**SQL**

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),

Salary DECIMAL(10, 2),

JoinDate DATE

);

-- Insert Departments

INSERT INTO Departments (DepartmentID, DepartmentName) VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT');

-- Insert Employees

INSERT INTO Employees (EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES

(101, 'Alice', 'Johnson', 1, 50000, '2022-01-10'),

(102, 'Bob', 'Smith', 2, 60000, '2021-06-15'),

(103, 'Charlie', 'Brown', 3, 55000, '2023-03-20');

------------------------------Exercise-1-----------------------------------

CREATE VIEW vw\_EmployeeBasicInfo AS

SELECT

E.EmployeeID,

E.FirstName,

E.LastName,

D.DepartmentName

FROM

Employees E

JOIN

Departments D ON E.DepartmentID = D.DepartmentID;

----------------------------Exercise-2---------------------------------------

CREATE VIEW vw\_EmployeeFullName AS

SELECT

EmployeeID,

FirstName,

LastName,

FirstName + ' ' + LastName AS FullName

FROM

Employees

-------------------------Exercise-3----------------------------------------------

CREATE VIEW vw\_EmployeeAnnualSalary AS

SELECT

EmployeeID,

FirstName,

LastName,

Salary,

Salary \* 12 AS AnnualSalary

FROM

Employees;

----------------------------Exercise-4------------------------------------------------

CREATE VIEW vw\_EmployeeReport AS

SELECT

E.EmployeeID,

E.FirstName + ' ' + E.LastName AS FullName,

D.DepartmentName,

E.Salary \* 12 AS AnnualSalary,

(E.Salary \* 12) \* 0.10 AS Bonus

FROM

Employees E

JOIN

Departments D ON E.DepartmentID = D.DepartmentID;

------------------Exercise 1: Create a Stored Procedure to Insert Employee--------------

CREATE PROCEDURE sp\_InsertEmployee

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE

AS

BEGIN

INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)

VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);

END;

------------- Exercise 2: Modify Stored Procedure (Add Salary in SELECT)------------------

CREATE PROCEDURE sp\_GetEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT FirstName, LastName

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

------to modify:

ALTER PROCEDURE sp\_GetEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT FirstName, LastName, Salary

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

---------Exercise 3: Delete a Stored Procedure-------------------

DROP PROCEDURE sp\_InsertEmployee;

--------- Exercise 4: Execute a Stored Procedure---------------

EXEC sp\_GetEmployeesByDept @DepartmentID = 2;

-----------Exercise 5: Return Data – Total Employees in Department-----

CREATE PROCEDURE sp\_CountEmployeesByDept

@DepartmentID INT

AS

BEGIN

SELECT COUNT(\*) AS TotalEmployees

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

-------------Exercise 6: Output Parameter – Total Salary by Department------

CREATE PROCEDURE sp\_TotalSalaryByDept

@DepartmentID INT,

@TotalSalary DECIMAL(10,2) OUTPUT

AS

BEGIN

SELECT @TotalSalary = SUM(Salary)

FROM Employees

WHERE DepartmentID = @DepartmentID;

END;

--To execute:

DECLARE @TotalSalary DECIMAL(10,2);

EXEC sp\_TotalSalaryByDept 1, @TotalSalary OUTPUT;

SELECT @TotalSalary AS TotalSalary;

----Exercise 7: Update Salary

CREATE PROCEDURE sp\_UpdateEmployeeSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END;

EXEC sp\_UpdateEmployeeSalary 1, 5500.00;

------Exercise 8: Bonus Based on Department----

CREATE PROCEDURE sp\_GiveBonus

@DepartmentID INT,

@BonusAmount DECIMAL(10,2)

AS

BEGIN

UPDATE Employees

SET Salary = Salary + @BonusAmount

WHERE DepartmentID = @DepartmentID;

END;

EXEC sp\_GiveBonus 1, 500.00;

----Exercise 9: Use Transactions------------

CREATE PROCEDURE sp\_UpdateSalaryWithTransaction

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRANSACTION;

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END;

--------Exercise 10: Dynamic SQL (Flexible Filter)------------

CREATE PROCEDURE sp\_GetEmployeesDynamic

@FilterColumn NVARCHAR(100),

@FilterValue NVARCHAR(100)

AS

BEGIN

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'SELECT \* FROM Employees WHERE ' + QUOTENAME(@FilterColumn) + ' = @Value';

EXEC sp\_executesql @SQL, N'@Value NVARCHAR(100)', @Value = @FilterValue;

END;

---------- Exercise 11: Error Handling--------

CREATE PROCEDURE sp\_SafeUpdateSalary

@EmployeeID INT,

@NewSalary DECIMAL(10,2)

