***RealSubject***. Each implement the same interface, ***Subject***. This allows the client to implement to an interface and have no knowledge that it is interacting with a ***Proxy*** versus the ***RealSubject***.

The participants classes in the proxy pattern are:

* **Subject** - Interface implemented by the RealSubject and representing its services. The interface must be implemented by the proxy as well so that the proxy can be used in any location where the RealSubject can be used.
* **Proxy**
* Maintains a reference that allows the Proxy to access the RealSubject.
* Implements the same interface implemented by the RealSubject so that the Proxy can be substituted for the RealSubject.
* Controls access to the RealSubject and may be responsible for its creation and deletion.
* Other responsibilities depend on the kind of proxy.
* **RealSubject** - the real object that the proxy represents.

Types of Proxies

There are a few different types of proxies. Each control access to different types of resources.

1. **Remote Proxy** – Our RealSubject is a remote object whose access is controlled by a Proxy object
2. **Virtual Proxy** – Our RealSubject or a resource that is part of our RealSubject is expensive to create. The Proxy object, in this case, would defer the creation until it was absolutely needed.
3. **Protection Proxy** – Our RealSubject may require access protection of some kind. The Proxy would provide that protection based on access rights.
4. **Smart Reference** – Our Proxy would provide additional behavior such as reference counts of the RealSubject pointer. This is commonly known as a smart pointer. Additionally, the Proxy may provide a locking mechanism to protect the RealSubject from being changed by other objects while an access is already in progress. Lastly, the Proxy could be responsible for creating the RealSubject when it is first needed.

Benefits of the Proxy Pattern

Each type of proxy a primary benefit.

1. **Remote Proxy**‘s primary benefit is that it hides the fact that the *RealSubject* lives within another process.
2. **Virtual Proxy**‘s primary benefit is that it defer the creation of an object to only when it is absolutely needed.
3. **Protection** and **Smart Reference Proxies** have the same primary benefit of providing additional behaviour pertaining to the *RealSubject* when access to the *RealSubject* is needed.

### Example - Virtual Proxy Example.

Consider an image viewer program that lists and displays high resolution photos. The program must show a list of all photos however it does not need to display the actual photo until the user selects an image item from a list.