Calculator Application with Unit Tests

This solution demonstrates a simple calculator application with comprehensive unit tests using .NET and NUnit. The solution contains two projects:

1. A class library with calculator functionality
2. A test project with NUnit tests

Prerequisites

- Windows 10 or 11
- Either:
  - Visual Studio 2022 (Community Edition or higher) with .NET desktop development workload installed
  - OR .NET 8.0 SDK installed (for CLI approach)

Creating the Solution

Option 1: Using Visual Studio 2022 GUI

1. Create the Solution and Calculator Project

1. Open Visual Studio 2022
2. Click "Create a new project"
3. Search for "Console App" and select "Console App" (Make sure it's the C# one)
4. Click "Next"
5. Configure your project:
   - Project name: `CalculatorApplication`
   - Solution name: `TestingDemoSolution`
   - Location: Choose your preferred location
   - Check "Place solution and project in the same directory"
6. Click "Next"
7. Select ".NET 8.0" as the target framework
8. Click "Create"

2. Create the Test Project

1. Right-click on the solution in Solution Explorer
2. Select Add → New Project
3. Search for "NUnit" and select "NUnit Test Project"
4. Click "Next"
5. Set project name as `CalculatorTests`
6. Click "Next"
7. Select ".NET 8.0" as the target framework
8. Click "Create"

3. Add Project Reference

1. Right-click on the `CalculatorTests` project in Solution Explorer
2. Select "Add" → "Project Reference"
3. Check the box next to `CalculatorApplication`
4. Click "OK"

Option 2: Using CLI (Command Line Interface)

Open a command prompt and run these commands:

```
# 1. Create a directory for your solution
mkdir TestingDemoSolution
cd TestingDemoSolution

# 2. Create the solution file
dotnet new sln -n TestingDemoSolution

# 3. Create the Calculator project
dotnet new console -n CalculatorApplication

# 4. Create the Test project
dotnet new nunit -n CalculatorTests

# 5. Add projects to the solution
dotnet sln add CalculatorApplication/CalculatorApplication.csproj
dotnet sln add CalculatorTests/CalculatorTests.csproj

# 6. Add reference from test project to main project
cd CalculatorTests
dotnet add reference ../CalculatorApplication/CalculatorApplication.csproj
cd ..
```

Adding the Calculator Service

1. Create the Calculator Service Class

1. In the `CalculatorApplication` project, create a new folder named `Service`
2. Create a new file `CalculateService.cs` in the `Service` folder
3. Add the following code:

```
using System;

namespace CalculatorApplication.Service
{
    public class CalculateService
    {
        public double Number1 { get; set; }
        public double Number2 { get; set; }

        public double Add()
```

```
        {
            return Number1 + Number2;
        }

        public double Subtract()
        {
            return Number1 - Number2;
        }

        public double Multiply()
        {
            return Number1 * Number2;
        }

        public double Divide()
        {
            if (Number2 == 0)
                throw new DivideByZeroException("Cannot divide by zero");
            return Number1 / Number2;
        }

        public double Power()
        {
            return Math.Pow(Number1, Number2);
        }

        public double SquareRoot()
        {
            if (Number1 < 0)
                throw new ArgumentException("Cannot calculate square root of a
negative number");
            return Math.Sqrt(Number1);
        }

        public int Factorial()
        {
            if (Number1 < 0)
                throw new ArgumentException("Cannot calculate factorial of a
negative number");
            if (Number1 > 20)
                throw new ArgumentException("Input too large, may cause
overflow");

            int input = (int)Number1;
            if (input != Number1)
                throw new ArgumentException("Factorial requires an integer
input");

            int factorial = 1;
            for(int i = 1; i <= input; i++)
            {
                factorial *= i;
            }
            return factorial;
```

```
        }
    }
}
```

