Proxy Design Pattern

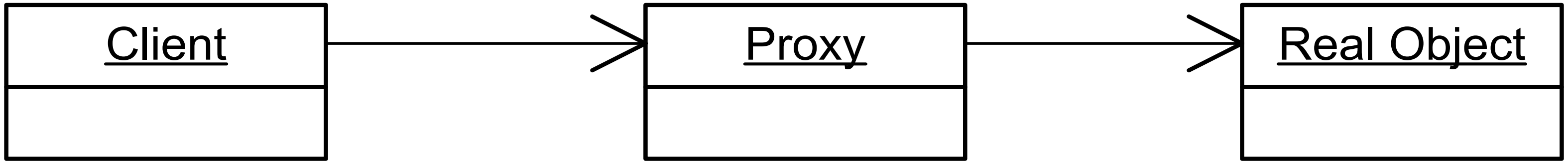
Source: Design Patterns – Elements of Reusable Object-Oriented Software; Gamma, et. al.

Problem

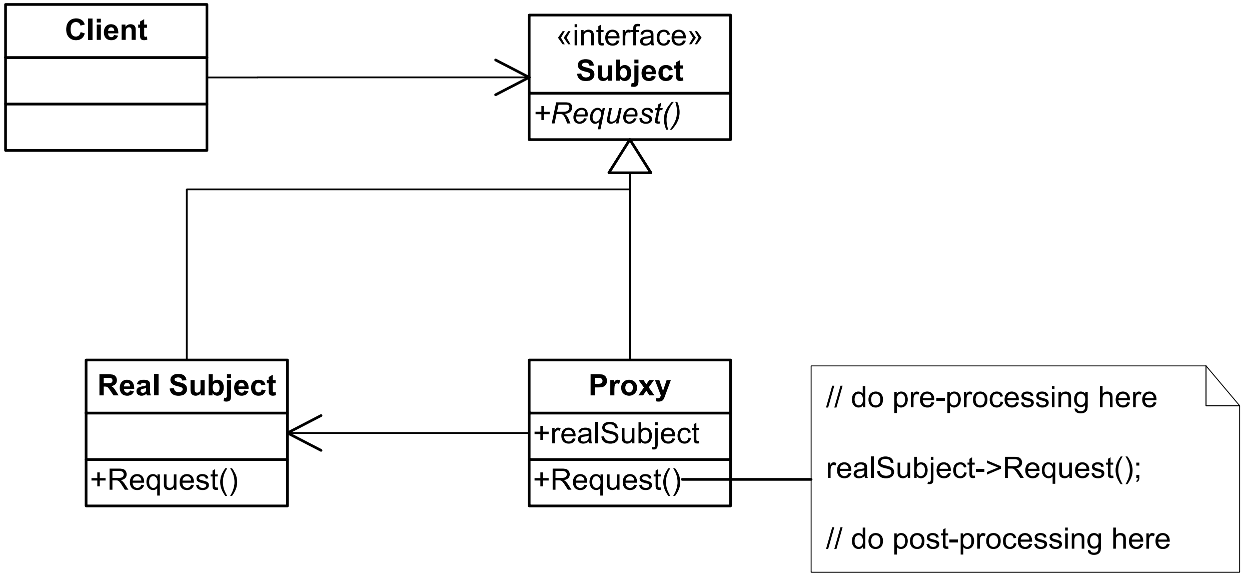
- You need to control access to an object

Solution

- Create a Proxy object that implements the same interface as the real object
- The Proxy object (usually) contains a reference to the real object
- Clients are given a reference to the Proxy, not the real object
- All client operations on the object pass through the Proxy, allowing the Proxy to perform additional processing



Solution



// do pre-processing here

realSubject->Request();