AS

BEGIN

BEGIN TRY

UPDATE Employees

SET Salary = @NewSalary

WHERE EmployeeID = @EmployeeID;

END TRY

BEGIN CATCH

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH

END;

------- Exercise 1: Scalar Function – Annual Salary---------

CREATE FUNCTION fn\_CalculateAnnualSalary (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 12;

END;

-----------Exercise 2: Table-Valued Function – Employees by Department------

CREATE FUNCTION fn\_GetEmployeesByDepartment (@DeptID INT)

RETURNS TABLE

AS

RETURN

(

SELECT \* FROM Employees WHERE DepartmentID = @DeptID

);

----------- Exercise 3: User-Defined Function – Bonus (10%)-----------

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

---------Exercise 4: Modify Bonus Function to 15%--------------------

ALTER FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.15;

END;

--------- Exercise 5: Delete the Bonus Function-------------

DROP FUNCTION fn\_CalculateBonus;

--------Exercise 6: Execute Annual Salary Function for All Employees------

SELECT

FirstName,

LastName,

Salary,

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees;

------Exercise 7: Annual Salary for a Specific Employee (ID = 1)-------

SELECT

dbo.fn\_CalculateAnnualSalary(Salary) AS AnnualSalary

FROM Employees

WHERE EmployeeID = 1;

------Exercise 8: Employees from Finance Department (ID = 3)------

SELECT \* FROM dbo.fn\_GetEmployeesByDepartment(3);

------Exercise 9: Nested Function – Total Compensation (Annual Salary + Bonus)----

CREATE FUNCTION fn\_CalculateBonus (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN @Salary \* 0.10;

END;

CREATE FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

------ Exercise 10: Modify Nested Function to Use Updated Bonus Logic------

ALTER FUNCTION fn\_CalculateTotalCompensation (@Salary DECIMAL(10,2))

RETURNS DECIMAL(10,2)

AS

BEGIN

RETURN dbo.fn\_CalculateAnnualSalary(@Salary) + dbo.fn\_CalculateBonus(@Salary);

END;

---Exercise 1: AFTER Trigger — Log Salary Changes

-- Step 1: Create a log table

CREATE TABLE EmployeeChanges (

ChangeID INT IDENTITY(1,1) PRIMARY KEY,

EmployeeID INT,

OldSalary DECIMAL(10,2),

NewSalary DECIMAL(10,2),

ChangeDate DATETIME

);

-- Step 2: Create AFTER UPDATE trigger

CREATE TRIGGER trg\_AfterSalaryUpdate

ON Employees

AFTER UPDATE

AS

BEGIN

INSERT INTO EmployeeChanges (EmployeeID, OldSalary, NewSalary, ChangeDate)

SELECT

d.EmployeeID,

d.Salary,

i.Salary,

GETDATE()

FROM

deleted d

INNER JOIN

inserted i ON d.EmployeeID = i.EmployeeID

WHERE

d.Salary <> i.Salary;

END;

---Exercise 2: INSTEAD OF DELETE Trigger — Prevent Deletion

CREATE TRIGGER trg\_PreventDelete

ON Employees

INSTEAD OF DELETE

AS

BEGIN

RAISERROR ('Deletion from Employees table is not allowed.', 16, 1);

END;

------- Exercise 3: LOGON Trigger — Restrict Access During Maintenance

CREATE TRIGGER trg\_BlockLoginDuringMaintenance

ON ALL SERVER

FOR LOGON

AS

BEGIN

IF DATEPART(HOUR, GETDATE()) = 2

BEGIN

ROLLBACK;

PRINT 'Login is blocked due to scheduled maintenance from 2 AM to 3 AM.';

END

END;

-------Exercise 4: Modify a Trigger using SSMS

------ Exercise 5: Delete a Trigger

DROP TRIGGER trg\_PreventDelete;

-- Exercise 6: Trigger to Update Computed Column

-- Step 1: Add new column

ALTER TABLE Employees ADD AnnualSalary DECIMAL(10,2);

-- Step 2: Create trigger to update AnnualSalary

CREATE TRIGGER trg\_UpdateAnnualSalary

ON Employees

AFTER INSERT, UPDATE

AS

BEGIN

UPDATE E

SET AnnualSalary = E.Salary \* 12

FROM Employees E

JOIN inserted I ON E.EmployeeID = I.EmployeeID;

END;

--- Exercise 1: Create a Cursor to Print Employee Details

DECLARE @EmployeeID INT,

@FirstName VARCHAR(50),

@LastName VARCHAR(50),

@DepartmentID INT,

@Salary DECIMAL(10,2),

@JoinDate DATE;

