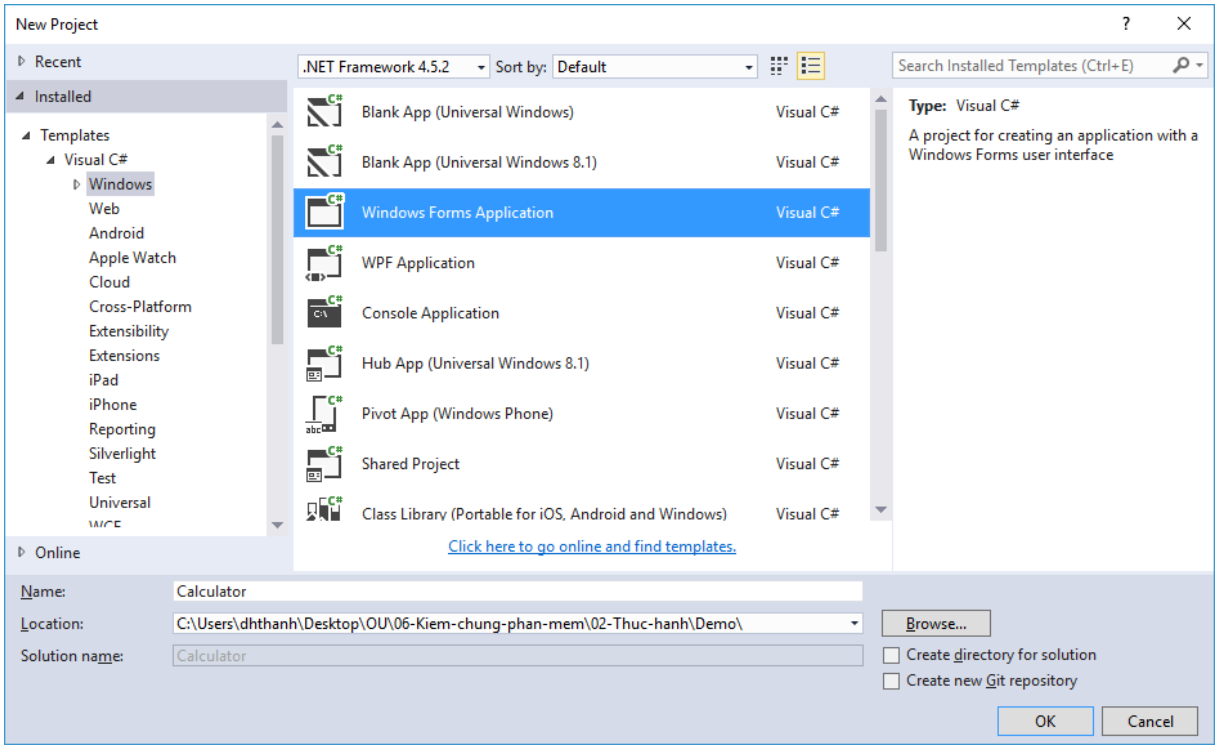
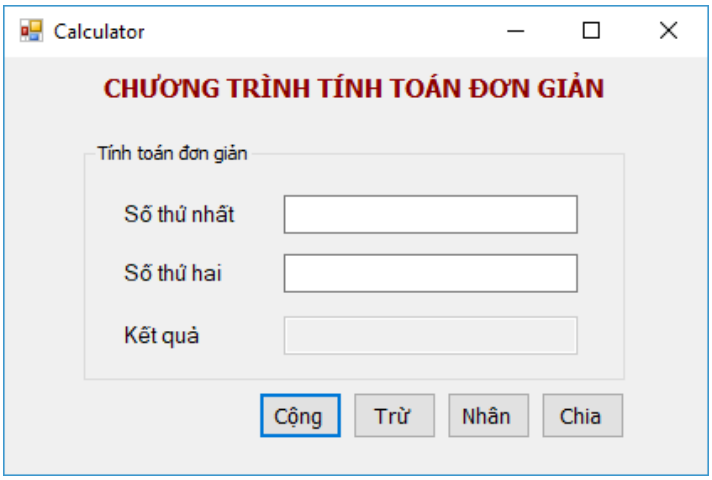
Bài thực hành 3 – Viết Unit Test trong C#

Giả sử tạo một project C# thực hiện các phép toán đơn giản cộng, trừ, nhân, chia các số nguyên, trong đó phép chia yêu cầu kết quả phải làm tròn về số nguyên gần nhất, chẳng hạn 1.2 làm tròn thành 1 và 1.5 hay 1.6 làm tròn thành 2.



