Assignment

Part 1: Basic Level (NUnit Fundamentals)

Exercise 1: Create First Test Case

Write a test class CalculatorTests for the following class:

```
public class Calculator
{
   public int Square(int a) => a * a;
}
```

✔ Task:

- Add a test method Square_ShouldReturnCorrectSum.
- Assert that 2 * 2 =4.

Exercise 2: Test with Multiple Assertions

Given:

```
public class StringHelper
{
   public string ToUpper(string input) => input.ToUpper();
}
```

✔ Task:

- Write a test containing 3 assertions for input "hello".
- Validate length, first character, and full uppercase string.

Exercise 3: Test Case Attribute

Write parameterized tests for a Multiply(int a, int b) method using:

```
[TestCase(2, 3, 6)]
[TestCase(-1, 5, -5)]
[TestCase(0, 19, 0)]
```

Exercise 4: Test Exceptions

Given:

```
public class StudentService
{
   public void ValidateAge(int age)
   {
      if (age < 0) throw new ArgumentException("Invalid age");
   }
}
```

✔ Task:

- Write a test to verify that an exception is thrown when age is negative.

Exercise 5: Setup and Teardown

Write a test class using:

- [SetUp]
- [TearDown]

✔ Task:

- Print a message in both methods.
- Ensure [SetUp] runs before each test and [TearDown] after each test.

🔹 **Part 2: Intermediate Level (Collections, Mocking, Constraints)**

Exercise 6: Collection Assertions

Given:

```
public List<int> GetEvenNumbers() => new List<int> { 2, 4, 6, 8 };
```

✔ Task:

- Assert collection length
- Assert elements are in ascending order
- Assert all are even numbers

Exercise 7: String Constraints

Test "NUnitFramework" using constraints:

- StartsWith
- EndsWith
- Contains
- Has.Length

Exercise 8: Testing Asynchronous Method

Given:

```
public async Task<int> GetMarksAsync()
{
   await Task.Delay(100);
   return 90;
}
```

✔ Task:

- Write async test that validates result 90.
- Ensure method is awaited properly.

Exercise 9: Test Case Source

✔ Task:

- Create custom data source returning list of student marks
- Use [TestCaseSource] to test whether marks are greater than 40

Exercise 1: Bank Account

```
public class BankAccount
{
   public decimal Balance { get; private set; }
   public BankAccount(decimal openingBalance) => Balance = openingBalance;
}
```

✔ Task:

Write a test verifying that setting opening balance = 500 is stored correctly.

Assert balance equals 500

Exercise 2: Test Deposit Method

Add method:

```
public void Deposit(decimal amount) => Balance += amount;
```

✔ Task:

- Deposit 200 into a 1000 balance.
- Assert final balance = 1200.

Exercise 3: Test Withdraw Method

Method:

```
public void Withdraw(decimal amount)
{
   if (amount > Balance)
      throw new InvalidOperationException("Insufficient funds");
   Balance -= amount;
```

}

✔ Tasks:

1. Withdraw 300 from 500 → balance = 200
2. Try to withdraw 600 from 500 → should throw exception

Exercise 4: TestCase Attribute

✔ Task: create parameterized tests:

Opening	Deposit	Expected
100	50	150
0	100	100
500	0	500

Use [TestCase(opening, deposit, expected)].

Exercise 5: Transaction History Count

Add:

```
public List<string> History { get; } = new List<string>();
public void Deposit(decimal amount)
{
   Balance += amount;
   History.Add("Deposit " + amount);
}
```

✔ Task:

- After two deposits, assert History count = 2.

Exercise 6: Using TestCaseSource

Create test data for withdrawals:

```
public static IEnumerable<object[]> WithdrawalCases()
{
   yield return new object[] { 1000, 200, 800 };
   yield return new object[] { 500, 100, 400 };
   yield return new object[] { 250, 50, 200 };
}
```

✔ Task:

- Apply [TestCaseSource(nameof(WithdrawalCases))]
- Assert correct remaining balance.

Exercise 7: Negative Deposit Should Throw Exception

Add rule:

```
if (amount <= 0)
   throw new ArgumentException("Amount must be positive");
```

✔ Task:

- Write a test for negative deposit → expect exception
- Validate exception *message* contains “positive”

Exercise 8: Balance Should Never Become Negative

✔ Task:

- Attempt withdrawal more than balance
- Assert InvalidOperationException
- Assert balance remains unchanged

Exercise 9: Test Interest Calculation

Method:

```
public void ApplyInterest(decimal rate)
{
```

```
    Balance += Balance * rate;
}
```

✔ Task:

- Opening: 1000
- Rate: 0.05
- Final: 1050

Use Assert.AreEqual(1050, account.Balance).