DECLARE employee\_cursor CURSOR FOR

SELECT EmployeeID, FirstName, LastName, DepartmentID, Salary, JoinDate

FROM Employees;

OPEN employee\_cursor;

FETCH NEXT FROM employee\_cursor INTO @EmployeeID, @FirstName, @LastName, @DepartmentID, @Salary, @JoinDate;

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT 'EmployeeID: ' + CAST(@EmployeeID AS VARCHAR) +

', Name: ' + @FirstName + ' ' + @LastName +

', DeptID: ' + CAST(@DepartmentID AS VARCHAR) +

', Salary: ' + CAST(@Salary AS VARCHAR) +

', JoinDate: ' + CAST(@JoinDate AS VARCHAR);

FETCH NEXT FROM employee\_cursor INTO @EmployeeID, @FirstName, @LastName, @DepartmentID, @Salary, @JoinDate;

END

CLOSE employee\_cursor;

DEALLOCATE employee\_cursor;

------- Exercise 2: Types of Cursors

-- Static Cursor (takes snapshot, no changes seen after open)

DECLARE static\_cursor STATIC CURSOR FOR

SELECT \* FROM Employees;

OPEN static\_cursor;

-- FETCH commands go here

CLOSE static\_cursor;

DEALLOCATE static\_cursor;

-- Dynamic Cursor (reflects all changes to data)

DECLARE dynamic\_cursor DYNAMIC CURSOR FOR

SELECT \* FROM Employees;

OPEN dynamic\_cursor;

-- FETCH commands go here

CLOSE dynamic\_cursor;

DEALLOCATE dynamic\_cursor;

-- Forward-Only Cursor (default, read-only, fastest)

DECLARE forward\_cursor CURSOR FORWARD\_ONLY FOR

SELECT \* FROM Employees;

OPEN forward\_cursor;

-- FETCH NEXT only

CLOSE forward\_cursor;

DEALLOCATE forward\_cursor;

-- Keyset-Driven Cursor (keys are fixed, data may change)

DECLARE keyset\_cursor KEYSET CURSOR FOR

SELECT \* FROM Employees;

OPEN keyset\_cursor;

-- FETCH commands go here

CLOSE keyset\_cursor;

DEALLOCATE keyset\_cursor;

**MOQ HANDS ON EXCERISE-1**

├── CustomerCommLib

│ ├── IMailSender.cs

│ └── CustomerComm.cs

│

└── CustomerComm.Tests

└── CustomerCommTests.cs

CustomerCommLib/IMailSender.cs:

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

}

CustomerCommLib/CustomerComm.cs:

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");

}

}

}

CustomerComm.Tests/CustomerCommTests.cs:

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerCommTests

{

private CustomerComm \_customerComm;

[OneTimeSetUp]

public void Init()

{

var mock = new Mock<IMailSender>();

mock.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

\_customerComm = new CustomerComm(mock.Object);

}

[TestCase]

public void SendMailToCustomer\_ValidScenario\_ReturnsTrue()

{

bool result = \_customerComm.SendMailToCustomer();

Assert.That(result, Is.True);

}

}

}

2: **Mock File System:**

├── MagicFilesLib

│ ├── IDirectoryExplorer.cs

│ └── DirectoryExplorer.cs

│

└── DirectoryExplorer.Tests

└── DirectoryExplorerTests.cs

MagicFilesLib/IDirectoryExplorer.cs

using System.Collections.Generic;

namespace MagicFilesLib

{

public interface IDirectoryExplorer

{

ICollection<string> GetFiles(string path);

}

}

MagicFilesLib/DirectoryExplorer.cs:

using System.Collections.Generic;

using System.IO;

namespace MagicFilesLib

{

public class DirectoryExplorer : IDirectoryExplorer

{

public ICollection<string> GetFiles(string path)

{

return Directory.GetFiles(path);

}

}

}

DirectoryExplorer.Tests/DirectoryExplorerTests.cs:

using NUnit.Framework;

using Moq;

using MagicFilesLib;

using System.Collections.Generic;

namespace DirectoryExplorer.Tests

{

[TestFixture]

public class DirectoryExplorerTests

{

private readonly string \_file1 = "file.txt";

private readonly string \_file2 = "file2.txt";

private IDirectoryExplorer \_directoryExplorer;

[OneTimeSetUp]

public void Init()

{

var mock = new Mock<IDirectoryExplorer>();

mock.Setup(d => d.GetFiles(It.IsAny<string>())).Returns(new List<string> { \_file1, \_file2 });

\_directoryExplorer = mock.Object;

}

[TestCase]

public void GetFiles\_ValidPath\_ReturnsMockedFiles()

{

var files = \_directoryExplorer.GetFiles("dummyPath");

