**NUNIT AND MOQ**

**1. Moq-Handson**

## **1. Write Testable Code with Moq**

### **Exercise 1: MailSender – Interface & Implementation**

// Project: CustomerCommLib

using System.Net;

using System.Net.Mail;

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

public class MailSender : IMailSender

{

public bool SendMail(string toAddress, string message)

{

MailMessage mail = new MailMessage();

SmtpClient smtpServer = new SmtpClient("smtp.gmail.com")

{

Port = 587,

Credentials = new NetworkCredential("thasleem.peer@example.com", "password"),

EnableSsl = true

};

mail.From = new MailAddress("thasleem.peer@example.com");

mail.To.Add(toAddress);

mail.Subject = "Test Mail";

mail.Body = message;

smtpServer.Send(mail);

return true;

}

}

### **Exercise 2: CustomerComm – Class Under Test**

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

return \_mailSender.SendMail("customer.123@abc.com", "Some Message");

}

}

## **2. Mock File Object for Unit Tests**

### **Exercise 1: DirectoryExplorer – Interface & Implementation**

// Project: MagicFilesLib

using System.Collections.Generic;

using System.IO;

public interface IDirectoryExplorer

{

ICollection<string> GetFiles(string path);

}

public class DirectoryExplorer : IDirectoryExplorer

{

public ICollection<string> GetFiles(string path)

{

return Directory.GetFiles(path);

}

}

## **3. Mock Database for Unit Tests**

### **Exercise 1: PlayerMapper – Interface & Implementation**

// Project: PlayersManagerLib

using System.Data.SqlClient;

public interface IPlayerMapper

{

bool IsPlayerNameExistsInDb(string name);

void AddNewPlayerIntoDb(string name);

}

public class PlayerMapper : IPlayerMapper

{

private readonly string \_connectionString =

"Data Source=(local);Initial Catalog=GameDB;Integrated Security=True";

public bool IsPlayerNameExistsInDb(string name)

{

using(SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using(SqlCommand command = connection.CreateCommand())

{

command.CommandText = "SELECT count(\*) FROM Player WHERE [Name] = @name";

command.Parameters.AddWithValue("@name", name);

return (int)command.ExecuteScalar() > 0;

}

}

}

public void AddNewPlayerIntoDb(string name)

{

using(SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using(SqlCommand command = connection.CreateCommand())

{

command.CommandText = "INSERT INTO Player ([Name]) VALUES (@name)";

command.Parameters.AddWithValue("@name", name);

command.ExecuteNonQuery();

}

}

}

}

### **Exercise 2: Player Class – Consuming the Mapper**

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)

playerMapper = new 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);

}

}

1. **NUNIT-HANDSON**

## 2. Calculator Unit Test Setup

### Exercise 1: Create Calculator Library

// Project: CalcLibrary

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b)

{

return a + b;

}

public int Subtract(int a, int b)

{

return a - b;

}

public int Multiply(int a, int b)

{

return a \* b;

}

public int Divide(int a, int b)

{

return a / b;

}

}

}

### Exercise 2: Create NUnit Test Project

// Project: CalcLibrary.Tests

using NUnit.Framework;

using CalcLibrary;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator \_calc;

[SetUp]

public void Init()

{

\_calc = new Calculator();

}

[TearDown]

public void Cleanup()

{

\_calc = null;

}

[Test]

public void Add\_WhenCalled\_ReturnsCorrectSum()

{

int result = \_calc.Add(10, 20);

Assert.That(result, Is.EqualTo(30));

}

[TestCase(5, 10, 15)]

[TestCase(-5, -5, -10)]

[TestCase(0, 0, 0)]

public void Add\_WithTestCases\_ReturnsExpectedResult(int a, int b, int expected)

{

int result = \_calc.Add(a, b);

Assert.That(result, Is.EqualTo(expected));

}

[Test]

[Ignore("Ignore this test temporarily")]

public void Subtract\_TestIgnored()

{

Assert.Fail("This test is ignored.");

}

}

}

**2. NUnit-Handson**

## **1. Objectives**

* Demonstrate parameterized test cases using an example
* Test methods that return a value  
  + Use Assert.AreEqual, Assert.Fail, AssertionException
* Test void methods
* Test methods that throw exceptions  
  + Use try-catch and verify message/type
* Explain why testing private methods is not beneficial
* Explain mocking framework and its usage (e.g., Moq for mocking DB, file, etc.)

