**Imagine a Real-World Scenario:**

Think of a high-security vault that holds sensitive information. Access to this vault requires a special key. Now, instead of giving everyone access to the vault directly, the vault owners create a "Proxy Guard." This guard stands between people and the vault. When someone wants to access the vault, they approach the guard with their key. The guard checks the key and grants access only if the key is correct. The guard also logs who accessed the vault.

**Subject Interface (IResource):**

* This is like the rules the guard follows when people approach them.
* It defines the actions that the real resource (vault) and the proxy (guard) should be able to do.

**Real Subject (RealResource):**

* This is like the actual vault holding the sensitive data.
* It follows the rules from the IResource interface.
* It provides the core functionality – granting access to the vault.

**Proxy (SecureResourceProxy):**

* This is like the guard that stands between people and the vault.
* It also follows the rules from the IResource interface.
* It checks the access key, and if it's correct, it grants access to the real vault. Otherwise, it denies access.
* It also keeps track of who accessed the vault.

**Client Code (Main Method):**

* This is where everything comes together.
* You create a proxy (guard) instance that controls access to a resource (vault).
* You try to access the resource with both correct and incorrect access keys.
* The proxy decides whether you get access or not.

**In Simple English:**

The Proxy pattern is like having a security guard between you and a vault. You want to access the vault, so you approach the guard. The guard checks your key – if it's the right key, the guard lets you access the vault. If it's the wrong key, the guard denies access. The guard also keeps track of who accessed the vault. Similarly, in your code, the proxy controls access to a resource. It checks if you have the right "key" (access) before letting you use the real resource. This pattern is useful for scenarios where you want to control access, add logging, or enhance functionality without changing the core object's behavior.

**Step 1: Define the Subject Interface (IResource)**

The IResource interface defines the contract for the real resource and its proxy. It includes a method to access the resource.

public interface IResource

{

void AccessResource();

}

**Step 2: Implement the Real Subject (RealResource)**

The RealResource class implements the IResource interface. It represents the actual resource that the proxy will control access to.

public class RealResource : IResource

{

**// ...**

public void AccessResource()

{

Console.WriteLine($"Accessing real resource: {\_resourceName}");

}

}

**Step 3: Implement the Proxy (SecureResourceProxy)**

The SecureResourceProxy class also implements the IResource interface. It acts as a protective barrier around the real resource, controlling access and adding extra functionality.

public class SecureResourceProxy : IResource

{

**// ...**

public void AccessResource()

{

if (CheckAccess(\_accessKey))

{

**// If authorized, create the real resource (if not already) and access it**

if (\_realResource == null)

{

\_realResource = new RealResource(\_resourceName);

}

\_realResource.AccessResource();

}

else

{

Console.WriteLine("Access denied.");

}

}

private bool CheckAccess(string accessKey)

{

**// Simulate access control logic**

return accessKey == "secret";

}

}

**Step 4: Client Code (Main Method)**

In the Main method, you create an instance of the SecureResourceProxy. You then try to access the resource with both the correct and incorrect access keys.

static void Main(string[] args)

{

IResource secureResource = new SecureResourceProxy("top-secret-data.txt", "secret");

**// Accessing with correct access key**

secureResource.AccessResource();

**// Accessing with incorrect access key**

IResource unauthorizedResource = new SecureResourceProxy("top-secret-data.txt", "wrong\_key");

unauthorizedResource.AccessResource();

}

**Explanation:**

**Proxy Pattern:** The Proxy pattern provides a surrogate or placeholder for another object to control its access or add additional functionality.

**Step 1:** The IResource interface defines the subject's contract, which both the real resource (RealResource) and the proxy (SecureResourceProxy) implement.

**Step 2:** The RealResource class represents the actual resource that will be accessed and provides the core functionality.

**Step 3:** The SecureResourceProxy class acts as a protective proxy, controlling access to the real resource and adding access control logic.

**Step 4:** In the Main method, you create instances of the proxy (SecureResourceProxy). You try accessing the resource with both the correct and incorrect access keys. This demonstrates the Proxy pattern, where the proxy controls access to the real subject while adding additional features, such as access control in this case.

By using the Proxy pattern, your code adds a layer of control and functionality to the access of the real resource. It's useful for scenarios where you need to restrict access, add logging, caching, or any other behavior without altering the core functionality of the real object.