Assert.That(files, Is.Not.Null);

Assert.That(files.Count, Is.EqualTo(2));

Assert.That(files, Does.Contain(\_file1));

}

}

}

3: **Mock Database Access:**

├── PlayersManagerLib

│ ├── IPlayerMapper.cs

│ ├── PlayerMapper.cs

│ └── Player.cs

│

└── PlayersManager.Tests

└── PlayerTests.cs

PlayersManagerLib/IPlayerMapper.cs:

namespace PlayersManagerLib

{

public interface IPlayerMapper

{

bool IsPlayerNameExistsInDb(string name);

void AddNewPlayerIntoDb(string name);

}

}

PlayersManagerLib/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)

{

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);

}

}

}

PlayersManager.Tests/PlayerTests.cs:

using NUnit.Framework;

using Moq;

using PlayersManagerLib;

using System;

namespace PlayersManager.Tests

{

[TestFixture]

public class PlayerTests

{

[Test]

public void RegisterNewPlayer\_ValidName\_ReturnsPlayerObject()

{

var mockMapper = new Mock<IPlayerMapper>();

mockMapper.Setup(m => m.IsPlayerNameExistsInDb("Virat")).Returns(false);

Player player = Player.RegisterNewPlayer("Virat", mockMapper.Object);

Assert.That(player.Name, Is.EqualTo("Virat"));

Assert.That(player.Age, Is.EqualTo(23));

Assert.That(player.Country, Is.EqualTo("India"));

Assert.That(player.NoOfMatches, Is.EqualTo(30));

}

[Test]

public void RegisterNewPlayer\_EmptyName\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => Player.RegisterNewPlayer("", new Mock<IPlayerMapper>().Object));

Assert.That(ex.Message, Is.EqualTo("Player name can’t be empty."));

}

[Test]

public void RegisterNewPlayer\_NameExists\_ThrowsArgumentException()

{

var mockMapper = new Mock<IPlayerMapper>();

mockMapper.Setup(m => m.IsPlayerNameExistsInDb("Sachin")).Returns(true);

var ex = Assert.Throws<ArgumentException>(() => Player.RegisterNewPlayer("Sachin", mockMapper.Object));

Assert.That(ex.Message, Is.EqualTo("Player name already exists."));

}

}

}

**NUNIT HANDS-ON-EXCERISE-1**

│

├── CalcLibrary (Class Library Project)

│ ├── Calculator.cs

│ ├── IDiscountService.cs

│ └── InvoiceCalculator.cs

│

└── CalcLibraryTests (NUnit Test Project)

└── CalculatorTests.cs

└── InvoiceCalculatorTests.cs

namespace CalcLibrary

{

public class Calculator

{

public int Add(int a, int b) => a + b;

public int Subtract(int a, int b) => a - b;

public int Multiply(int a, int b) => a \* b;

public double Divide(int a, int b)

{

if (b == 0) throw new DivideByZeroException("Cannot divide by zero.");

return (double)a / b;

}

}

}

namespace CalcLibrary

{

public interface IDiscountService

{

double GetDiscountPercentage();

}

}

namespace CalcLibrary

{

public class InvoiceCalculator

{

private readonly IDiscountService \_discountService;

public InvoiceCalculator(IDiscountService discountService)

{

\_discountService = discountService;

}

public double CalculateFinalAmount(double amount)

{

double discount = \_discountService.GetDiscountPercentage();

return amount - (amount \* discount / 100);

}

}

}

using NUnit.Framework;

using CalcLibrary;

using System;

namespace CalcLibraryTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new Calculator();

}

[TearDown]

public void Teardown()

{

\_calculator = null;

}

[TestCase(2, 3, 5)]

[TestCase(-1, -1, -2)]

[TestCase(-1, 1, 0)]

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

{

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

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

}

[TestCase(5, 3, 2)]

[TestCase(3, 5, -2)]

[TestCase(-1, -1, 0)]

public void Subtract\_WithVariousInputs\_ReturnsExpectedResult(int a, int b, int expected)

{

int result = \_calculator.Subtract(a, b);

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

}

[TestCase(2, 3, 6)]

[TestCase(0, 5, 0)]

[TestCase(-2, 4, -8)]

public void Multiply\_WithVariousInputs\_ReturnsExpectedResult(int a, int b, int expected)

{

int result = \_calculator.Multiply(a, b);

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

}

[TestCase(10, 2, 5)]

[TestCase(9, 3, 3)]

[TestCase(-10, 2, -5)]

public void Divide\_WithVariousInputs\_ReturnsExpectedResult(int a, int b, double expected)

{

double result = \_calculator.Divide(a, b);

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

}

[Test]

public void Divide\_ByZero\_ThrowsDivideByZeroException()

