Unit Testing with Moq – Mocking External Dependencies

# I Learned This

In this exercise, I learned how to write unit tests using NUnit and Moq to isolate and mock external dependencies.  
The scenario involved sending email through a `MailSender` class. Instead of triggering real emails, I used mocking   
to simulate the behavior of the mail server, thus ensuring the test runs quickly and reliably.

# Objectives Covered

• Understand the meaning and use of mocking in Unit Testing.  
  
• Use Dependency Injection (DI) to create testable classes.  
  
• Create mock objects for classes that depend on external systems like SMTP.  
  
• Use NUnit `[TestFixture]`, `[OneTimeSetUp]`, `[TestCase]` attributes.  
  
• Write unit tests using Moq to simulate dependency behavior.

## MailSender.cs (Library Code)

• Defines `IMailSender` interface and `MailSender` class that sends an email.  
  
• Contains `CustomerComm` class which is the testable unit, depending on IMailSender.

## CustomerCommTests.cs (Test Code)

• Uses Moq to create a mock of IMailSender.  
  
• Injects the mock into CustomerComm.  
  
• Verifies that `SendMailToCustomer()` returns `true` using `Assert.That`.

## Library Code

using System.Net;

using System.Net.Mail;

namespace CustomerCommLib

{

    public interface IMailSender

    {

        bool SendMail(string toAddress, string message);

    }

    public class MailSender : IMailSender

    {

        public bool SendMail(string toAddress, string message)

        {

            return true; // Just for build, no actual email

        }

    }

    public class CustomerComm

    {

        IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

            return \_mailSender.SendMail("cust123@abc.com", "Some Message");

        }

    }

}

## Test Code

using NUnit.Framework;

using Moq;

using CustomerCommLib; //

namespace CustomerComm.Tests //

{

    [TestFixture]

    public class CustomerCommTests

    {

        private CustomerCommLib.CustomerComm \_customerComm;

        private Mock<IMailSender> \_mockMailSender;

        [OneTimeSetUp]

        public void Setup()

        {

            \_mockMailSender = new Mock<IMailSender>();

            \_mockMailSender

                .Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>()))

                .Returns(true);

            \_customerComm = new CustomerCommLib.CustomerComm(\_mockMailSender.Object);

        }

        [TestCase]

        public void SendMailToCustomer\_ShouldReturnTrue()

        {

            bool result = \_customerComm.SendMailToCustomer();

            Assert.That(result, Is.True);

        }

    }

}

# Output

