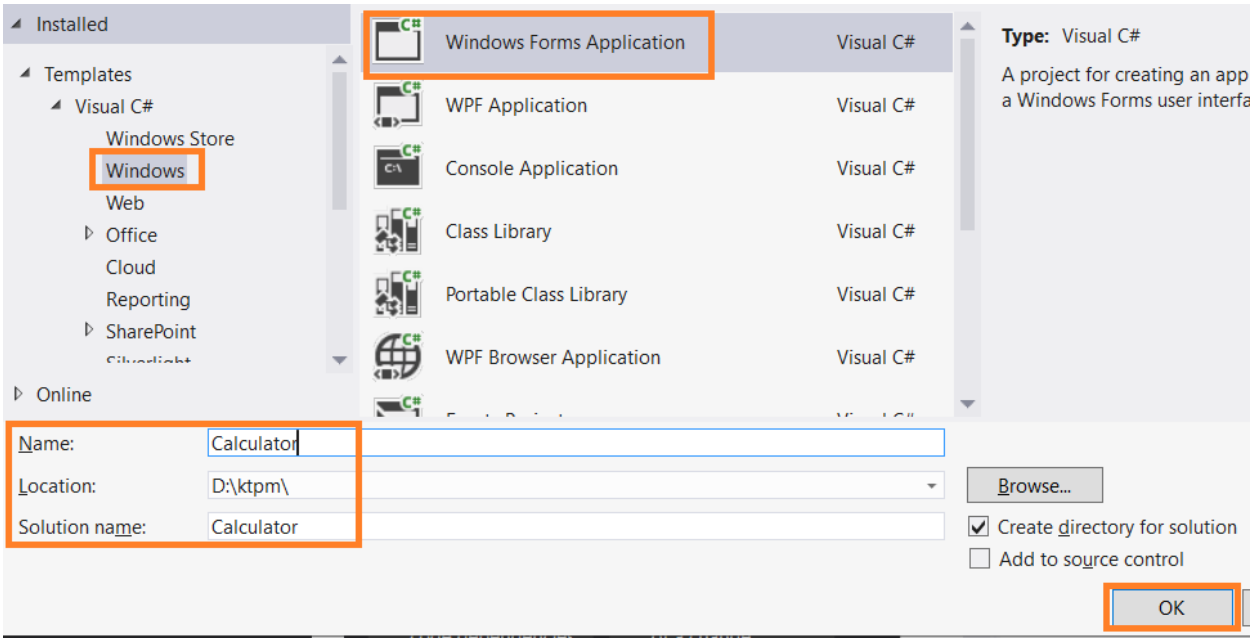
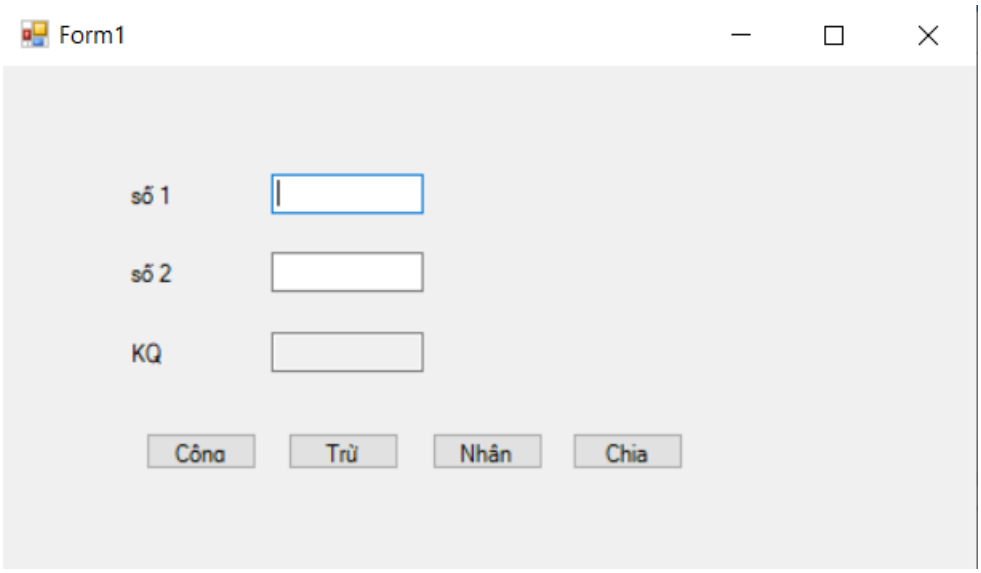
MS UNIT và NUNIT (C#)

