Proxy Design Pattern

Intent of the DP

This design pattern aims to:

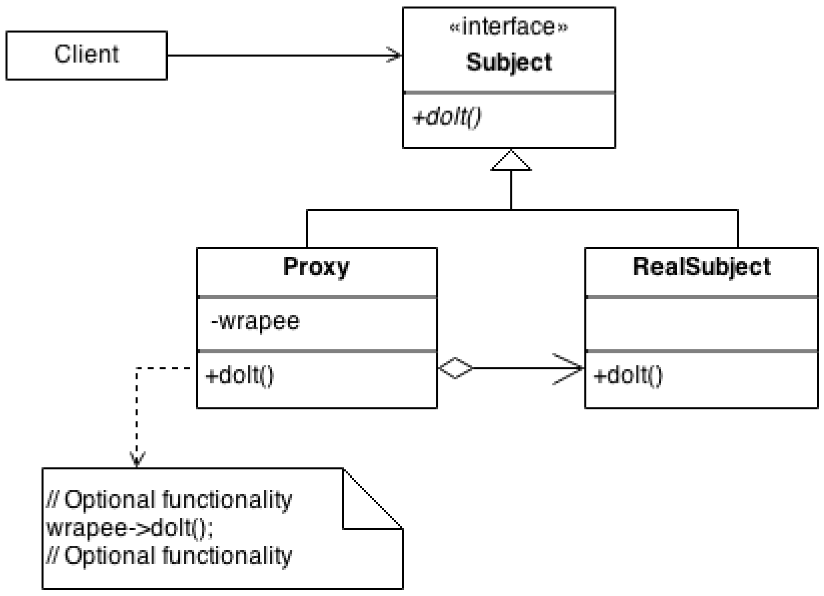
- Add a wrapper and delegation to protect the real component from undue complexity
- Provide a „Placeholder” for an object to control references to it
- Use an extra level of indirection to support distributed, controlled or intelligent access

Explanation

Links to sources:

- https://sourcemaking.com/design_patterns/proxy
- https://dzone.com/articles/design-patterns-proxy
- https://www.amazon.com/Spring-Design-Patterns-application-development/dp/1788299450
- https://www.tutorialspoint.com/design_pattern/proxy_pattern.htm
- http://www.baeldung.com/java-structural-design-patterns

The Proxy is known as a structual pattern, as it is used to form large object structures accrossmany disparate objects. A simple diagram showing a way of implementing the proxy:



The Proxy design pattern supports the following SOLID principles:

- Liskov substitution principle – because it provides access to the client through an interface, the instantiation of the real object can be changed to any object that provides implementation of that interface
- Interface segregation principle – the real object can do more than just a single functionality for the client, however, through the proxy we can provide to specific clients just the functionality required by the client, through multiple interfaces
- Dependency inversion principle – since the client has access to the real object through an interface, abstraction, this principle is met

Related Patterns

Decorator and Proxy have different purposes but similar structures. Both describe how to provide a level of indirection to another object, and the implementations keep a reference to the object to which they forward requests. Differences between these patterns are :

- Decorator focuses on dynamically adding functions to an object, while Proxy pattern focuses on controlling access to an object

- A Decorator is always passed its delegatee, while a Proxy might create it himself, or he might have it injected.
- Proxy provides the same interface, while Decorator provides an enhanced interface.

Common Situations for Use

- Virtual Proxies – delaying the creation and initialization of expensive objects until needed, where the objects are created on demand
- Remote Proxies – providing a local representation for an object that is in a different address space. A common example is Java RMI stub objects.
- Protection Proxies – where a proxy controls access to the methods of the real objects by giving access to some objects while denying access to others

Can be Mistaken for

The Proxy design pattern can be mistaken for the Decorator and Adapter design patterns, because they have similar structures, and at first glance they might be similar. However, the Proxy usually has information about the real subject at compile time itself, wereas, Decorator and Adapter get injected at runtime. Also, Adapter and Decorator alter and decorate the functionalities of pre-existing instances, while the Proxy provides the same interface as the object is is holding a reference to, and does not modify the data in any manner.

Student: Timis Adrian Ioan
Group: 30432