// do post-processing here

Consequences

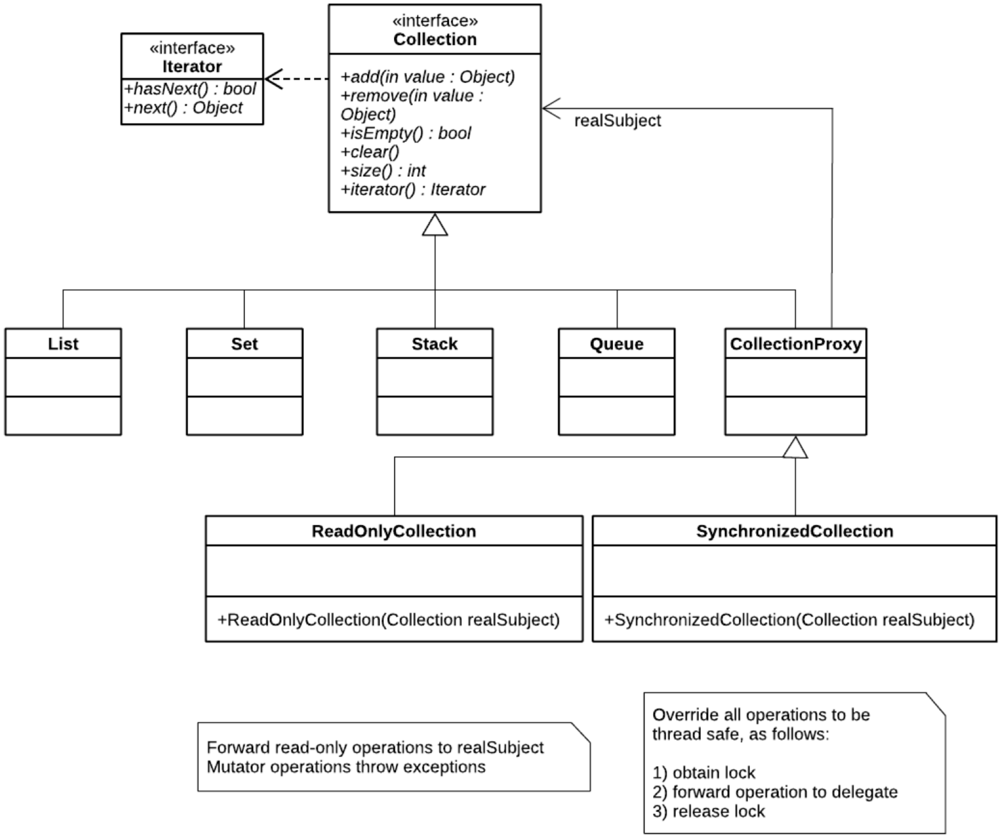
- Provides an additional level of indirection between client and object that may be used to insert arbitrary services

- Proxies are invisible to the client, so introducing proxies does not affect client code

Known Uses: Java Collections

- Read-only Collections
  - Wrap collection object in a proxy that only allows read-only operations to be invoked on the collection
  - All other operations throw exceptions
  - List Collections.unmodifiableList(List list);
    - Returns read-only List proxy
- Synchronized Collections
  - Wrap collection object in a proxy that ensures only one thread at a time is allowed to access the collection
  - Proxy acquires lock before calling a method, and releases lock after the method completes
  - List Collections.synchronizedList(List list);
    - Returns a synchronized List proxy

Java Collections



«interface»
Iterator

+hasNext() : bool
+next() : Object

«interface»
Collection

+add(in value : Object)
+remove(in value : Object)
+isEmpty() : bool
+clear()
+size() : int
+iterator() : Iterator

List

Set

Stack

Queue

CollectionProxy

ReadOnlyCollection

+ReadOnlyCollection(Collection realSubject)

SynchronizedCollection

+SynchronizedCollection(Collection realSubject)

Forward read-only operations to realSubject
Mutator operations throw exceptions

Override all operations to be thread safe, as follows:

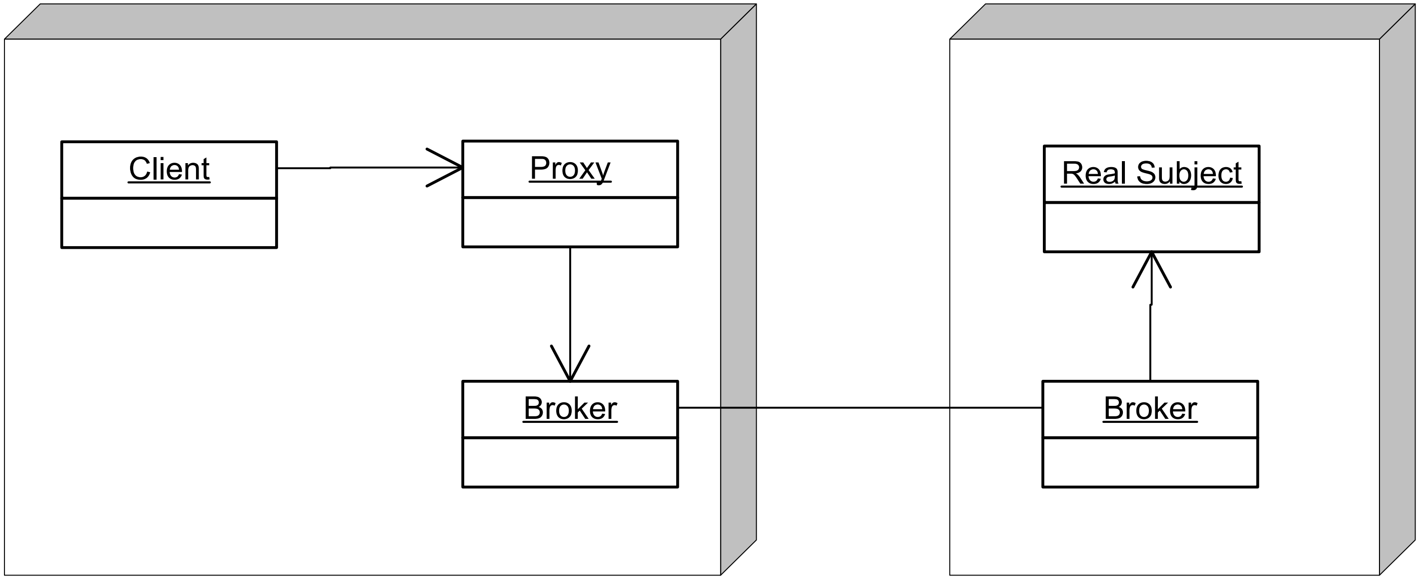
1) obtain lock
2) forward operation to delegate
3) release lock

```
Collection safeCollection = new SynchronizedCollection(new ArrayList());
// safely access collection from multiple threads
```

Known Uses: Distributed Objects

- The Client and Real Subject are in different processes or on different machines, and so a direct method call will not work

- The Proxy's job is to pass the method call across process or machine boundaries, and return the result to the client (with Broker's help)



Known Uses: Secure Objects

- Different clients have different levels of access privileges to an object
- Clients access the object through a proxy
- The proxy either allows or rejects a method call depending on what method is being called and who is calling it (i.e., the client's identity)

Known Uses: Lazy Loading

- Some objects are expensive to instantiate (i.e., consume lots of resources or take a long time to initialize)
- Rather than instantiating an expensive object right away, create a proxy instead, and give the proxy to the client
- The proxy creates the object on demand when the client first uses it
- If the client never uses the object, the expense of creating it is never incurred
- A hybrid approach can be used, where the proxy implements some operations itself, and only needs to create the real object if the client calls one of the operations it doesn't implement
- Proxies must store whatever information is needed to create the object on-the-fly (file name, network address, etc.)

Known Uses: Lazy Loading

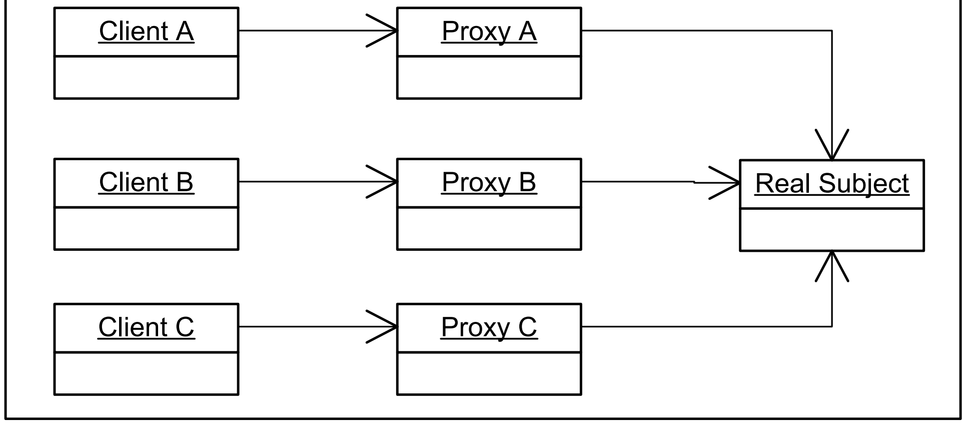
- Examples
- Object-Oriented Databases
  - Graph of objects stored on disk
  - Objects contain references to each other
  - Load proxies initially, and only load the real object from disk if a method is actually called on it
- Resource Conservation
  - If you need to store thousands of objects in memory at once, proxies can be used to save memory by only loading objects that are actually used
  - Objects that are used can be unloaded after awhile, freeing up memory
- Word Processor
  - Documents that contain lots of multimedia objects should still load fast
  - Create proxies that represent large images, movies, etc., and only load objects on demand as they become visible on the screen (only a small part of the document is visible at a time)