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



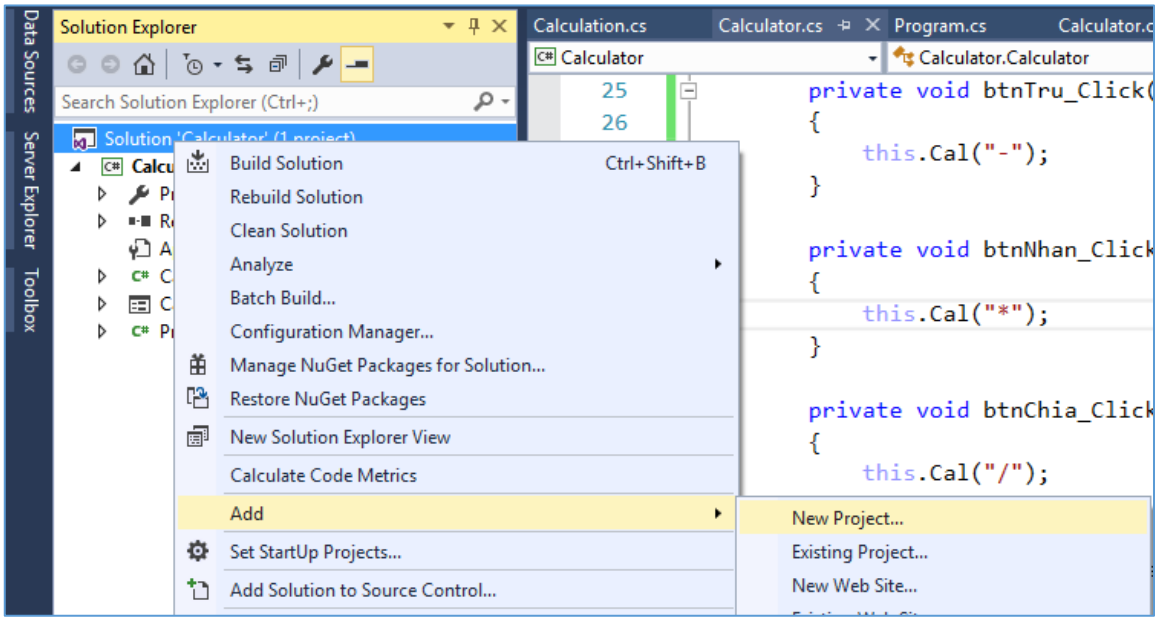
Tạo một tập tin `Calculation.cs` chứa lớp public Calculation dùng có phương thức `Execute` để thực hiện phép tính đơn giản với hai số nguyên.

```
namespace Calculator
{
    public class Calculation
    {
```

