DotNet-FSE Mandatory Hands-On

WEEK-2 **NAME:** Sri Ranjani Priya P

TOPIC :- NUnit

EXERCISE 1:

Write Testable Code with Moq

CODE:(C#)

File: ICalculatorService.cs

```
namespace CalculatorApp
{
    public interface ICalculatorService
    {
        int Add(int a, int b);
        int Subtract(int a, int b);
    }
}
```

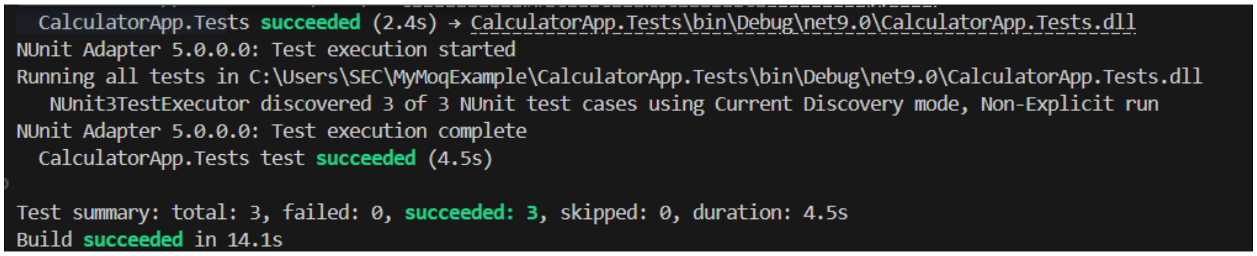
File: Calculator.cs

```
namespace CalculatorApp
{
    public class Calculator
    {
        private readonly ICalculatorService _service;

        public Calculator(ICalculatorService service)
        {
            _service = service;
        }

        public int AddNumbers(int a, int b) => _service.Add(a, b);

        public int SubtractNumbers(int a, int b) =>
_service.Subtract(a, b);
    }
}
```

File: CalculatorTests.cs

```csharp
using NUnit.Framework;
using Moq;
using CalculatorApp;

namespace CalculatorApp.Tests
{
    public class CalculatorTests
    {
        [Test]
        public void AddNumbers_ReturnsCorrectValue()
        {
            var mock = new Mock<ICalculatorService>();
            mock.Setup(s => s.Add(2, 3)).Returns(5);

            var calc = new Calculator(mock.Object);
            var result = calc.AddNumbers(2, 3);

            Assert.That(result, Is.EqualTo(5));
        }

        [Test]
        public void SubtractNumbers_ReturnsCorrectValue()
        {
            var mock = new Mock<ICalculatorService>();
            mock.Setup(s => s.Subtract(5, 3)).Returns(2);

            var calc = new Calculator(mock.Object);
            var result = calc.SubtractNumbers(5, 3);

            Assert.That(result, Is.EqualTo(2));
        }
    }
}
```

Output:



```
  CalculatorApp.Tests succeeded (2.4s) → CalculatorApp.Tests\bin\Debug\net9.0\CalculatorApp.Tests.dll
NUnit Adapter 5.0.0.0: Test execution started
Running all tests in C:\Users\SEC\MyMoqExample\CalculatorApp.Tests\bin\Debug\net9.0\CalculatorApp.Tests.dll
   NUnit3TestExecutor discovered 3 of 3 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  CalculatorApp.Tests test succeeded (4.5s)

Test summary: total: 3, failed: 0, succeeded: 3, skipped: 0, duration: 4.5s
Build succeeded in 14.1s
```

EXERCISE 2:

Create a unit test project using NUnit for the given **ConverterLib** project.

