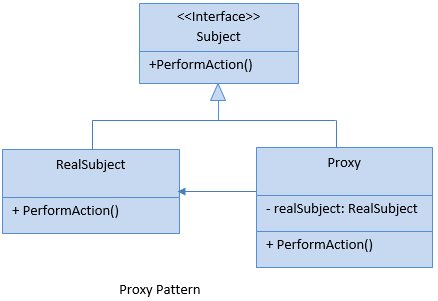
What is Proxy Pattern

The proxy design pattern is used to provide a surrogate object, which references to other object.

Proxy pattern involves a class, called proxy class, which represents functionality of another class.

Proxy Pattern - UML Diagram & Implementation

The UML class diagram for the implementation of the proxy design pattern is given below:



The classes, interfaces and objects in the above UML class diagram are as follows:

1. Subject

This is an interface having members that will be implemented by RealSubject and Proxy class.

1. RealSubject

This is a class which we want to use more efficiently by using proxy class.

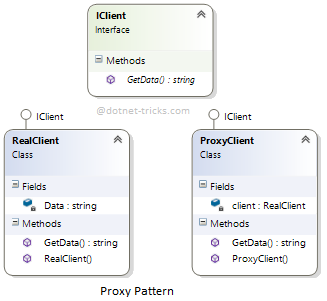
1. Proxy

This is a class which holds the instance of RealSubject class and can access RealSubject class members as required.

C# - Implementation Code

1. public interface Subject
2. {
3. void PerformAction();
4. }
6. public class RealSubject : Subject
7. {
8. public void PerformAction()
9. {
10. Console.WriteLine("RealSubject action performed.");
11. }
12. }
14. public class Proxy : Subject
15. {
16. private RealSubject \_realSubject;
18. public void PerformAction()
19. {
20. if (\_realSubject == null)
21. \_realSubject = new RealSubject();
23. \_realSubject.PerformAction();
24. }
25. }

Proxy Pattern - Example



Who is what?

The classes, interfaces and objects in the above class diagram can be identified as follows:

1. **IClient**- Subject Interface.
2. **RealClient** - RealSubject Class.
3. **ProxyClient** - Proxy Class.

C# - Sample Code

1. */// <summary>*
2. */// The 'Subject interface*
3. */// </summary>*
4. public interface IClient
5. {
6. string GetData();
7. }
9. */// <summary>*
10. */// The 'RealSubject' class*
11. */// </summary>*
12. public class RealClient : IClient
13. {
14. string Data;
15. public RealClient()
16. {
17. Console.WriteLine("Real Client: Initialized");
18. Data = "Dot Net Tricks";
19. }
21. public string GetData()
22. {
23. return Data;
24. }
25. }
27. */// <summary>*
28. */// The 'Proxy Object' class*
29. */// </summary>*
30. public class ProxyClient : IClient
31. {
32. RealClient client = new RealClient();
33. public ProxyClient()
34. {
35. Console.WriteLine("ProxyClient: Initialized");
36. }
38. public string GetData()
39. {
40. return client.GetData();
41. }
42. }
44. */// <summary>*
45. */// Proxy Pattern Demo*
46. */// </summary>*
47. class Program
48. {
49. static void Main(string[] args)
50. {
51. ProxyClient proxy = new ProxyClient();
52. Console.WriteLine("Data from Proxy Client = {0}", proxy.GetData());
54. Console.ReadKey();
55. }
56. }

Proxy Pattern Demo - Output

https://dotnettricks.blob.core.windows.net/img/designpatterns/proxy2.png

There are various kinds of proxies, some of them are as follows:

1. **Virtual proxies** : Hand over the creation of an object to another object
2. **Authentication proxies** : Checks the access permissions for a request
3. **Remote proxies** : Encodes requests and send them across a network
4. **Smart proxies** : Change requests before sending them across a network

When to use it?

1. Objects need to be created on demand means when their operations are requested.
2. Access control for the original object is required.
3. Allow to access a remote object by using a local object(it will refer to a remote object).