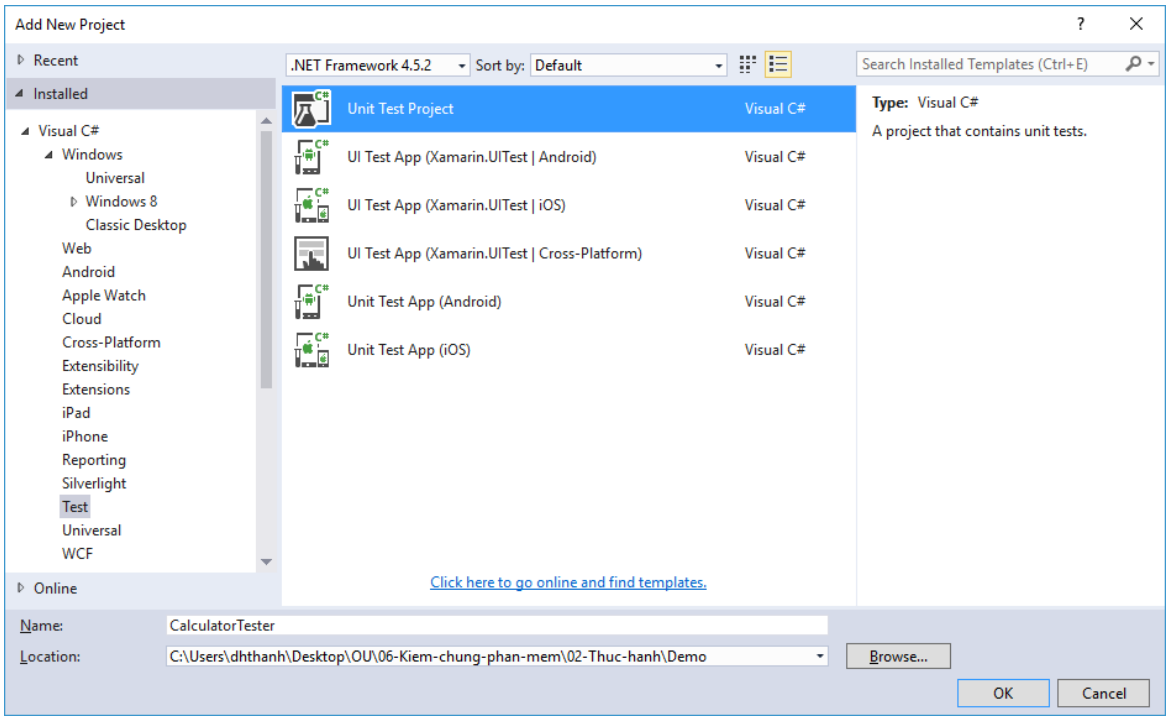
```
        private int a;
        private int b;

        public Calculation(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;
        }
    }
}
```

Tạo project kiểm thử để kiểm thử các phép toán trong chương trình trên.

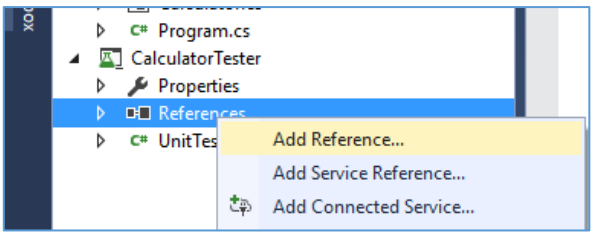
Click chuột phải Solution > Add > New Projects...



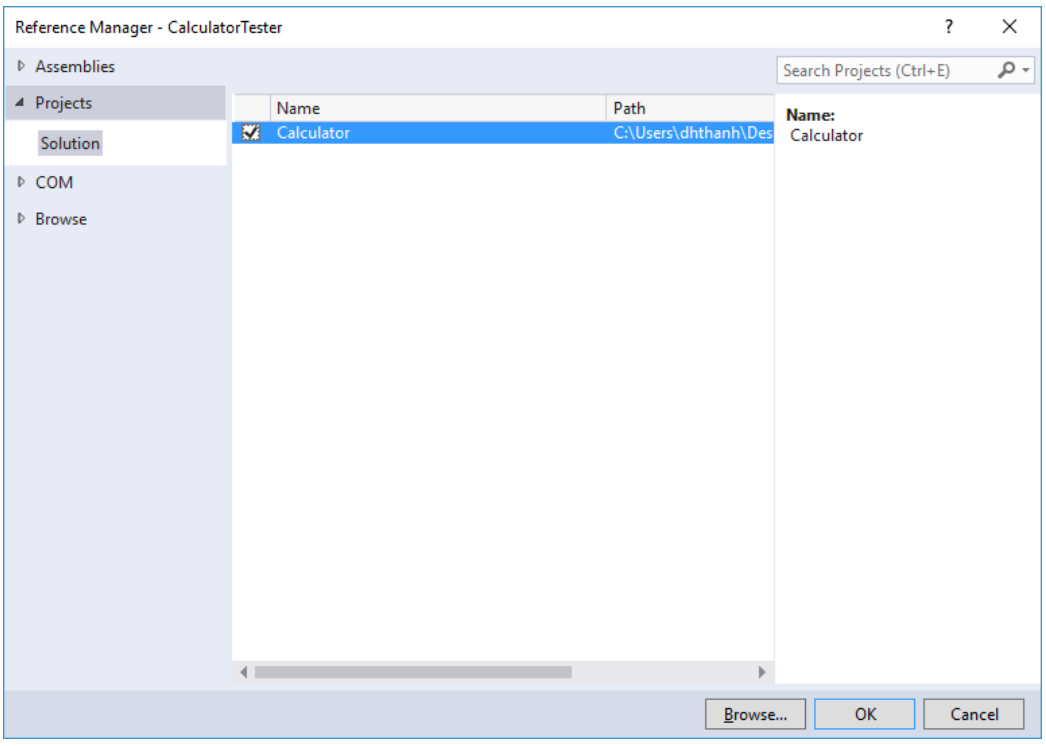
Sau đó chọn loại project là "Unit Test Project" và đặt tên là `CalculatorTester`