## **2. Calculator Library**

// Project: CalcLibrary

namespace CalcLibrary

{

public class Calculator

{

private int \_result;

public int Result => \_result;

public int Add(int a, int b)

{

\_result = a + b;

return \_result;

}

public int Subtract(int a, int b)

{

\_result = a - b;

return \_result;

}

public int Multiply(int a, int b)

{

\_result = a \* b;

return \_result;

}

public int Divide(int a, int b)

{

if (b == 0)

throw new ArgumentException("Divisor cannot be zero");

\_result = a / b;

return \_result;

}

public void AllClear()

{

\_result = 0;

}

}

}

## **3. NUnit Test Cases**

// Project: CalcLibrary.Tests

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibrary.Tests

{

[TestFixture]

public class CalculatorTests

{

private Calculator \_calc;

[SetUp]

public void Setup()

{

\_calc = new Calculator();

}

// 1. Test Subtraction - Parameterized

[TestCase(10, 5, 5)]

[TestCase(0, 0, 0)]

[TestCase(-5, -3, -2)]

[TestCase(100, 25, 75)]

public void Subtract\_WithInputs\_ReturnsExpected(int a, int b, int expected)

{

var result = \_calc.Subtract(a, b);

Assert.AreEqual(expected, result);

}

// 2. Test Multiplication - Parameterized

[TestCase(2, 3, 6)]

[TestCase(0, 100, 0)]

[TestCase(-4, 5, -20)]

[TestCase(-3, -2, 6)]

public void Multiply\_WithInputs\_ReturnsExpected(int a, int b, int expected)

{

var result = \_calc.Multiply(a, b);

Assert.AreEqual(expected, result);

}

// 3. Test Division - Parameterized + Exception

[TestCase(10, 2, 5)]

[TestCase(20, 4, 5)]

[TestCase(9, 3, 3)]

public void Divide\_WithValidInputs\_ReturnsExpected(int a, int b, int expected)

{

var result = \_calc.Divide(a, b);

Assert.AreEqual(expected, result);

}

[Test]

public void Divide\_ByZero\_ThrowsArgumentException()

{

try

{

var result = \_calc.Divide(5, 0);

Assert.Fail("Division by zero"); // Should not reach here

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Divisor cannot be zero"));

}

}

// 4. Test Void Method - AllClear

[Test]

public void TestAddAndClear()

{

var result = \_calc.Add(4, 6);

Assert.AreEqual(10, result);

\_calc.AllClear();

Assert.AreEqual(0, \_calc.Result);

}

}

}

**3.NUNIT HANDS ON**

## 1. Source: URL Host Name Parser

### Exercise 1: UtilLib – Class & Method to Test

// Project: UtilLib

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

if (string.IsNullOrWhiteSpace(url))

return "Invalid URL";

try

{

var uri = new Uri(url);

return uri.Host;

}

catch (UriFormatException)

{

return "Invalid URL";

}

}

}

}

## 2. NUnit Test Project – UtilLib.Tests

### Exercise 2: Unit Test Setup & Structure

// Project: UtilLib.Tests

using NUnit.Framework;

using UtilLib;

namespace UtilLib.Tests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser \_parser;

[SetUp]

public void SetUp()

{

\_parser = new UrlHostNameParser();

}

[TearDown]

public void TearDown()

{

\_parser = null;

}

// Valid input test

[Test]

public void ParseHostName\_ValidUrl\_ReturnsHostName()

{

string url = "https://www.example.com/products";

string result = \_parser.ParseHostName(url);

Assert.That(result, Is.EqualTo("www.example.com"));

}

// Invalid input test (blank or malformed)

[Test]

public void ParseHostName\_InvalidUrl\_ReturnsInvalidURL()

{

string url = "htp:/bad";

string result = \_parser.ParseHostName(url);

Assert.That(result, Is.EqualTo("Invalid URL"));

}

// Empty string input

[Test]

public void ParseHostName\_EmptyString\_ReturnsInvalidURL()

{

string url = "";

string result = \_parser.ParseHostName(url);

Assert.That(result, Is.EqualTo("Invalid URL"));

}

// Null string input

[Test]

public void ParseHostName\_NullInput\_ReturnsInvalidURL()

