

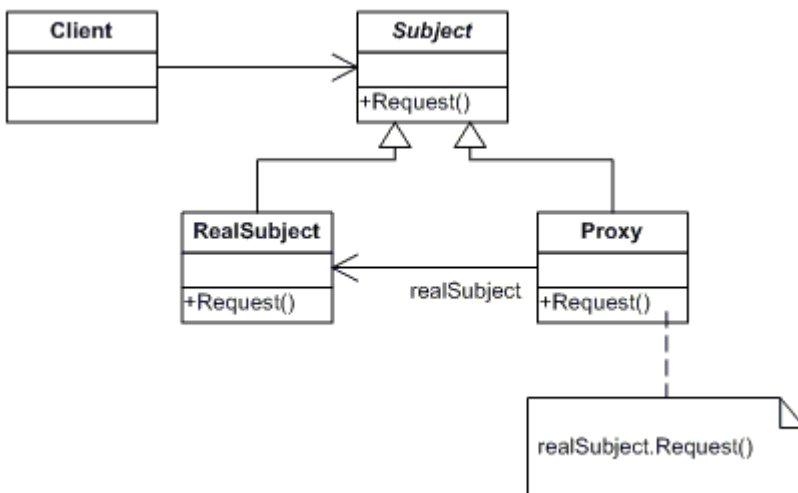
Design Patterns: C# (3)

- Proxy
 - Structural pattern

from: <http://www.dofactory.com>

Proxy

- Provide a substitute or placeholder for another object to control access to it.



participants

The classes and/or objects participating in this pattern are:

- **Proxy (MathProxy)**
 - 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.
 - other responsibilities depend on the kind of proxy:
 - *remote proxies* are responsible for encoding a request and its arguments and for sending the encoded request to the real subject in a different address space.
 - *virtual proxies* may cache additional information about the real subject so that they can postpone accessing it. For example, the ImageProxy from the Motivation caches the real image's extent.
 - *protection proxies* check that the caller has the access permissions required to perform a request.
- **Subject (IMath)**
 - defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
- **RealSubject (Math)**
 - defines the real object that the proxy represents.

C# source code:

```
using System;

namespace DoFactory.GangOfFour.Proxy.Structural
{
    /// <summary>
    /// MainApp startup class for Structural
    /// Proxy Design Pattern.
    /// </summary>
    class MainApp
    {
        /// <summary>
        /// Entry point into console application.
        /// </summary>
        static void Main()
        {
```

```
// Create proxy and request a service
Proxy proxy = new Proxy();
proxy.Request();

// Wait for user
Console.ReadKey();
}
}

/// <summary>
/// The 'Subject' abstract class
/// </summary>
abstract class Subject
{
    public abstract void Request();
}

/// <summary>
/// The 'RealSubject' class
/// </summary>
class RealSubject : Subject
{
    public override void Request()
    {
        Console.WriteLine("Called RealSubject.Request()");
    }
}

/// <summary>
/// The 'Proxy' class
/// </summary>
class Proxy : Subject
{
    private RealSubject _realSubject;
```

```
public override void Request()
{
    // Use 'lazy initialization'
    if (_realSubject == null)
    {
        _realSubject = new RealSubject();
    }

    _realSubject.Request();
}
}
```

C# source code (example)

```
using System;

namespace DoFactory.GangOfFour.Proxy.RealWorld
{
    /// <summary>
    /// MainApp startup class for Real-World
    /// Proxy Design Pattern.
    /// </summary>
    class MainApp
    {
        /// <summary>
        /// Entry point into console application.
        /// </summary>
        static void Main()
        {
            // Create math proxy
            MathProxy proxy = new MathProxy();

            // Do the math
        }
    }
}
```

```

        Console.WriteLine("4 + 2 = " + proxy.Add(4, 2));
        Console.WriteLine("4 - 2 = " + proxy.Sub(4, 2));
        Console.WriteLine("4 * 2 = " + proxy.Mul(4, 2));
        Console.WriteLine("4 / 2 = " + proxy.Div(4, 2));

        // Wait for user
        Console.ReadKey();
    }
}

/// <summary>
/// The 'Subject interface
/// </summary>
public interface IMath
{
    double Add(double x, double y);
    double Sub(double x, double y);
    double Mul(double x, double y);
    double Div(double x, double y);
}

/// <summary>
/// The 'RealSubject' class
/// </summary>
class Math : IMath
{
    public double Add(double x, double y) { return x + y; }
    public double Sub(double x, double y) { return x - y; }
    public double Mul(double x, double y) { return x * y; }
    public double Div(double x, double y) { return x / y; }
}

/// <summary>
/// The 'Proxy Object' class

```

```
/// </summary>
class MathProxy : IMath
{
    private Math _math = new Math();

    public double Add(double x, double y)
    {
        return _math.Add(x, y);
    }

    public double Sub(double x, double y)
    {
        return _math.Sub(x, y);
    }

    public double Mul(double x, double y)
    {
        return _math.Mul(x, y);
    }

    public double Div(double x, double y)
    {
        return _math.Div(x, y);
    }
}
```