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 `Calculation`.

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

namespace CalculatorTester
{
    [TestClass]
    public class UnitTest1
    {
        private Calculation cal;

        [TestInitialize]
        public void SetUp()
        {
            this.cal = new Calculation(10, 5);
        }

        [TestMethod]
        public void TestAddOperator()
        {
            Assert.AreEqual(cal.Execute("+"), 15);
        }

        [TestMethod]
        public void TestSubOperator()
        {
            Assert.AreEqual(cal.Execute("-"), 5);
        }

        [TestMethod]
        public void TestMulOperator()
        {
            Assert.AreEqual(cal.Execute("*"), 50);
        }

        [TestMethod]
        public void TestDivOperator()
        {
            Assert.AreEqual(cal.Execute("/"), 2);
        }

        [TestMethod]
        [ExpectedException(typeof(DivideByZeroException))]
        public void TestDivByZero()
        {
            Calculation c = new Calculation(2, 0);
            c.Execute("/");
        }
    }
}
```

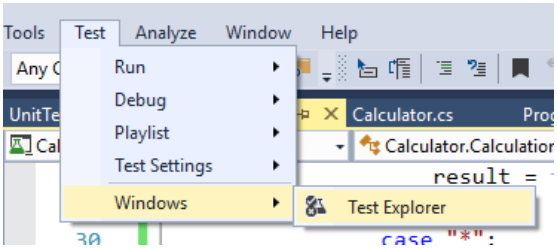
Trong đó:

- `[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.

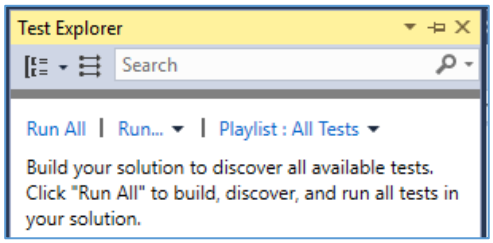
- `[ExpectedException(typeof(DivideByZeroException))]`: kết quả mong muốn là xuất hiện ngoại lệ `DivideByZeroException` khi thực hiện phép chia cho 0.
- `[Timeout]`: thiết lập timeout khi thực thi test case.
- `[Ignore]`: bỏ qua tạm thời test case khi thực thi.
- Các phương thức của `Assert.AreEqual` 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:

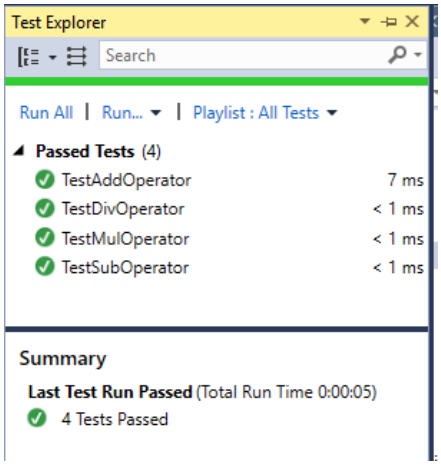
- Mở cửa sổ Test Explorer: menu Test > Windows > Test Explorer



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



- Kết quả chạy các test case như sau:

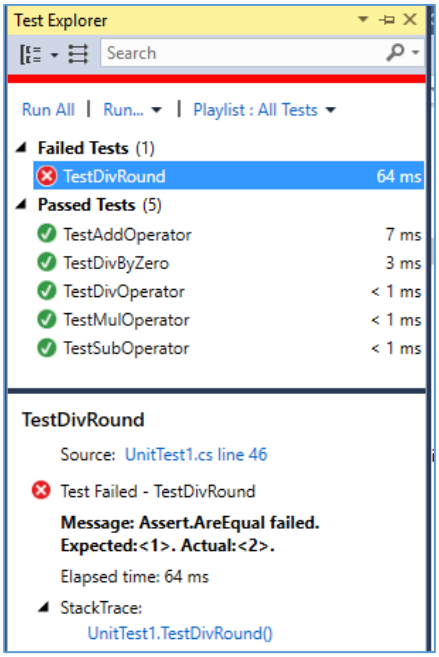


- Thêm 1 test case kiểm tra làm tròn trong kết quả phép chia.