{

string url = null;

string result = \_parser.ParseHostName(url);

Assert.That(result, Is.EqualTo("Invalid URL"));

}

}

}

**4 NUNIT HANDS ON**

## **AccountsManagerLib – Source Code**

// Project: AccountsManagerLib

namespace AccountsManagerLib

{

public class AccountsManager

{

public string Login(string userId, string password)

{

if (string.IsNullOrWhiteSpace(userId) || string.IsNullOrWhiteSpace(password))

throw new ArgumentException("User ID or password cannot be empty");

if ((userId == "user\_11" && password == "secret@user11") ||

(userId == "user\_22" && password == "secret@user22"))

return $"Welcome {userId}!!!";

return "Invalid user id/password";

}

}

}

## AccountsManagerLib.Tests – Test Class

// Project: AccountsManagerLib.Tests

using NUnit.Framework;

using AccountsManagerLib;

using System;

namespace AccountsManagerLib.Tests

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager \_accountsManager;

[SetUp]

public void Setup()

{

\_accountsManager = new AccountsManager();

}

[Test]

public void Login\_ValidCredentialsUser11\_ReturnsWelcomeMessage()

{

string result = \_accountsManager.Login("user\_11", "secret@user11");

Assert.That(result, Is.EqualTo("Welcome user\_11!!!"));

}

[Test]

public void Login\_ValidCredentialsUser22\_ReturnsWelcomeMessage()

{

string result = \_accountsManager.Login("user\_22", "secret@user22");

Assert.That(result, Is.EqualTo("Welcome user\_22!!!"));

}

[Test]

public void Login\_InvalidCredentials\_ReturnsInvalidMessage()

{

string result = \_accountsManager.Login("invalid\_user", "wrongpass");

Assert.That(result, Is.EqualTo("Invalid user id/password"));

}

[Test]

public void Login\_EmptyUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => \_accountsManager.Login("", "somepass"));

Assert.That(ex.Message, Is.EqualTo("User ID or password cannot be empty"));

}

[Test]

public void Login\_EmptyPassword\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => \_accountsManager.Login("user\_11", ""));

Assert.That(ex.Message, Is.EqualTo("User ID or password cannot be empty"));

}

[Test]

public void Login\_NullUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => \_accountsManager.Login(null, "password"));

Assert.That(ex.Message, Is.EqualTo("User ID or password cannot be empty"));

}

[Test]

public void Login\_NullPassword\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => \_accountsManager.Login("user\_22", null));

Assert.That(ex.Message, Is.EqualTo("User ID or password cannot be empty"));

}

}

}

**5. NUNIT HANDS ON**

## **CollectionsLib – Source Code**

**// Project: CollectionsLib**

using System;

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public DateTime JoinDate { get; set; }

public override bool Equals(object obj)

{

if (obj is Employee other)

return Id == other.Id;

return false;

}

public override int GetHashCode()

{

return Id.GetHashCode();

}

}

public class EmployeeRepository

{

public List<Employee> GetEmployees()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Thasleem", JoinDate = new DateTime(2021, 01, 01) },

new Employee { Id = 101, Name = "Aisha", JoinDate = new DateTime(2023, 01, 01) },

new Employee { Id = 102, Name = "Karthik", JoinDate = new DateTime(2022, 05, 01) }

};

}

public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Thasleem", JoinDate = new DateTime(2021, 01, 01) },

new Employee { Id = 102, Name = "Karthik", JoinDate = new DateTime(2022, 05, 01) },

new Employee { Id = 101, Name = "Aisha", JoinDate = new DateTime(2023, 01, 01) }

};

}

}

}

## **CollectionsLib.Tests – Unit Tests**

// Project: CollectionsLib.Tests

using NUnit.Framework;

using CollectionsLib;

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib.Tests

{

[TestFixture]

public class EmployeeRepositoryTests

{

private EmployeeRepository \_repo;

[SetUp]

public void Init()

{

\_repo = new EmployeeRepository();

}

[Test]

public void GetEmployees\_NoNullsInCollection\_Passes()

{

var employees = \_repo.GetEmployees();

Assert.That(employees, Is.All.Not.Null);

}

[Test]

public void GetEmployees\_ContainsEmployeeWithId100\_Passes()