Giả sử tạo một project C# thực hiện các phép toán đơn giản cộng, trừ, nhân, chia các số Nguyên

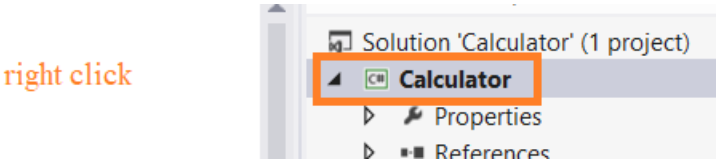


Thiết kế giao diện tính toán như bên dưới và việc xử lý các phép tính đều thông qua lớp `Caculation`.

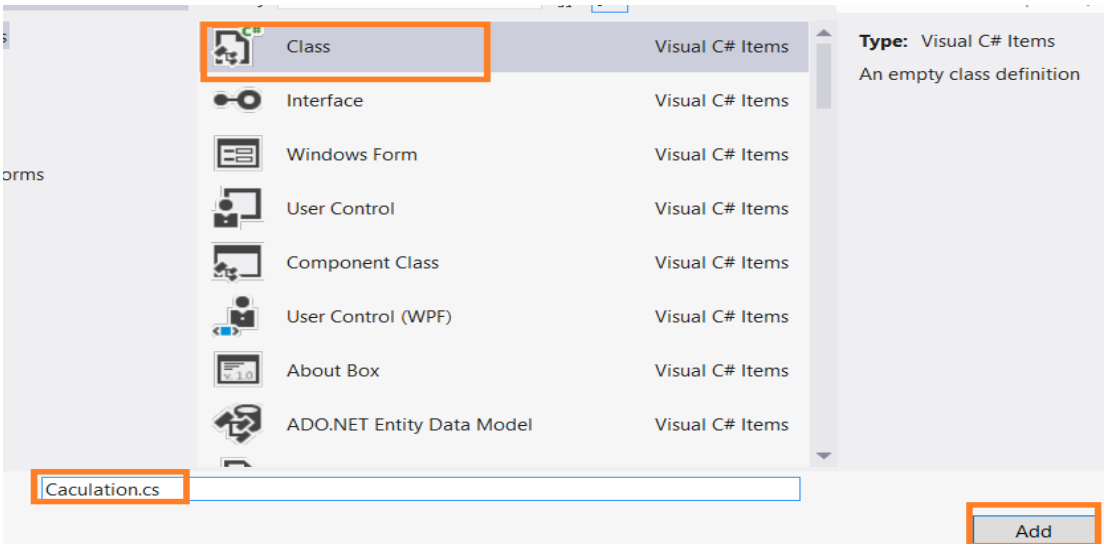


Tạo một tập tin `Calculation.cs` chứa lớp public Calculation dùng có phương thức `Execute` để thực hiện phép tính đơn giản với hai số nguyên.

Tạo project kiểm thử để kiểm thử các phép toán trong chương trình trên.
Click chuột phải Solution > Add > New Projects...