```
[TestMethod]
public void TestDivRound()
{
    Calculation c = new Calculation(5, 3);
    Assert.AreEqual(c.Execute("/"), 2);
```

```
}
```

- Chạy lại tất cả các test case đã viết và kết quả 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:

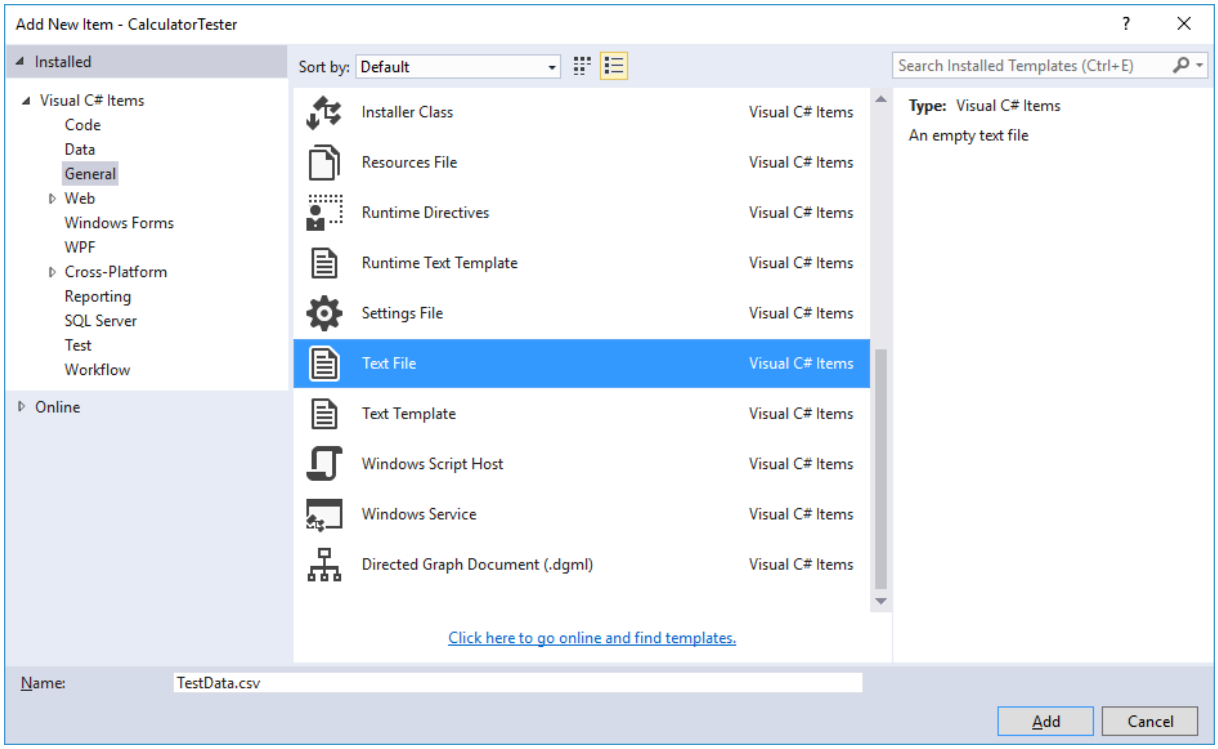
```
[ExpectedException(typeof(<expected_exception>))]
```

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

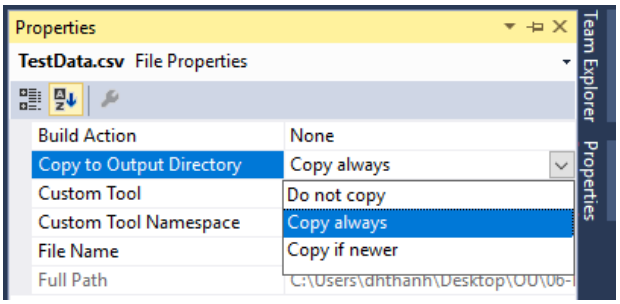
```
[TestMethod, Ignore]
```

DATA-DRIVEN UNIT TEST

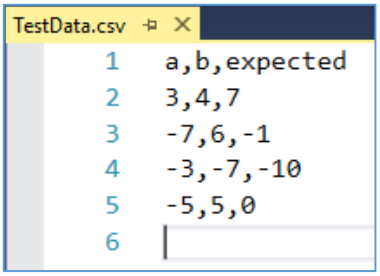