{

var employees = \_repo.GetEmployees();

Assert.That(employees.Any(e => e.Id == 100), Is.True);

}

[Test]

public void GetEmployees\_UniqueEmployees\_Passes()

{

var employees = \_repo.GetEmployees();

var unique = employees.Distinct().ToList();

Assert.That(employees.Count, Is.EqualTo(unique.Count));

}

[Test]

public void GetEmployees\_CompareWithPreviousYearEmployees\_ClassicModel()

{

var current = \_repo.GetEmployees();

var previous = \_repo.GetEmployeesWhoJoinedInPreviousYears();

CollectionAssert.AreEquivalent(current, previous); // Classic Model

}

[Test]

public void GetEmployees\_CompareWithPreviousYearEmployees\_ConstraintModel()

{

var current = \_repo.GetEmployees();

var previous = \_repo.GetEmployeesWhoJoinedInPreviousYears();

Assert.That(current, Is.EquivalentTo(previous)); // Constraint Model

}

}

}

**6. NUNIT HANDS ON**

FourSeasonsLib – Source Code

// Project: FourSeasonsLib

namespace FourSeasonsLib

{

public class SeasonFinder

{

public string GetSeason(string month)

{

month = month.Trim().ToLower();

if (month == "february" || month == "march")

return "Spring";

if (month == "april" || month == "may" || month == "june")

return "Summer";

if (month == "july" || month == "august")

return "Monsoon";

if (month == "september" || month == "october" || month == "november")

return "Autumn";

if (month == "december" || month == "january")

return "Winter";

return "Invalid month";

}

}

}

## FourSeasonsLib.Tests – NUnit Test Project Using TestCaseSource

// Project: FourSeasonsLib.Tests

using NUnit.Framework;

using FourSeasonsLib;

using System.Collections;

namespace FourSeasonsLib.Tests

{

[TestFixture]

public class SeasonFinderTests

{

private SeasonFinder \_finder;

[SetUp]

public void Init()

{

\_finder = new SeasonFinder();

}

// Static data source

public static IEnumerable SeasonTestData

{

get

{

yield return new TestCaseData("February").Returns("Spring");

yield return new TestCaseData("March").Returns("Spring");

yield return new TestCaseData("April").Returns("Summer");

yield return new TestCaseData("May").Returns("Summer");

yield return new TestCaseData("June").Returns("Summer");

yield return new TestCaseData("July").Returns("Monsoon");

yield return new TestCaseData("August").Returns("Monsoon");

yield return new TestCaseData("September").Returns("Autumn");

yield return new TestCaseData("October").Returns("Autumn");

yield return new TestCaseData("November").Returns("Autumn");

yield return new TestCaseData("December").Returns("Winter");

yield return new TestCaseData("January").Returns("Winter");

yield return new TestCaseData("Hello").Returns("Invalid month");

yield return new TestCaseData("").Returns("Invalid month");

}

}

[Test, TestCaseSource(nameof(SeasonTestData))]

public string GetSeason\_Month\_ReturnsExpectedSeason(string month)

{

return \_finder.GetSeason(month);

}

}

}

**7. NUNIT HANDS ON**

## **LeapYearCalculatorLib – Source Code**

// Project: LeapYearCalculatorLib

namespace LeapYearCalculatorLib

{

public class LeapYearCalculator

{

public int IsLeapYear(int year)

{

if (year < 1753 || year > 9999)

return -1;

if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))

return 1;

return 0;

}

}

}

## **LeapYearCalculatorLib.Tests – Parameterized NUnit Tests**

**// Project: LeapYearCalculatorLib.Tests**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearCalculatorTests

{

private LeapYearCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new LeapYearCalculator();

}

[TestCase(2000, 1)] // Leap year divisible by 400

[TestCase(2024, 1)] // Leap year divisible by 4 not 100

[TestCase(1900, 0)] // Not a leap year (div by 100 but not 400)

[TestCase(2023, 0)] // Normal year

[TestCase(1752, -1)] // Invalid (before 1753)

[TestCase(10000, -1)] // Invalid (after 9999)

public void IsLeapYear\_GivenYear\_ReturnsExpectedResult(int year, int expected)

{

var result = \_calculator.IsLeapYear(year);

Assert.That(result, Is.EqualTo(expected));

