The Proxy Pattern

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

**Proxy Pattern: Motivation**

There are situations in which a client does not or can not reference an Object directly, but wants to still interact with the object.

A proxy object can act as the intermediary between the client and the target object.

The proxy object has the same interface as the target object.

The proxy holds a reference to the target object and can forward requests to the target as required.

Proxies are useful wherever there is a need for a more sophisticated reference to an object than a simple pointer or simple reference can provide.
The Proxy Pattern Defined

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

![Diagram showing the Proxy Pattern](image)

**Participants**

**Proxy:**
- maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same.
- provides an interface identical to Subject's so that a proxy can be substituted for the real subject.
- controls access to the real subject and may be responsible for creating and deleting it.

**Subject:** defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.

**RealSubject:** defines the real object that the proxy represents.
Different ways proxies control access

Proxy Pattern control and manage access.

The Proxy pattern can manifest itself in many different ways:

- A *Virtual Proxy* allows the creation of a memory intensive object on demand. The object will not be created until it is really needed.

- A *Protection Proxy* (firewall) controls access to a resource based on access rights.

- *Remote Proxy* controls access to a remote object - As we have seen in the case of RMI,

Proxy Pattern: Structural Code

```java
/**
 * Test driver for the pattern.
 */
public class Test {
    public static void main(String arg[])
    {
        Subject real = new RealSubject();
        Proxy proxy = new Proxy();
        proxy.setRealSubject(real);
        proxy.request();
    }
}

/**
 * Defines the common interface for RealSubject and Proxy so that
 * a Proxy can be used anywhere a RealSubject is expected.
 */
public interface Subject {
    void request();
}

/**
 * Defines the real object that the proxy represents.
 */
public class RealSubject implements Subject {
    public void request() {
        // Do something based on the interface.
    }
}
```
/* Maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same. Provides an interface identical to Subject's so that a proxy can be substituted for the real subject. */

class Proxy implements Subject {
    private Subject realSubject;
    public void setRealSubject( Subject subject ) {
        realSubject = subject;
    }
    public Subject getRealSubject() {
        return realSubject;
    }
    public void request() {
        realSubject.request();
    }
}

Virtual Proxy: Example

import java.util.*;

interface Image {
    public void displayImage();
}

class RealImage implements Image {
    private String filename;
    public RealImage(String filename) {
        this.filename = filename;
        System.out.println("Loading "+filename);
    }
    public void displayImage() { System.out.println("Displaying "+filename); }
}
Protection Proxy: Example

The real client stores an Account Number. Only the users who know the valid password can access this Account Number. Real client is protected by a proxy who knows the password. If a user wants to get an Account Number, first the proxy asks him/her to authenticate itself. If the user entered a correct password the proxy will call the real client and pass the Account Number to the user.

/// C# example
using System;
namespace ConsoleApplicationTest.FundamentalPatterns.ProtectionProxyPattern {
    public interface IClient{
        string GetAccountNo();
    }
    public class RealClient : IClient{
        private string accountNo="AB-111111";
        public RealClient(){
            Console.WriteLine("RealClient: Initialized");
        }
        public string GetAccountNo(){
            Console.WriteLine("RealClient's AccountNo: " + accountNo);
            return accountNo;
        }
    }
}
public class ProtectionProxy : IClient {
    private string password = null;  //password to get secret
    RealClient client = null;
    public ProtectionProxy(string pwd) {
        Console.WriteLine("ProtectionProxy: Initialized");
        password = pwd;
        client = new RealClient();
    }
    public String GetAccountNo() {
        Console.WriteLine("Password: ");
        string tmpPwd = Console.ReadLine();
        if (tmpPwd == password) {
            return client.GetAccountNo();
        } else {
            Console.WriteLine("ProtectionProxy: Illegal password!");
            return "";
        }
    }
}

class ProtectionProxyExample {
    public static void Main(string[] args) {
        IClient client = new ProtectionProxy("thePassword");
        Console.WriteLine("main received: "+client.GetAccountNo());
        Console.WriteLine("main received: "+client.GetAccountNo());
        Console.ReadLine();
    }
}

Proxy vs Decorator

- Proxy is structurally similar to Decorator, but the two differ in their purpose.

- The Decorator Pattern adds behavior to an object, while a proxy controls access to it.

- Like any wrapper, proxies will increase the number of classes and objects in your design.