CODE:(C#)

FILE: IDollarToEuroExchangeRateFeed.cs

```csharp
namespace ConverterLib
{
    public interface IDollarToEuroExchangeRateFeed
    {
        double GetActualUSDToEuroRate();
    }
}
```

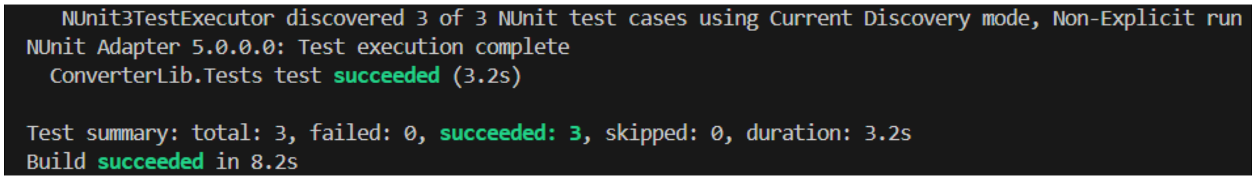
FILE: Converter.cs

```csharp
namespace ConverterLib{
    public class Converter
    {
        private readonly IDollarToEuroExchangeRateFeed _feed;

        public Converter(IDollarToEuroExchangeRateFeed feed)
        {
            _feed = feed;
        }

        public double USDToEuro(double amount)
        {
            double rate = _feed.GetActualUSDToEuroRate();
            return amount * rate;
        }
    }}
```

FILE: ConverterTests.cs

```
using NUnit.Framework;
using Moq;
using ConverterLib;

namespace ConverterLib.Tests
{
    public class ConverterTests
    {
        [Test]
        public void USDToEuro_ValidRate_ReturnsCorrectValue()
        {
            var mockFeed = new
Mock<IDollarToEuroExchangeRateFeed>();
            mockFeed.Setup(feed =>
feed.GetActualUSDToEuroRate()).Returns(0.85);

            var converter = new Converter(mockFeed.Object);
            var result = converter.USDToEuro(100);

            Assert.That(result, Is.EqualTo(85.0));
        }

        [Test]
        public void USDToEuro_ZeroRate_ReturnsZero()
        {
            var mockFeed = new
Mock<IDollarToEuroExchangeRateFeed>();
            mockFeed.Setup(feed =>
feed.GetActualUSDToEuroRate()).Returns(0.0);

            var converter = new Converter(mockFeed.Object);
            var result = converter.USDToEuro(100);

            Assert.That(result, Is.EqualTo(0.0));
        }
    }
}
```

OUTPUT:



```
   NUnit3TestExecutor discovered 3 of 3 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  ConverterLib.Tests test succeeded (3.2s)

Test summary: total: 3, failed: 0, succeeded: 3, skipped: 0, duration: 3.2s
Build succeeded in 8.2s
```

EXERCISE 3:

Create a class library in C# that sends emails to users upon a transaction using SmtpClient. Since real email sending cannot be unit tested directly, refactor the code by defining an interface IMailSender and inject it into the email service class. Then, write a unit test using NUnit and Moq to verify the send mail functionality without sending an actual email.

CODE:(C#)

FILE: IMailSender.cs

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

FILE: MailSender.cs

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

namespace EmailServiceLib
{
    public class MailSender : IMailSender
    {
        public bool SendMail(string toAddress, string message)
        {
            MailMessage mail = new MailMessage();
            SmtpClient smtpServer = new
SmtpClient("smtp.gmail.com");
```

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

            smtpServer.Port = 587;
            smtpServer.Credentials = new
NetworkCredential("username", "password");
            smtpServer.EnableSsl = true;
            smtpServer.Send(mail);

            return true;
        }
    }
}
```

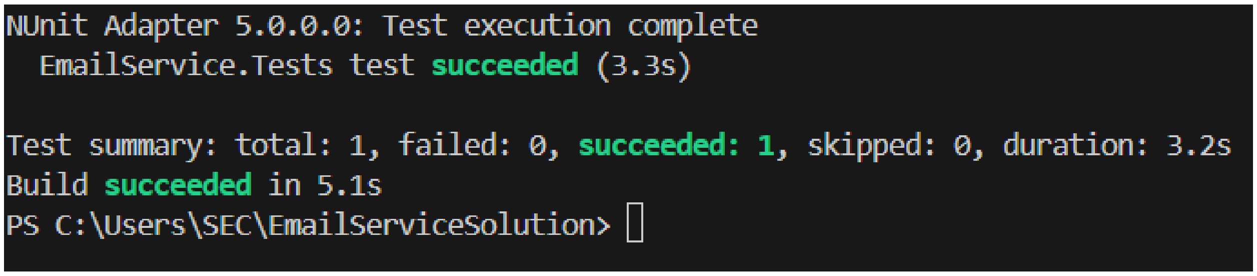
FILE: EmailService.cs

```
namespace EmailServiceLib
{
    public class CustomerMailService   // 🔄 Changed from
EmailService
    {
        private IMailSender _mailSender;

        public CustomerMailService(IMailSender mailSender)
        {
            _mailSender = mailSender;
        }

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

FILE: EmailServiceTests.cs

