1. Write Testtable code with Moq

```
public interface ICalculatorService
{
    int Add(int a, int b);
    int Subtract(int a, int b);
}
public class MathProcessor
{
    private readonly ICalculatorService _calculatorService;

    public MathProcessor(ICalculatorService calculatorService)
    {
        _calculatorService = calculatorService;
    }

    public int AddAndDouble(int a, int b)
    {
        int sum = _calculatorService.Add(a, b);
        return sum * 2;
    }

    public int SubtractAndSquare(int a, int b)
    {
        int diff = _calculatorService.Subtract(a, b);
        return diff * diff;
    }
}

using Moq;
using NUnit.Framework;

[TestFixture]
public class MathProcessorTests
{
    private Mock<ICalculatorService> _mockCalculator;
    private MathProcessor _mathProcessor;

    [SetUp]
    public void Setup()
    {
        _mockCalculator = new Mock<ICalculatorService>();
        _mathProcessor = new MathProcessor(_mockCalculator.Object);
    }

    [Test]
    public void AddAndDouble_ReturnsCorrectResult()
```

```
    {
        // Arrange
        _mockCalculator.Setup(x => x.Add(2, 3)).Returns(5);

        // Act
        var result = _mathProcessor.AddAndDouble(2, 3);

        // Assert
        Assert.AreEqual(10, result);
        _mockCalculator.Verify(x => x.Add(2, 3), Times.Once);
    }

    [Test]
    public void SubtractAndSquare_ReturnsCorrectResult()
    {
        // Arrange
        _mockCalculator.Setup(x => x.Subtract(5, 3)).Returns(2);

        // Act
        var result = _mathProcessor.SubtractAndSquare(5, 3);

        // Assert
        Assert.AreEqual(4, result);
        _mockCalculator.Verify(x => x.Subtract(5, 3), Times.Once);
    }
}
```