Known Uses: Copy-on-Write

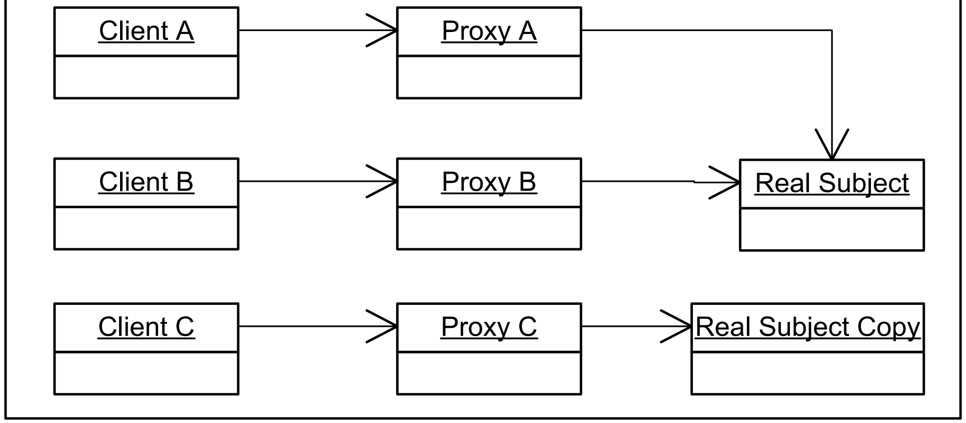
- Multiple clients share the same object as long as nobody tries to change it
- When a client attempts to change the object, they get their own private copy of the object
- Read-only clients continue to share the original object, while writers get their own copies
- Allows resource sharing, while making it look like everyone has their own object
- A String class could use this approach to optimize copying
- To make this work, clients are given proxies rather than direct references to the object
- When a write operation occurs, a proxy makes a private copy of the object on-the-fly to insulate other clients from the changes

Known Uses: Copy-on-Write

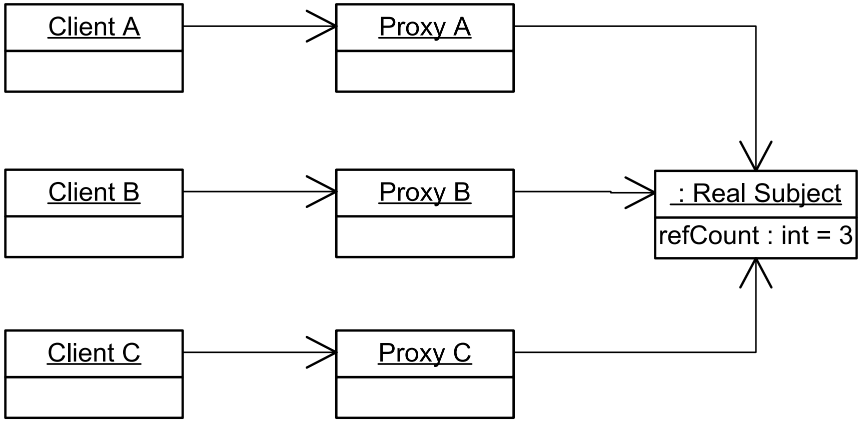
Before Write



After Write



Known Uses: Reference Counting



- Proxies maintain the reference count inside the object
- The last proxy to go away is responsible for deleting the object (i.e., when the reference count goes to 0, delete the object)

Proxy vs. Decorator

- The Proxy pattern is very similar in structure to the Decorator pattern. Both patterns create a “wrapper” around another object. (Adapter is a third kind of “wrapper”, but is clearly different from Decorator and Proxy because an adapter has a different interface than the wrapped object).

- Here are some differences between the two patterns:
  - The intents of the patterns are different. Decorator is used to add responsibilities to an object (without using inheritance). A Proxy is used to “control access” to an object. Rather than adding functionality, a Proxy might actually prevent access to functionality.
    - Read-only collections, Secure objects

  - Decorators are often organized into chains where each decorator adds a separate responsibility. Proxies are usually not organized into chains (although they could be).

  - A Decorator always stores a reference to the wrapped object; a Proxy may or may not, depending on what it does
    - Distributed objects (proxy never stores direct object reference)
    - Lazy Loading (proxy sometimes stores a direct object reference)