2. Create the Test Class

1. In the `CalculatorTests` project, rename `UnitTest1.cs` to `CalculatorTests.cs`
2. Replace its contents with:

```
using CalculatorApplication.Service;

namespace CalculatorTests
{
    [TestFixture]
    public class CalculatorTests
    {
        private CalculateService _calculator;

        [SetUp]
        public void Setup()
        {
            _calculator = new CalculateService();
        }

        [Test]
        public void Add_TwoPositiveNumbers_ReturnsCorrectSum()
        {
            // Arrange
            _calculator.Number1 = 5;
            _calculator.Number2 = 3;

            // Act
            double result = _calculator.Add();

            // Assert
            Assert.That(result, Is.EqualTo(8));
        }

        [Test]
        public void Add_NegativeNumbers_ReturnsCorrectSum()
        {
            _calculator.Number1 = -5;
            _calculator.Number2 = -3;
            Assert.That(_calculator.Add(), Is.EqualTo(-8));
        }

        [Test]
        public void Subtract_PositiveNumbers_ReturnsCorrectDifference()
        {
            _calculator.Number1 = 10;
            _calculator.Number2 = 3;
```

```
        Assert.That(_calculator.Subtract(), Is.EqualTo(7));
    }

    [Test]
    public void Subtract_WithNegativeResult_ReturnsNegativeNumber()
    {
        _calculator.Number1 = 3;
        _calculator.Number2 = 10;
        Assert.That(_calculator.Subtract(), Is.EqualTo(-7));
    }

    [Test]
    public void Multiply_TwoPositiveNumbers_ReturnsCorrectProduct()
    {
        _calculator.Number1 = 4;
        _calculator.Number2 = 5;
        Assert.That(_calculator.Multiply(), Is.EqualTo(20));
    }

    [Test]
    public void Multiply_WithZero_ReturnsZero()
    {
        _calculator.Number1 = 5;
        _calculator.Number2 = 0;
        Assert.That(_calculator.Multiply(), Is.EqualTo(0));
    }

    [Test]
    public void Divide_TwoPositiveNumbers_ReturnsCorrectQuotient()
    {
        _calculator.Number1 = 10;
        _calculator.Number2 = 2;
        Assert.That(_calculator.Divide(), Is.EqualTo(5));
    }

    [Test]
    public void Divide_ByZero_ThrowsDivideByZeroException()
    {
        _calculator.Number1 = 10;
        _calculator.Number2 = 0;
        Assert.Throws<DivideByZeroException>(() => _calculator.Divide());
    }

    [Test]
    public void Power_PositiveBaseAndExponent_ReturnsCorrectResult()
    {
        _calculator.Number1 = 2;
        _calculator.Number2 = 3;
        Assert.That(_calculator.Power(), Is.EqualTo(8));
    }

    [Test]
    public void Power_ZeroExponent_ReturnsOne()
    {
```

```
        _calculator.Number1 = 5;
        _calculator.Number2 = 0;
        Assert.That(_calculator.Power(), Is.EqualTo(1));
    }

    [Test]
    public void SquareRoot_PositiveNumber_ReturnsCorrectResult()
    {
        _calculator.Number1 = 16;
        Assert.That(_calculator.SquareRoot(), Is.EqualTo(4));
    }

    [Test]
    public void SquareRoot_Zero_ReturnsZero()
    {
        _calculator.Number1 = 0;
        Assert.That(_calculator.SquareRoot(), Is.EqualTo(0));
    }

    [Test]
    public void SquareRoot_NegativeNumber_ThrowsArgumentException()
    {
        _calculator.Number1 = -4;
        Assert.Throws<ArgumentException>(() => _calculator.SquareRoot());
    }

    [Test]
    public void Factorial_PositiveInteger_ReturnsCorrectResult()
    {
        _calculator.Number1 = 5;
        Assert.That(_calculator.Factorial(), Is.EqualTo(120));
    }

    [Test]
    public void Factorial_Zero_ReturnsOne()
    {
        _calculator.Number1 = 0;
        Assert.That(_calculator.Factorial(), Is.EqualTo(1));
    }

    [Test]
    public void Factorial_NegativeNumber_ThrowsArgumentException()
    {
        _calculator.Number1 = -5;
        Assert.Throws<ArgumentException>(() => _calculator.Factorial());
    }

    [Test]
    public void Factorial_NonInteger_ThrowsArgumentException()
    {
        _calculator.Number1 = 5.5;
        Assert.Throws<ArgumentException>(() => _calculator.Factorial());
    }
```

```
        [Test]
        public void Factorial_TooLarge_ThrowsArgumentException()
        {
            _calculator.Number1 = 21;
            Assert.Throws<ArgumentException>(() => _calculator.Factorial());
        }
    }
}
```

Running the Tests

Using Visual Studio 2022

1. Open Test Explorer:
    - Go to View → Test Explorer
    - Or press Ctrl+E, T
2. In Test Explorer:
    - Click "Run All" to run all tests
    - Click individual tests to run them separately
    - Use the group by feature to organize tests
    - Green check marks indicate passed tests
    - Red X marks indicate failed tests

Using CLI

From the solution directory, you can:

1. Run all tests:

```
dotnet test
```

2. List all available tests:

```
dotnet test --list-tests
```

3. Run tests with detailed output:

```
dotnet test --verbosity detailed
```

4. Run specific tests using filters:

```
dotnet test --filter "FullyQualifiedName~Factorial"  # Runs all tests with
"Factorial" in the name
```

Project Structure

```
TestingDemoSolution/
├── CalculatorApplication/
│   ├── Service/
│   │   └── CalculateService.cs
│   └── Program.cs
├── CalculatorTests/
│   └── CalculatorTests.cs
└── TestingDemoSolution.sln
```

Test Categories

The test suite includes tests for:

- Basic arithmetic operations (Add, Subtract, Multiply, Divide)
- Mathematical functions (Power, Square Root)
- Factorial calculation
- Edge cases (zero, negative numbers)
- Error conditions (division by zero, invalid inputs)

Each test follows the Arrange-Act-Assert pattern and has a descriptive name indicating what it tests.