Decorator Pattern

Existing Abstraction — A — PREREQUISITE

```
public interface IModem
{
    void Send(byte[] data);
}
```

Original Implementation — B — PREREQUISITE

```
public class Modem : IModem
{
    public void Send(byte[] data)
    {
        // Sends data over the connection
    }
}
```

Implement Abstraction — 1

```
public class LoggingModem : IModem
{
    private IModem modem;
    public LoggingModem(IModem modem)
    {
        this.modem = modem;
    }
    public void Send(byte[] data)
    {
        Debug.WriteLine(Encoding.UTF8.GetString(data));
        this.modem.Send(data);
    }
}
```

Store Original locally — 2
(You can achieve the same by using a public setter)

Add new behavior and delegate original call — 3

ii

i

PURPOSE

¬ Alternative to subclassing.
¬ Adding responsibilities dynamically.

BENEFITS

i Adding extra behavior without changing the original class (OCP).
ii Add an extra class for new responsibilities (SRP).
iii It is easier to test than a derived class as it's easier to isolate.

```
[TestClass]
public class LoggingModemTest
{
    [TestMethod]
    public void Send_ValidInput_SendIsCalled()
    {
        var modem = new ModemStub();
        var target = new LoggingModem(modem);
        var input = new byte[] { 0x00, 0x01, 0x02 };

        target.Send(input);

        Assert.IsTrue(modem.SendIsCalled);
    }

    internal class ModemStub : IModem
    {
        public bool SendIsCalled { get; private set; }
        public void Send(byte[] data)
        {
            this.SendIsCalled = true;
        }
    }
}
```

Arrange — Act — Assert

iii