{

Assert.Throws<DivideByZeroException>(() => \_calculator.Divide(10, 0));

}

[Test, Ignore("Pending subtraction enhancement")]

public void Subtract\_NegativeTest\_Placeholder()

{

Assert.Fail("Not implemented yet.");

}

}

}

using NUnit.Framework;

using CalcLibrary;

using Moq;

namespace CalcLibraryTests

{

[TestFixture]

public class InvoiceCalculatorTests

{

[Test]

public void CalculateFinalAmount\_WithMockedDiscountService\_ReturnsDiscountedAmount()

{

var mockDiscountService = new Mock<IDiscountService>();

mockDiscountService.Setup(ds => ds.GetDiscountPercentage()).Returns(10);

var invoiceCalculator = new InvoiceCalculator(mockDiscountService.Object);

double result = invoiceCalculator.CalculateFinalAmount(1000);

Assert.That(result, Is.EqualTo(900));

}

}

}

**Total Tests:** 15

**Passed:** 14

 **Skipped:** 1

 **Failed:** 0

**NUNIT HANDS-ON-EXCERISE-2**

Structure

Solution

│

├── MathLibrary (Class Library)

│ ├── Calculator.cs

│ └── IMathService.cs (optional for mocking)

│

└── MathLibraryTests (NUnit Test Project)

├── CalculatorTests.cs

└── CalculatorMockTests.cs

**1. MathLibrary/Calculator.cs**

namespace MathLibrary

{

public class Calculator

{

private double result;

public double GetResult => result;

public double Add(double a, double b)

{

result = a + b;

return result;

}

public double Subtract(double a, double b)

{

result = a - b;

return result;

}

public double Multiply(double a, double b)

{

result = a \* b;

return result;

}

public double Divide(double a, double b)

{

if (b == 0)

throw new ArgumentException("Division by zero is not allowed.");

result = a / b;

return result;

}

public void AllClear()

{

result = 0;

}

}

}

**2. MathLibraryTests/CalculatorTests.cs**

using NUnit.Framework;

using MathLibrary;

using System;

namespace MathLibraryTests

{

[TestFixture]

public class CalculatorTests

{

private Calculator calculator;

[SetUp]

public void Init()

{

calculator = new Calculator();

}

[TestCase(5, 3, 2)]

[TestCase(10, 15, -5)]

[TestCase(-5, -5, 0)]

public void Subtract\_WithVariousInputs\_ReturnsExpected(double a, double b, double expected)

{

var actual = calculator.Subtract(a, b);

Assert.AreEqual(expected, actual);

}

[TestCase(2, 3, 6)]

[TestCase(-2, 4, -8)]

[TestCase(0, 10, 0)]

public void Multiply\_WithVariousInputs\_ReturnsExpected(double a, double b, double expected)

{

var actual = calculator.Multiply(a, b);

Assert.AreEqual(expected, actual);

}

[TestCase(10, 2, 5)]

[TestCase(-10, 2, -5)]

[TestCase(9, 3, 3)]

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

{

var actual = calculator.Divide(a, b);

Assert.AreEqual(expected, actual);

}

[Test]

public void Divide\_ByZero\_ThrowsExceptionWithMessage()

{

try

{

calculator.Divide(10, 0);

Assert.Fail("Division by zero");

}

catch (ArgumentException ex)

{

Assert.That(ex.Message, Is.EqualTo("Division by zero is not allowed."));

}

}

[Test]

public void TestAddAndClear()

{

double result = calculator.Add(4, 6);

Assert.AreEqual(10, result);

Assert.AreEqual(10, calculator.GetResult);

calculator.AllClear();

Assert.AreEqual(0, calculator.GetResult);

}

}

}

3. MathLibraryTests/CalculatorMockTests.cs

namespace MathLibrary

{

public interface IMathService

{

double GetValue();

}

}

using NUnit.Framework;

using Moq;

using MathLibrary;

[TestFixture]

public class CalculatorMockTests

{

[Test]

public void Add\_UsesMockedServiceValue\_AddsCorrectly()

{

var mock = new Mock<IMathService>();

mock.Setup(m => m.GetValue()).Returns(5);

var calc = new Calculator();

double result = calc.Add(mock.Object.GetValue(), 5);

Assert.AreEqual(10, result);

}

}

Output:

 **Total Tests Run:** 11

 **Passed:** 11

 **Failed:** 0

 **Skipped:** 0

**3. hands-on-excerise**

Solution

│

├── UtilLib (Class Library Project)

│ └── UrlHostNameParser.cs

│

└── UtilLib.Tests (NUnit Test Project)