Thực thi test case với các dữ liệu test có sẵn: tạo thư mục `Data` trong project test, click phải chuột vào thư mục `Data > New Item...` tạo tập tin `TestData.csv`.



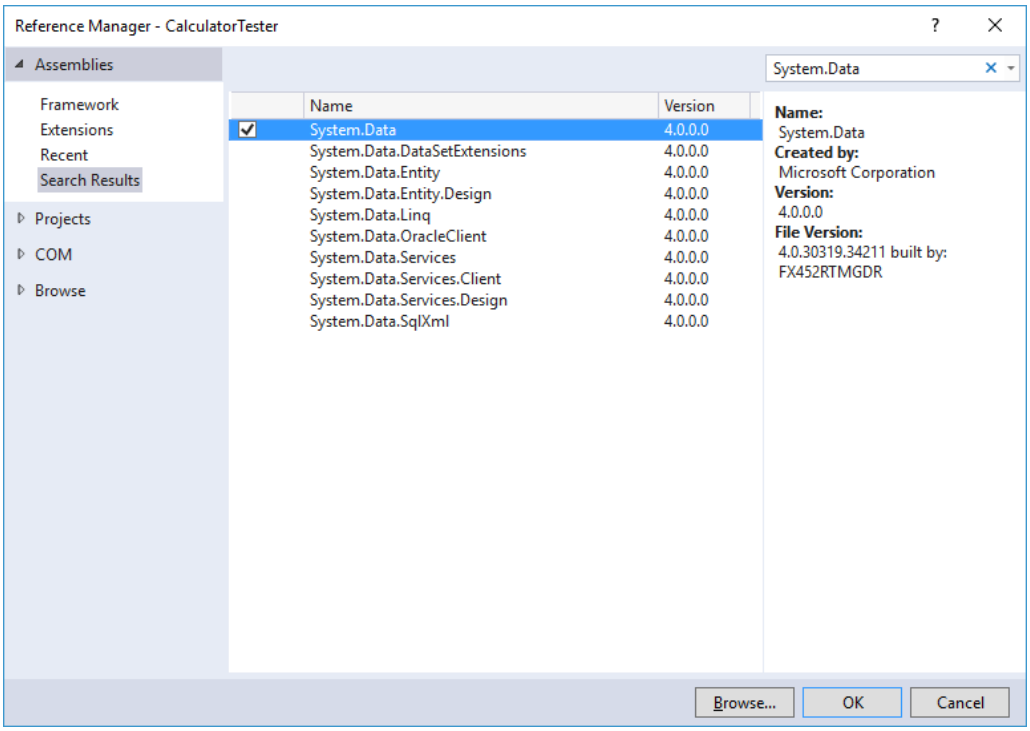
Chuột phải tập tin `TestData.csv` chọn Properties, thiết lập thuộc tính "Copy to Output Directory" thành "Copy always" để tập tin này sẽ được sao chép vào thư mục `bin` khi build project.



Nhập dữ liệu vào tập tin `TestData.csv` như sau:



Thêm reference `System.Data` vào project test



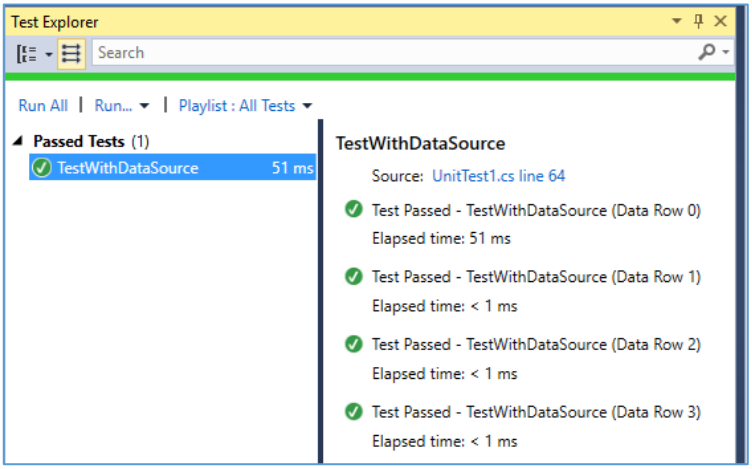
Tạo đối tượng `TestContext` trong lớp unit test như sau:

