Alright, let's design a simple system using the Proxy design pattern, and then provide an implementation in C#.

**UML**

For the sake of simplicity in this text format, I'll describe the UML in text.

**Classes**:

1. ISubject
   * This is an interface that defines the common operations.
   * Methods:
     + Request()
2. RealSubject (implements ISubject)
   * This is the real object that the proxy represents.
   * Methods:
     + Request()
3. Proxy (implements ISubject)
   * This class represents the proxy to the real object.
   * Attributes:
     + \_realSubject: RealSubject
   * Methods:
     + Request()

**Sequence:**

1. The client calls the Request() method on the Proxy.
2. If the Proxy hasn't already done so, it instantiates a RealSubject object.
3. The Proxy forwards the request to the RealSubject's Request() method.

**C# Implementation**

csharp

using System;

namespace ProxyPattern

{

// 1. ISubject Interface

public interface ISubject

{

void Request();

}

// 2. RealSubject Class

public class RealSubject : ISubject

{

public void Request()

{

Console.WriteLine("RealSubject: Handling the Request.");

}

}

// 3. Proxy Class

public class Proxy : ISubject

{

private RealSubject \_realSubject;

public void Request()

{

if (\_realSubject == null)

{

\_realSubject = new RealSubject();

}

Console.WriteLine("Proxy: Forwarding request to RealSubject.");

\_realSubject.Request();

}

}

// Client

class Program

{

static void Main(string[] args)

{

ISubject proxy = new Proxy();

proxy.Request();

}

}

}

When you run this, you'll get:

vbnet

Proxy: Forwarding request to RealSubject.

RealSubject: Handling the Request.

The proxy checks if the RealSubject has been created and creates it if it hasn't. It then forwards the request to the real subject. This design is useful for controlling access, logging, lazy instantiation, and many other use cases.