DZone

Proxy Pattern Tutorial with Java Examples

by James Sugrue MVB · Mar. 12, 10 · Java Zone

Today's pattern is the Proxy pattern, another simple but effective pattern that helps with controlling use and access of resources.

Proxy in the Real World

A Proxy can also be defined as a surrogate. In the real work a cheque or credit card is a proxy for what is in our bank account. It can be used in place of cash, which is what is needed, and provides a means of accessing that cash when required. And that's exactly what the Proxy pattern does - controls and manage access to the object they are "protecting".

Design Patterns Refcard
For a great overview of the most popular design patterns, DZone's Design Patterns Refcard is the best place to start.

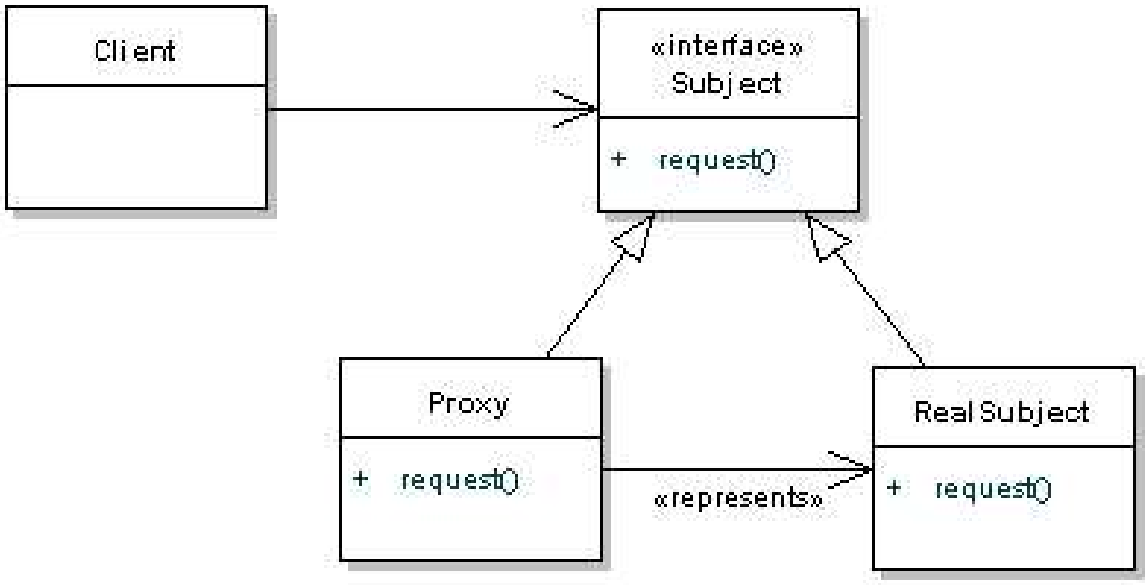
The Proxy Pattern

The Proxy is known as a **structural** pattern,as it's used to form large object structures across many disparate objects. Thedefinition of Proxy provided in the original Gang of Four book on DesignPatterns states:

Allows for object level access control by acting as a pass through entity or a placeholder object.

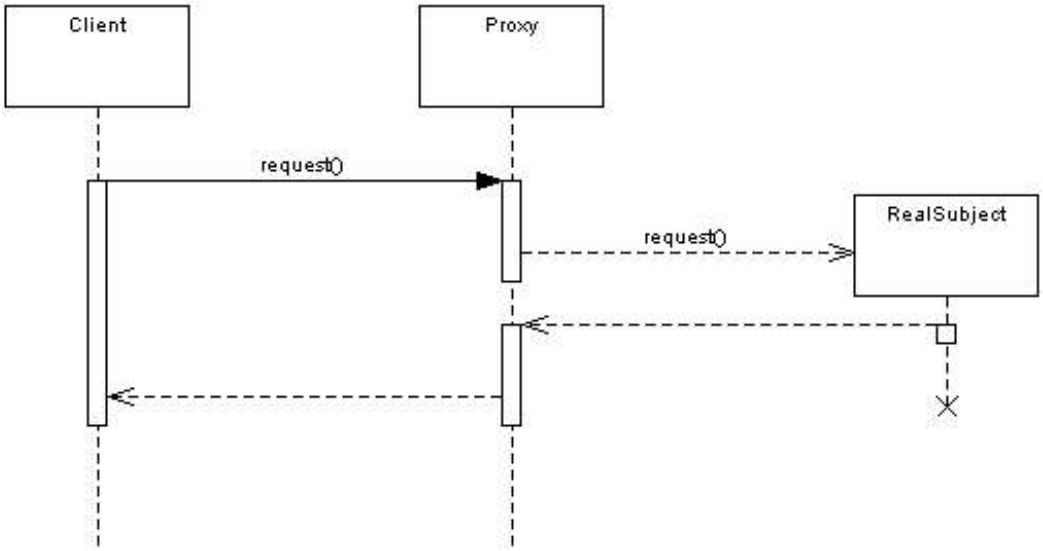
So it's quite a simple concept - to save on the amount of memory used, you might use a Proxy. Similarly, if you want to control access to an object, the pattern becomes useful.