└── UrlHostNameParserTests.cs

UtilLib/UrlHostNameParser.cs:

using System;

namespace UtilLib

{

public class UrlHostNameParser

{

public string ParseHostName(string url)

{

if (string.IsNullOrWhiteSpace(url))

return "Invalid URL";

try

{

Uri uri = new Uri(url);

return uri.Host;

}

catch

{

return "Invalid URL";

}

}

}

}

2. **UtilLib.Tests/UrlHostNameParserTests.cs**

using NUnit.Framework;

using UtilLib;

namespace UtilLib.Tests

{

[TestFixture]

public class UrlHostNameParserTests

{

private UrlHostNameParser \_parser;

[SetUp]

public void SetUp()

{

\_parser = new UrlHostNameParser();

}

[Test]

public void ParseHostName\_ValidHttpUrl\_ReturnsHostName()

{

string result = \_parser.ParseHostName("http://example.com/path");

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

}

[Test]

public void ParseHostName\_InvalidUrl\_ReturnsInvalidUrl()

{

string result = \_parser.ParseHostName("not a valid url");

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

}

[Test]

public void ParseHostName\_EmptyInput\_ReturnsInvalidUrl()

{

string result = \_parser.ParseHostName("");

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

}

[Test]

public void ParseHostName\_NullInput\_ReturnsInvalidUrl()

{

string result = \_parser.ParseHostName(null);

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

}

}

}

Output:

Expected string length 11 but was 13. Strings differ at index 0.

Expected: "example.com"

But was: "wronghost.com"

**NUNIT HANDS-ON-EXCERISE-4**

Solution

│

├── AccountsManagerLib (Class Library Project)

│ └── AccountsManager.cs

│

└── AccountsManagerLib.Tests (NUnit Test Project)

└── AccountsManagerTests.cs

1. AccountsManagerLib/AccountsManager.cs

using System;

namespace AccountsManagerLib

{

public class AccountsManager

{

public string Login(string userId, string password)

{

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

throw new ArgumentException("User ID and password are required.");

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

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

{

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

}

return "Invalid user id/password";

}

}

}

2.AccountsManagerLib.Tests/AccountsManagerTests.cs

using NUnit.Framework;

using AccountsManagerLib;

using System;

namespace AccountsManagerLib.Tests

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager \_manager;

[SetUp]

public void SetUp()

{

\_manager = new AccountsManager();

}

[Test]

public void Login\_ValidCredentialsForUser11\_ReturnsWelcomeMessage()

{

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

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

}

[Test]

public void Login\_ValidCredentialsForUser22\_ReturnsWelcomeMessage()

{

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

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

}

[Test]

public void Login\_InvalidCredentials\_ReturnsInvalidMessage()

{

string result = \_manager.Login("user\_11", "wrongPassword");

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

}

[Test]

public void Login\_EmptyUserId\_ThrowsArgumentException()

{

var ex = Assert.Throws<ArgumentException>(() => \_manager.Login("", "secret@user11"));

Assert.That(ex.Message, Is.EqualTo("User ID and password are required."));

}

[Test]

public void Login\_EmptyPassword\_ThrowsArgumentException()

{

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

Assert.That(ex.Message, Is.EqualTo("User ID and password are required."));

}

[Test]

public void Login\_NullInputs\_ThrowsArgumentException()

{

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

Assert.That(ex.Message, Is.EqualTo("User ID and password are required."));

}

}

}

OUTPUT: Passed! - Failed: 0, Passed: 6, Skipped: 0, Total: 6

**NUnit-Handson-EXCERSISE-5**

Solution

│

├── CollectionsLib (Class Library Project)

│ ├── Employee.cs

│ └── EmployeeManager.cs

│

└── CollectionsLib.Tests (NUnit Test Project)

└── EmployeeManagerTests.cs

CollectionsLib/Employee.cs:

using System;

namespace CollectionsLib

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public override bool Equals(object obj)

{

if (obj is Employee other)

return this.Id == other.Id;

return false;

}

public override int GetHashCode()

{

return Id.GetHashCode();

}

}

}

1. CollectionsLib/EmployeeManager.cs:

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib

{

public class EmployeeManager

{

public List<Employee> GetEmployees()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 102, Name = "Charlie" }

};

}

public List<Employee> GetEmployeesWhoJoinedInPreviousYears()

{

return new List<Employee>

{

new Employee { Id = 100, Name = "Alice" },

new Employee { Id = 101, Name = "Bob" },

new Employee { Id = 102, Name = "Charlie" }

};

}

}

}

1. CollectionsLib.Tests/EmployeeManagerTests.cs:

using NUnit.Framework;

using CollectionsLib;

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib.Test

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager \_manager;

[SetUp]

public void SetUp()