}

}

}

**8. NUNIT HANDS ON**

**UserManagerLib – Source Code**

// Project: UserManagerLib

namespace UserManagerLib

{

public class User

{

public string PANCardNo { get; set; }

}

public class UserManager

{

public string CreateUser(User user)

{

if (user == null || string.IsNullOrWhiteSpace(user.PANCardNo))

throw new NullReferenceException("PANCard number is required.");

if (user.PANCardNo.Length != 10)

throw new FormatException("PANCard number must be exactly 10 characters long.");

return "User created successfully.";

}

}

}

## **UserManagerLib.Tests – NUnit Unit Tests**

// Project: UserManagerLib.Tests

using NUnit.Framework;

using UserManagerLib;

using System;

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserManagerTests

{

private UserManager \_userManager;

[SetUp]

public void Setup()

{

\_userManager = new UserManager();

}

[Test]

public void CreateUser\_ValidPANCard\_ReturnsSuccessMessage()

{

var user = new User { PANCardNo = "ABCDE1234Z" };

var result = \_userManager.CreateUser(user);

Assert.That(result, Is.EqualTo("User created successfully."));

}

[Test]

public void CreateUser\_EmptyPANCard\_ThrowsNullReferenceException()

{

var user = new User { PANCardNo = "" };

var ex = Assert.Throws<NullReferenceException>(() => \_userManager.CreateUser(user));

Assert.That(ex.Message, Is.EqualTo("PANCard number is required."));

}

[Test]

public void CreateUser\_NullUser\_ThrowsNullReferenceException()

{

User user = null;

var ex = Assert.Throws<NullReferenceException>(() => \_userManager.CreateUser(user));

Assert.That(ex.Message, Is.EqualTo("PANCard number is required."));

}

[TestCase("ABC123")] // Too short

[TestCase("ABCDE123456")] // Too long

public void CreateUser\_InvalidPANLength\_ThrowsFormatException(string pan)

{

var user = new User { PANCardNo = pan };

var ex = Assert.Throws<FormatException>(() => \_userManager.CreateUser(user));

Assert.That(ex.Message, Is.EqualTo("PANCard number must be exactly 10 characters long."));

}

}

}

**9. NUNIT HANDS ON**

**ConverterLib – Source Code**

**// Project: ConverterLib**

namespace ConverterLib

{

public interface IDollarToEuroExchangeRateFeed

{

double GetActualUSDToEuroExchangeRate();

}

public class Converter

{

private readonly IDollarToEuroExchangeRateFeed \_rateFeed;

public Converter(IDollarToEuroExchangeRateFeed rateFeed)

{

\_rateFeed = rateFeed;

}

public double USDToEuro(double amountInUSD)

{

double rate = \_rateFeed.GetActualUSDToEuroExchangeRate();

return amountInUSD \* rate;

}

}

}

## **ConverterLib.Tests – NUnit + Moq Based Test**

**// Project: ConverterLib.Tests**

using NUnit.Framework;

using Moq;

using ConverterLib;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

private Mock<IDollarToEuroExchangeRateFeed> \_mockRateFeed;

private Converter \_converter;

[SetUp]

public void Setup()

{

\_mockRateFeed = new Mock<IDollarToEuroExchangeRateFeed>();

}

[Test]

public void USDToEuro\_WithFixedRate\_ReturnsCorrectConversion()

{

\_mockRateFeed.Setup(x => x.GetActualUSDToEuroExchangeRate()).Returns(0.85);

\_converter = new Converter(\_mockRateFeed.Object);

double result = \_converter.USDToEuro(100);

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

}

[Test]

public void USDToEuro\_ZeroInput\_ReturnsZero()

{

\_mockRateFeed.Setup(x => x.GetActualUSDToEuroExchangeRate()).Returns(0.85);

\_converter = new Converter(\_mockRateFeed.Object);

double result = \_converter.USDToEuro(0);

Assert.That(result, Is.EqualTo(0));

}

[Test]

public void USDToEuro\_NegativeInput\_ReturnsNegativeEuro()

{

\_mockRateFeed.Setup(x => x.GetActualUSDToEuroExchangeRate()).Returns(0.85);

\_converter = new Converter(\_mockRateFeed.Object);

double result = \_converter.USDToEuro(-50);

Assert.That(result, Is.EqualTo(-42.5));

}

}

}