Tạo mới 1 class Caculation




```
namespace Calculator
{
    public class Caculation
    {
        private int a,b;
        public Caculation(int a, int b)
        {
            this.a = a;
            this.b = b;
        }
        public int Execute(string CalSymbol)
        {
            int result = 0;
            switch (CalSymbol)
            {
                case "+":
                    result = this.a + this.b;
                    break;
                case "-":
```

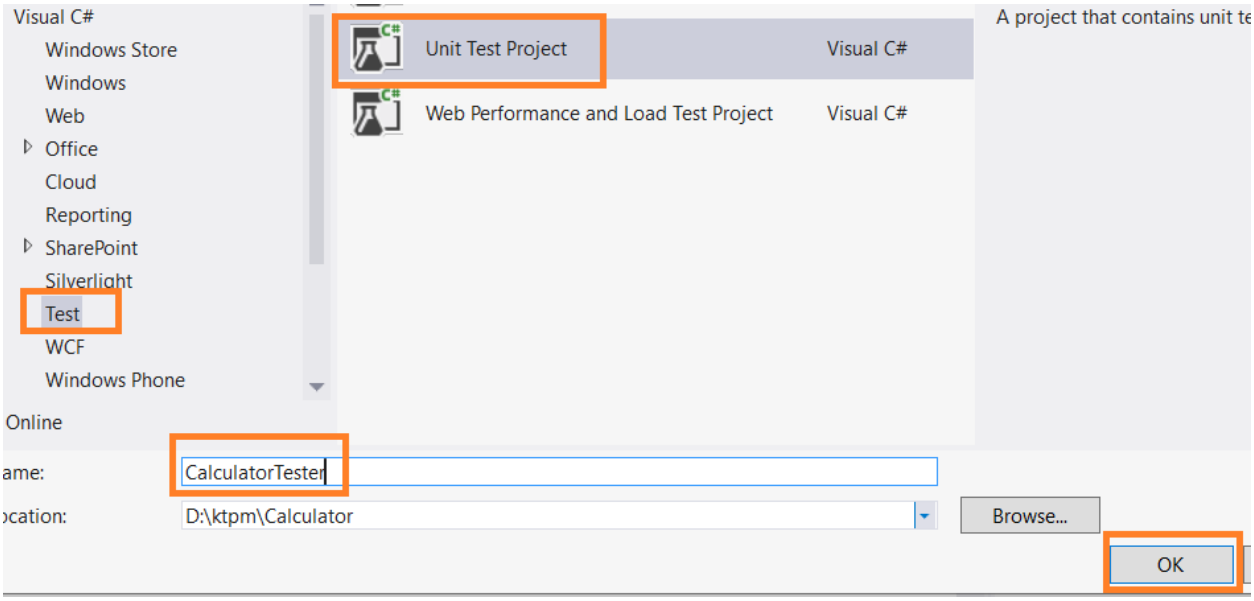
```
                    result = this.a - this.b;
                    break;
                case "*":
                    result = this.a * this.b;
                    break;
                case "/":
                    result = this.a / this.b;
                    break;
            }
            return result;
        }
    }
}
```

Vào winform

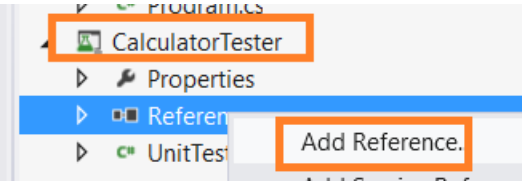
```
private void btn_Cong_Click(object sender, EventArgs e)
        {
            int a, b, ketqua;
            a = int.Parse(txt_1.Text);
            b = int.Parse(txt_2.Text);
            Caculation c = new Caculation(a, b);
            ketqua = c.Execute("+");
            txt_kq.Text = ketqua.ToString();
        }
```

Tạo project kiểm thử để kiểm thử các phép toán trong chương trình trên.
Click chuột phải Solution > Add > New Projects...

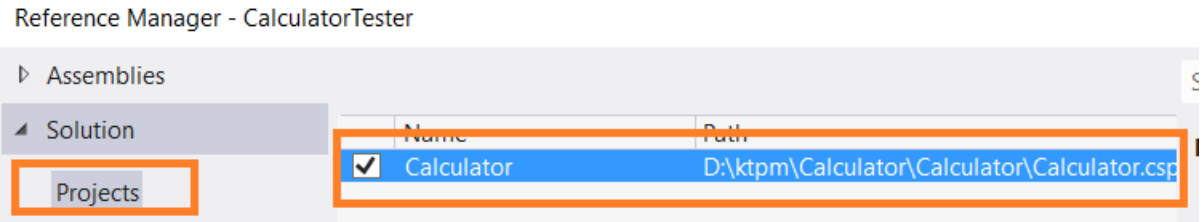
Sau đó chọn loại project là "Unit Test Project" và đặt tên là `CalculatorTester`