```
using NUnit.Framework;
using Moq;
using EmailServiceLib;

namespace EmailService.Tests
{
    [TestFixture]
    public class EmailServiceTests
    {
        private Mock<IMailSender> _mockSender = null!;
        private CustomerMailService _emailService = null!;

        [SetUp]
        public void Setup()
        {
            _mockSender = new Mock<IMailSender>();
            _mockSender.Setup(x => x.SendMail(It.IsAny<string>(),
It.IsAny<string>()))
                       .Returns(true);

            _emailService = new
CustomerMailService(_mockSender.Object);
        }

        [Test]
        public void SendCustomerMail_ShouldReturnTrue()
        {
            var result = _emailService.SendCustomerMail();
            Assert.That(result, Is.True);
        }     }
}
```

OUTPUT:



```
NUnit Adapter 5.0.0.0: Test execution complete
  EmailService.Tests test succeeded (3.3s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 3.2s
Build succeeded in 5.1s
PS C:\Users\SEC\EmailServiceSolution> 
```

EXERCISE 4:

Create a class library in C# to manage player information by interacting with a database. Since actual database operations cannot be unit tested directly, define an interface IPlayerMapper to abstract database access. Implement dependency injection to allow mocking. Then, write a unit test using NUnit and Moq to test the player registration logic without connecting to a real database.

CODE:(C#)

FILE: IPlayerMapper.csPlayer.cs

```
namespace PlayersManagerLib
{
    public interface IPlayerMapper
    {
        bool IsPlayerNameExistsInDb(string name);
        void AddNewPlayerIntoDb(string name);
    }
}
```

FILE: Player.cs

```
using System;

namespace PlayersManagerLib
{
    public class Player
    {
        public string Name { get; private set; }
        public int Age { get; private set; }
        public string Country { get; private set; }
        public int NoOfMatches { get; private set; }

        public Player(string name, int age, string country, int
noOfMatches)
        {
            Name = name;
            Age = age;
            Country = country;
            NoOfMatches = noOfMatches;
        }

        public static Player RegisterNewPlayer(string name,
IPlayerMapper playerMapper = null)
        {
            if (playerMapper == null)
```

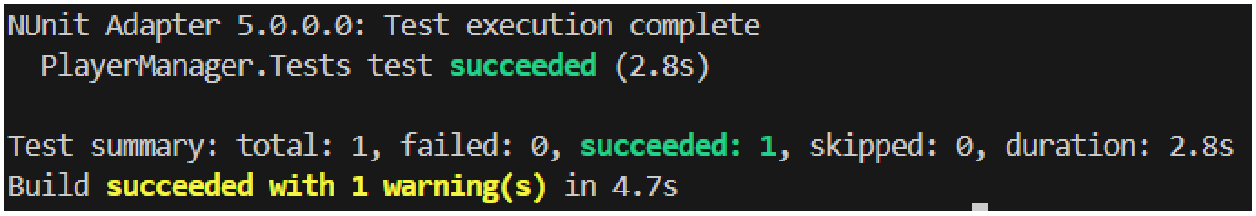
```
                throw new
ArgumentNullException(nameof(playerMapper));

            if (string.IsNullOrWhiteSpace(name))
                throw new ArgumentException("Player name can’t be
empty.");

            if (playerMapper.IsPlayerNameExistsInDb(name))
                throw new ArgumentException("Player name already
exists.");

            playerMapper.AddNewPlayerIntoDb(name);
            return new Player(name, 23, "India", 30);
        }
    }
}
```

FILE: PlayerTests.cs

```
using NUnit.Framework;
using Moq;
using PlayersManagerLib;

namespace PlayerManager.Tests
{
    [TestFixture]
    public class PlayerTests
    {
        private Mock<IPlayerMapper> _mockMapper = null!;