Let's take a look at the diagram definition before we go into more detail.



As usual, when dealing with design patterns we code to interfaces. In this case, the interface that the client knows about is the **Subject**. Both the **Proxy** and **RealSubject** objects implement the **Subject** interface, but the client may not be able to access the RealSubject without going through the **Proxy**. It's quite common that the **Proxy** would handle the creation of the **RealSubject** object, but it will at least have a reference to it so that it can pass messages along.

Let's take a look at this in action with a sequence diagram.



As you can see it's quite simple - the Proxy is providing a barrier between the client and the real implementation.

There are many different flavours of Proxy, depending on it's purpose. You may have a protection proxy, to control access rights to an object. A virtual proxy handles the case where an object might be expensive to create, and a remote proxy controls access to a remote object.

You'll have noticed that this is very similar to the Adapter pattern. However, the main difference between bot is that the adapter will expose a different interface to allow interoperability. The Proxy exposes the same interface, but gets in the way to save processing time or memory.

Would I Use This Pattern?

This pattern is recommended when either of the following scenarios occur in your application:

- The object being represented is external to the system.
- Objects need to be created on demand.
- Access control for the original object is required
- Added functionality is required when an object is accessed.

Typically, you'll want to use a proxy when communication with a third party is an expensive operation, perhaps over a network. The proxy would allow you to hold your data until you are ready to commit, and can limit the amount of times that the communication is called.

The proxy is also useful if you want to decouple actual implementation code from the access to a particular library. Proxy is also useful for access to large files, or graphics. By using a proxy, you can delay loading the resource until you really need the data inside. Without the concept of proxies, an application could be slow, and appear non-responsive.

So How Does It Work In Java?

Let's continue with the idea of using a proxy for loading images. First, we should create a common interface for the real and proxy implementations to use:

```
1  public interface Image{  public void displayImage();}
```

The RealImage implementation of this interface works as you'd expect:

```
1  public class RealImage implements Image{   public RealImage(URL url)   {      //load up the i
```

Now the Proxy implementation can be written, which provides access to the RealImage class. Note that it's only when we call the displayImage() method that it actually uses the RealImage. Until then, we don't need the data.

```
1  public class ProxyImage implements Image{    private URL url;     public ProxyImage(URL url)
```

And it's really as simple as that. As far as the client is concerned, they will just deal with the interface.

Watch Out for the Downsides

Watch Out for the Downsides

Usually this is the stage that I point out the disadvantages to the pattern. Proxy is quite simple, and pragmatic, and it's one pattern that I can't think of any downsides for. Perhaps you know of some? If so, please share them in the comments section.

Next Up

The Decorator pattern is a close relation to the Proxy pattern, so we'll take a look at that next week.

Enjoy the Whole "Design Patterns Uncovered" Series:

Creational Patterns

- Learn The Abstract Factory Pattern
- Learn The Builder Pattern
- Learn The Factory Method Pattern
- Learn The Prototype Pattern
- Learn The Singleton Pattern

Structural Patterns

- Learn The Adapter Pattern
- Learn The Bridge Pattern
- Learn The Composite Pattern
- Learn The Decorator Pattern
- Learn The Facade Pattern
- Learn The Flyweight Pattern
- Learn The Proxy Pattern

Behavioral Patterns

- Learn The Chain of Responsibility Pattern
- Learn The Command Pattern
- Learn The Interpreter Pattern
- Learn The Iterator Pattern
- Learn The Mediator Pattern
- Learn The Memento Pattern
- Learn The Observer Pattern
- Learn The State Pattern
- Learn The Strategy Pattern

- Learn The Template Method Pattern
- Learn The Visitor Pattern

Like This Article? Read More From DZone

Observer Pattern Tutorial with Java Examples

Singleton Pattern Tutorial with Java Examples

Factory Method Pattern Tutorial with Java Examples

Free DZone Refcard
Getting Started With Scala

Topics: JAVA , DESIGN PATTERNS , PATTERNS , DESIGN PATTERNS UNCOVERED

Java Partner Resources