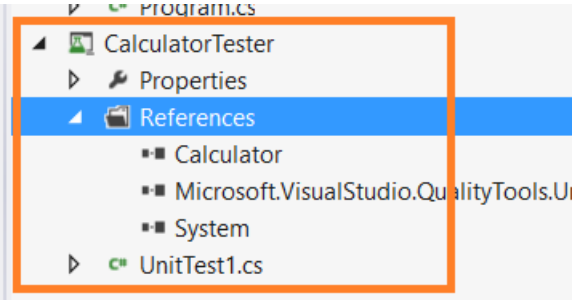
Tại project Unit Test, thực hiện Add Reference để tham chiếu đến project cần thực hiện Unit Test



Chọn project Calculator để test.



Viết code kiểm thử phương thức `Execute` trong lớp `Caculation`



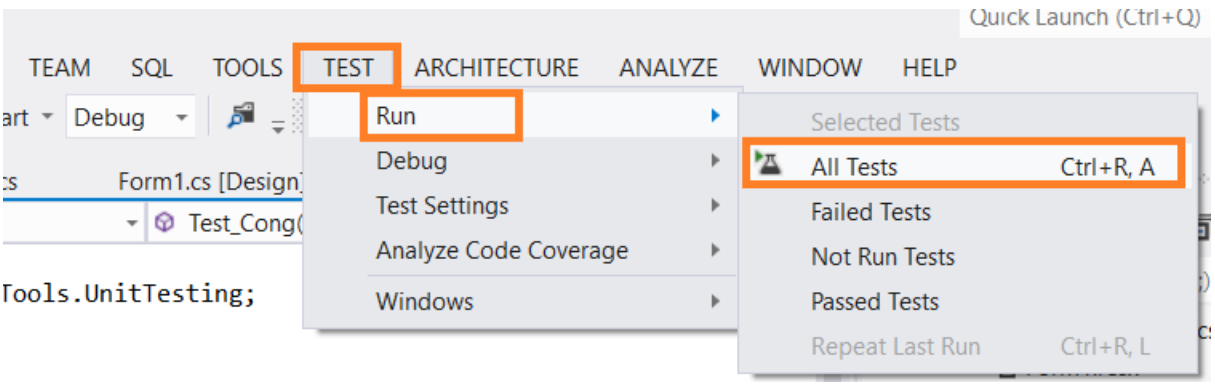

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Calculator;

namespace CalculatorTester
{
    [TestClass]
    public class UnitTest1
    {
        private Caculation c;
        [TestInitialize] // thiet lap du lieu dung chung cho TC
        public void SetUp()
        {
            c = new Caculation(10, 5);
        }
        [TestMethod] //TC1: a =10, b = 5, kq= 15
        public void Test_Cong()
        {
            int expected, actual;
          //  Caculation c = new Caculation(a,b);
            expected = 15;
            actual = c.Execute("+");
            Assert.AreEqual(expected, actual);
```

```
        }
        // sv thực hiện tiếp cho các TC tiếp theo……

       [ExpectedException(typeof(DivideByZeroException))]
        public void Test_ChiaZero()
        {
            c = new Caculation(10,0);
            c.Execute("/");