        [SetUp]
        public void Setup()
        {
            _mockMapper = new Mock<IPlayerMapper>();


            _mockMapper.Setup(x =>
x.IsPlayerNameExistsInDb(It.IsAny<string>())).Returns(false);
```

```
            _mockMapper.Setup(x =>
x.AddNewPlayerIntoDb(It.IsAny<string>()));
        }

        [Test]
        public void
RegisterNewPlayer_ShouldReturnPlayerWithCorrectAttributes()
        {

            var player = Player.RegisterNewPlayer("Ranjani",
_mockMapper.Object);


            Assert.That(player.Name, Is.EqualTo("Ranjani"));
            Assert.That(player.Age, Is.EqualTo(23));
            Assert.That(player.Country, Is.EqualTo("India"));
        }
    }
}
```

OUTPUT:



```
NUnit Adapter 5.0.0.0: Test execution complete
  PlayerManager.Tests test succeeded (2.8s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 2.8s
Build succeeded with 1 warning(s) in 4.7s
```

EXERCISE 5:

Create a class library in C# that retrieves a list of files from a directory. Since Directory.GetFiles() is static and cannot be directly unit tested, refactor the code using an interface. Then, write a unit test using NUnit and Moq to test the file retrieval logic without accessing the actual file system.

CODE:(C#)

FILE: IDirectoryExplorer.cs

```
using System.Collections.Generic;

```

```
namespace MagicFilesLib
{
    public interface IDirectoryExplorer
    {
        ICollection<string> GetFiles(string path);
    }
}
```

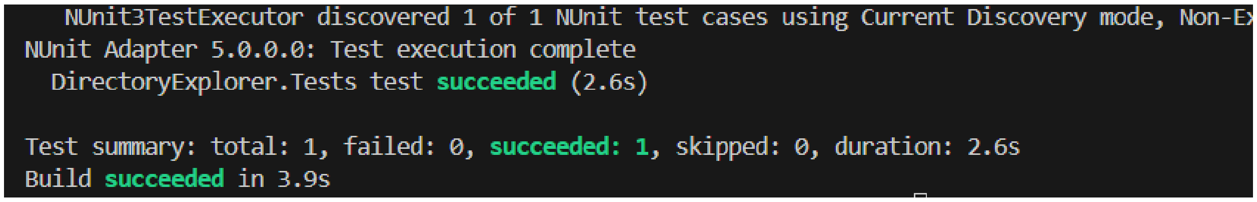
FILE: DirectoryExplorer.cs

```
using System.IO;
using System.Collections.Generic;

namespace MagicFilesLib
{
    public class DirectoryExplorer : IDirectoryExplorer
    {
        public ICollection<string> GetFiles(string path)
        {
            return Directory.GetFiles(path);
        }
    }
}
```

FILE: DirectoryExplorerTests.cs

```
using NUnit.Framework;
using Moq;
using MagicFilesLib;
using System.Collections.Generic;

namespace DirectoryExplorer.Tests
{
    [TestFixture]
    public class DirectoryExplorerTests
    {
        private Mock<IDirectoryExplorer> _mockExplorer = null!;
        private readonly string _file1 = "file.txt";
        private readonly string _file2 = "file2.txt";

        [SetUp]
        public void Setup()
        {
            _mockExplorer = new Mock<IDirectoryExplorer>();
            _mockExplorer.Setup(x => x.GetFiles(It.IsAny<string>()))
                         .Returns(new List<string> { _file1, _file2
});
        }

        [Test]
        public void ShouldReturnListOfFiles()
        {
            var result = _mockExplorer.Object.GetFiles("anypath");

            Assert.That(result, Is.Not.Null);
            Assert.That(result.Count, Is.EqualTo(2));
            Assert.That(result, Does.Contain(_file1));
        }
    }
}
```

OUTPUT:



```
   NUnit3TestExecutor discovered 1 of 1 NUnit test cases using Current Discovery mode, Non-Ex
NUnit Adapter 5.0.0.0: Test execution complete
  DirectoryExplorer.Tests test succeeded (2.6s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 2.6s
Build succeeded in 3.9s
```