Moq Handson

---

1. What is Mocking in Unit Testing?

Mocking is creating fake objects that act like real ones.
For example, instead of using a real email service in your test, you create a mock that pretends to send emails.
This helps test your code without depending on real systems like databases or networks.

---

2. Why is Mocking Useful for Test-Driven Development (TDD)?

In TDD, we write tests before writing the actual code.
Mocking helps because:

- You can test even if real systems (like database or email) are not ready.
- Your tests run faster.
- You only test one part of the code at a time.

---

3. How to Use Dependency Injection (DI) to Make Code Testable

Dependency Injection means passing required objects from outside, instead of creating them inside the class.
This makes code easier to test.

Example:
You can pass a mock email sender to your class during testing.

---

4. How to Use Moq to Mock External Dependencies

Moq is a tool used to create mock objects in C#.
Steps:

1. Install Moq using NuGet.

2. Create a mock object using `new Mock<IMailSender>()`.

3. Set what the mock should return using `.Setup()`.

4. Pass the mock to your class.

This lets you test your logic without using real services.

---

5. How to Write Unit Tests Using NUnit

NUnit is a testing framework used in C#.
Steps:

1. Install NUnit and NUnit Adapter using NuGet.

2. Write test classes using `[TestFixture]`.

3. Write test methods using `[Test]`.

4. Use `Assert` to check if the output is correct.

Example:
`Assert.IsTrue(result);` checks that `result` is true.

TASK 1 – Write Testable Code with Moq

🔧 Step 1: Create the Mail Sending Project

- Open Visual Studio

- Click on "Create a new project"

- Choose "Class Library (.NET Core)" → Click Next

- Name it `CustomerCommLib`

- Click "Create"

Step 2: Create Interface for Email Sender

- In the Solution Explorer, right-click on the project
- Choose "Add" → "Class"
- Name it `IMailSender.cs`
- Paste the following code:

```
namespace CustomerCommLib
{
    public interface IMailSender
    {
        bool SendMail(string toAddress, string message);
    }
}
```

Step 3: Create the Real Mail Sender

- Rename the default `Class1.cs` to `MailSender.cs`
- Replace the code with the following:

```
using System.Net;
using System.Net.Mail;

namespace CustomerCommLib
{
    public class MailSender : IMailSender
    {
        public bool SendMail(string toAddress, string message)
        {
            MailMessage mail = new MailMessage();
            SmtpClient smtp = new SmtpClient("smtp.gmail.com")
            {
                Port = 587,
                Credentials = new NetworkCredential("username",
"password"),
                EnableSsl = true
            };

            mail.From = new
MailAddress("your_email_address@gmail.com");
            mail.To.Add(toAddress);
            mail.Subject = "Test Mail";
            mail.Body = message;

            smtp.Send(mail);
            return true;
        }
    }
}
```

Note: This class sends real emails, so we won't actually test it directly.

Step 4: Create the Main Logic Class

- Right-click on the project → Add → Class → Name it `CustomerComm.cs`
- Paste this code:

```
namespace CustomerCommLib
{
    public class CustomerComm
    {
        private readonly IMailSender _mailSender;
        public CustomerComm(IMailSender mailSender)
        {
            _mailSender = mailSender;
        }

        public bool SendMailToCustomer()
        {
            return _mailSender.SendMail("cust123@abc.com", "Some
Message");
        }
    }
}
```

This class uses Dependency Injection to receive the mail sender, making it testable.

Step 5: Build the Project

- Go to the "Build" menu → Click "Build Solution"
- Make sure there are no errors

TASK 2 – Create Unit Test with Moq and NUnit

Step 1: Create the Test Project

- In Visual Studio, go to File → Add → New Project
- Select "Class Library (.NET Core)"
- Name it `CustomerComm.Tests`
- Click "Create"

Step 2: Add NuGet Packages

- Right-click on `CustomerComm.Tests` → "Manage NuGet Packages"
- Search for and install:
  - `NUnit`
  - `NUnit3TestAdapter`
  - `Microsoft.NET.Test.Sdk`
  - `Moq`

Step 3: Add Reference to Main Project

- Right-click `CustomerComm.Tests` → Add → Project Reference
- Check and select `CustomerCommLib` → Click OK

Step 4: Write the Unit Test Code

- Delete `Class1.cs`
- Add a new class → Name it `CustomerCommTests.cs`
- Paste this code:

```
using Moq;
using NUnit.Framework;
using CustomerCommLib;

namespace CustomerComm.Tests
{
    [TestFixture]
    public class CustomerCommTests
    {
        private Mock<IMailSender>? _mockMailSender;
        private CustomerCommLib.CustomerComm? _customerComm;

        [OneTimeSetUp]
        public void Setup()
        {
            _mockMailSender = new Mock<IMailSender>();
            _mockMailSender.Setup(ms =>
ms.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);
            _customerComm = new
CustomerCommLib.CustomerComm(_mockMailSender.Object);
        }

        [Test]
        public void SendMailToCustomer_ShouldReturnTrue()
        {
            bool result = _customerComm!.SendMailToCustomer();
            Assert.IsTrue(result);
        }
    }
}
```

What's Happening in the Test Code?

- We use Moq to create a mock object for `IMailSender`
- We tell the mock to always return `true` when `SendMail()` is called
- We inject the mock into the `CustomerComm` class
- We run the test to make sure `SendMailToCustomer()` returns `true`
- No real email is sent

Step 5: Run the Test

- Go to "Test" → "Test Explorer"
- Click "Run All"