{

\_manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_ShouldContainNoNullValues()

{

var employees = \_manager.GetEmployees();

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

}

[Test]

public void GetEmployees\_ShouldContainEmployeeWithId100()

{

var employees = \_manager.GetEmployees();

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

}

[Test]

public void GetEmployees\_ShouldReturnUniqueEmployees()

{

var employees = \_manager.GetEmployees();

var distinctCount = employees.Select(e => e.Id).Distinct().Count();

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

}

[Test]

public void GetEmployeesAndPreviousYearsEmployees\_ShouldBeEqual\_ConstraintModel()

{

var current = \_manager.GetEmployees();

var previous = \_manager.GetEmployeesWhoJoinedInPreviousYears();

CollectionAssert.AreEqual(current, previous);

}

[Test]

public void GetEmployeesAndPreviousYearsEmployees\_ShouldBeEqual\_ClassicModel()

{

var current = \_manager.GetEmployees();

var previous = \_manager.GetEmployeesWhoJoinedInPreviousYears();

Assert.That(current, Is.EqualTo(previous));

}

}

}

OUTPUT: Passed! - Failed: 0, Passed: 5, Skipped: 0, Total: 5

**HANDS-ON-EXCERSICE-6**

Solution

│

├── FourSeasonsLib (Class Library Project)

│ └── SeasonFinder.cs

│

└── FourSeasonsLib.Tests (NUnit Test Project)

└── SeasonFinderTests.cs

1.FourSeasonsLib/SeasonFinder.cs:

using System;

namespace FourSeasonsLib

{

public class SeasonFinder

{

public string GetSeason(string month)

{

if (string.IsNullOrWhiteSpace(month))

return "Invalid";

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

switch (month)

{

case "february":

case "march":

return "Spring";

case "april":

case "may":

case "june":

return "Summer";

case "july":

case "august":

return "Monsoon";

case "september":

case "october":

case "november":

return "Autumn";

case "december":

case "january":

return "Winter";

default:

return "Invalid";

}

}

}

}

2. FourSeasonsLib.Tests/SeasonFinderTests.cs:

using NUnit.Framework;

using FourSeasonsLib;

using System.Collections.Generic;

namespace FourSeasonsLib.Tests

{

[TestFixture]

public class SeasonFinderTests

{

private SeasonFinder \_finder;

[SetUp]

public void Setup()

{

\_finder = new SeasonFinder();

}

// Data source for TestCaseSource (external method)

public static IEnumerable<TestCaseData> ValidMonthSeasonCases()

{

yield return new TestCaseData("February", "Spring");

yield return new TestCaseData("March", "Spring");

yield return new TestCaseData("April", "Summer");

yield return new TestCaseData("May", "Summer");

yield return new TestCaseData("June", "Summer");

yield return new TestCaseData("July", "Monsoon");

yield return new TestCaseData("August", "Monsoon");

yield return new TestCaseData("September", "Autumn");

yield return new TestCaseData("October", "Autumn");

yield return new TestCaseData("November", "Autumn");

yield return new TestCaseData("December", "Winter");

yield return new TestCaseData("January", "Winter");

}

[Test, TestCaseSource(nameof(ValidMonthSeasonCases))]

public void GetSeason\_ValidMonth\_ReturnsCorrectSeason(string month, string expected)

{

var result = \_finder.GetSeason(month);

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

}

// Inline TestCaseSource using a static property

public static IEnumerable<TestCaseData> InvalidInputs =>

new[]

{

new TestCaseData(null, "Invalid"),

new TestCaseData("", "Invalid"),

new TestCaseData("invalidMonth", "Invalid")

};

[Test, TestCaseSource(nameof(InvalidInputs))]

public void GetSeason\_InvalidInputs\_ReturnsInvalid(string month, string expected)

{

var result = \_finder.GetSeason(month);

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

}

}

}

OUTPUT: Total tests: 15

Passed: 15

Failed: 0

Skipped: 0

Test Run Successful.

**HANDS-ON-EXCERISE-7**

Solution

│

├── LeapYearCalculatorLib (Class Library Project)

│ └── LeapYearCalculator.cs

│

└── LeapYearCalculatorLib.Tests (NUnit Test Project)

└── LeapYearCalculatorTests.cs

1. LeapYearCalculatorLib/LeapYearCalculator.cs

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;

}

}

}

1. LeapYearCalculatorLib.Tests/LeapYearCalculatorTests.cs:

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)]

[TestCase(2016, 1)]

[TestCase(2024, 1)]

public void IsLeapYear\_ValidLeapYears\_ReturnsOne(int year, int expected)

{

int result = \_calculator.IsLeapYear(year);

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

}

[TestCase(2017, 0)]