        }
```

Trong đó:

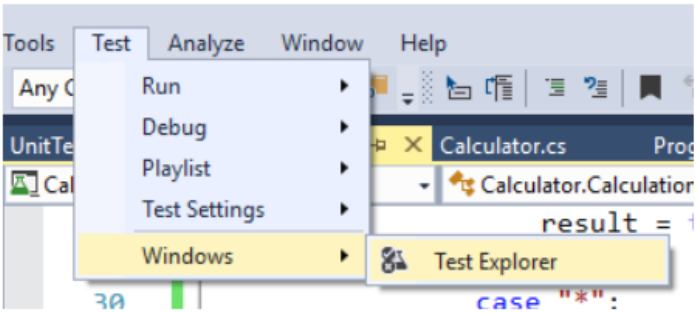
- `[TestClass]`: đánh dấu đây là lớp unit test.
- `[TestMethod]`: phương thức là một test case.
- `[TestInitialize]`: phương thức thực thi trước khi chạy các test case.
- `[TestCleanup]`: phương thức chạy sau cùng trước khi hoàn tất chạy các test case.
- `[ExpectedException(typeof(DivideByZeroException))]`: kết quả mong muốn là xuất hiện ngoại lệ `DivideByZeroException` khi thực hiện phép chia cho 0.
- `[Timeout]`: thiết lập timeout khi thực thi test case.
- `[Ignore]`: bỏ qua tạm thời test case khi thực thi.
- Các phương thức của `Assert.AreEqual` dùng kiểm tra kết quả của phương thức Execute có bằng với kết quả mong muốn của test case hay không.

Chạy các unit test đã viết:

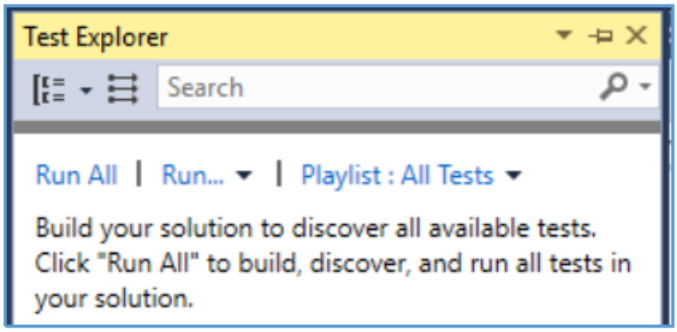


Hoặc

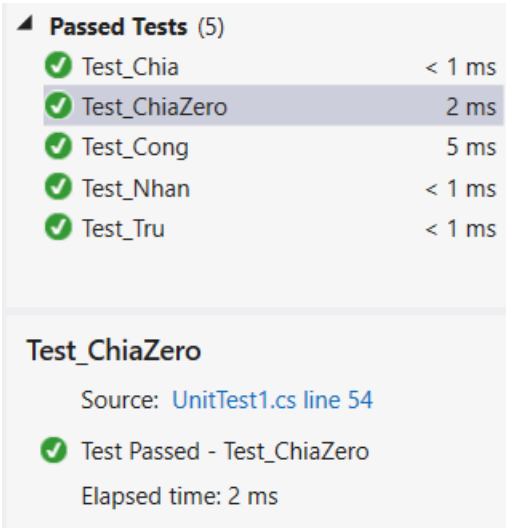
- Mở cửa sổ Test Explorer: menu Test > Windows > Test Explorer



- Click vào link "Run All" để chạy tất cả test case



- Kết quả chạy các test case như sau:



Một vài phương thức của Assert:

- `Assert.AreEqual(expected, actual [, message])`: kiểm tra expected và actual bằng nhau, message nếu được truyền vào sẽ là thông điệp thông báo khi expected và actual không bằng nhau.
- `Assert.IsNull(object [, message])`: kiểm tra một đối tượng là `null`.
- `Assert.IsNotNull(object [, message])`: kiểm tra một đối tượng khác `null`.
- `Assert.AreSame(expected, actual [, message])`: kiểm tra expected và actual tham chiếu đến cùng đối tượng.
- `Assert.IsTrue(bool condition [, message])`: kiểm tra biểu thức condition có là true không.
- `Assert.IsFalse(bool condition [, message])`: kiểm tra biểu thức condition có là false không.
- `Assert.Fail([string message])`

Kiểm tra ngoại lệ: trong nhiều tình huống cũng cần kiểm tra một ngoại lệ xảy ra hoặc không xảy ra một ngoại lệ nào đó, sử dụng annotation như sau:

```
[ExpectedException(typeof(<expected_exception>))]
```

Tạm bỏ qua một test case nào đó không chạy sử dụng annotation như sau:

```
[TestMethod, Ignore]
```