[TestCase(1900, 0)]

[TestCase(2100, 0)]

public void IsLeapYear\_ValidNonLeapYears\_ReturnsZero(int year, int expected)

{

int result = \_calculator.IsLeapYear(year);

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

}

[TestCase(1000, -1)]

[TestCase(1500, -1)]

[TestCase(10000, -1)]

public void IsLeapYear\_InvalidYear\_ReturnsMinusOne(int year, int expected)

{

int result = \_calculator.IsLeapYear(year);

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

}

}

}

OUTPUT: Passed! - Failed: 0, Passed: 9, Skipped: 0, Total: 9

**HANDS-ON-EXCERISE-8**

│

├── UserManagerLib (Class Library Project)

│ └── UserManager.cs

│

└── UserManagerLib.Tests (NUnit Test Project)

└── UserManagerTests.cs

1. UserManagerLib/UserManager.cs:

using System;

namespace UserManagerLib

{

public class UserManager

{

public string PANCardNo { get; set; }

public string CreateUser()

{

if (string.IsNullOrWhiteSpace(PANCardNo))

throw new NullReferenceException("PANCardNo cannot be null or empty.");

if (PANCardNo.Length != 10)

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

return "User created successfully.";

}

}

}

1. UserManagerLib.Tests/UserManagerTests.cs:

using NUnit.Framework;

using UserManagerLib;

using System;

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserManagerTests

{

private UserManager \_manager;

[SetUp]

public void Setup()

{

\_manager = new UserManager();

}

[Test]

public void CreateUser\_ValidPAN\_ReturnsSuccessMessage()

{

\_manager.PANCardNo = "ABCDE1234Z";

var result = \_manager.CreateUser();

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

}

[Test]

public void CreateUser\_NullPAN\_ThrowsNullReferenceException()

{

\_manager.PANCardNo = null;

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

Assert.That(ex.Message, Is.EqualTo("PANCardNo cannot be null or empty."));

}

[Test]

public void CreateUser\_EmptyPAN\_ThrowsNullReferenceException()

{

\_manager.PANCardNo = "";

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

Assert.That(ex.Message, Is.EqualTo("PANCardNo cannot be null or empty."));

}

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

[TestCase("ABCDEFGHIJKL")] // too long

public void CreateUser\_InvalidLengthPAN\_ThrowsFormatException(string input)

{

\_manager.PANCardNo = input;

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

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

}

}

}

OUTPUT: Passed! - Failed: 0, Passed: 5, Skipped: 0, Total: 5

**HANDSON-EXCERISE-9**

│

├── ConverterLib (Class Library Project)

│ ├── Converter.cs

│ └── IDollarToEuroExchangeRateFeed.cs

│

└── ConverterLib.Tests (NUnit Test Project)

└── ConverterTests.cs

**1. ConverterLib/IDollarToEuroExchangeRateFeed.cs**

namespace ConverterLib

{

public interface IDollarToEuroExchangeRateFeed

{

double GetActualDollarValue();

}

1. ConverterLib/Converter.cs:

namespace ConverterLib

{

public class Converter

{

private readonly IDollarToEuroExchangeRateFeed \_rateFeed;

public Converter(IDollarToEuroExchangeRateFeed rateFeed)

{

\_rateFeed = rateFeed;

}

public double USDToEuro(double dollarAmount)

{

double rate = \_rateFeed.GetActualDollarValue();

return dollarAmount \* rate;

}

}

}

1. ConverterLib.Tests/ConverterTests.cs:

using NUnit.Framework;

using Moq;

using ConverterLib;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

private Mock<IDollarToEuroExchangeRateFeed> \_mockFeed;

private Converter \_converter;

[SetUp]

public void Setup()

{

\_mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

}

[Test]

public void USDToEuro\_ValidAmount\_ReturnsConvertedValue()

{

// Arrange

\_mockFeed.Setup(x => x.GetActualDollarValue()).Returns(0.9); // Example conversion rate

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

// Act

double result = \_converter.USDToEuro(100);

// Assert

Assert.That(result, Is.EqualTo(90.0));

}

[Test]

public void USDToEuro\_ZeroAmount\_ReturnsZero()

{

\_mockFeed.Setup(x => x.GetActualDollarValue()).Returns(0.85);

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

double result = \_converter.USDToEuro(0);

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

}

[Test]

public void USDToEuro\_NegativeAmount\_ReturnsNegativeConvertedValue()

{

\_mockFeed.Setup(x => x.GetActualDollarValue()).Returns(1.0);

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

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

Assert.That(result, Is.EqualTo(-50.0));

}

}

}

OUTPUT: Passed! - Failed: 0, Passed: 3, Skipped: 0, Total: 3