```
public TestContext TestContext { get; set; }
```

Viết test case sử dụng dữ liệu này chạy các test case như sau:

```
[DataSource("Microsoft.VisualStudio.TestTools.DataSource.CSV",
            @".\Data\TestData.csv", "TestData#csv", DataAccessMethod.Sequential)]
[TestMethod]
public void TestWithDataSource()
{
    int a = int.Parse(TestContext.DataRow[0].ToString());
    int b = int.Parse(TestContext.DataRow[1].ToString());
    int expected = int.Parse(TestContext.DataRow[2].ToString());

    Calculation c = new Calculation(a, b);
    int actual = c.Execute("+");
    Assert.AreEqual(expected, actual);
}
```

Thực thi test case trên sẽ có kết quả như sau:



Bài 1: Viết unit test kiểm tra hàm tính x^n bằng đệ quy, với x là số thực và n là số nguyên bất kỳ, biết x^n được tính như sau:

$$x^n = \begin{cases} 1 & nếu\ n = 0 \\ x^{n-1} \times x^n & nếu\ n > 0 \\ \dfrac{x^{n+1}}{x} & nếu\ n < 0 \end{cases}$$

```
static double Power(float x, float n)
{
    if (n == 0)
        return 1.0;
    else if (n > 0)
        return n * Power(x, n - 1);
    else
        return Power(x, n + 1) / x;
}
```

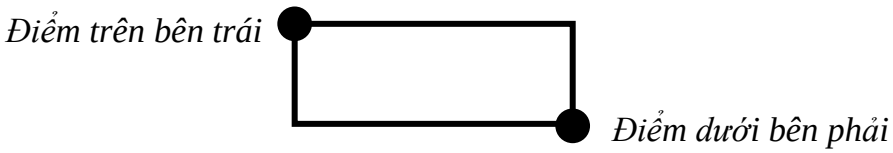
Bài 2: Viết unit test kiểm tra chương trình tính giá trị đa thức tại một giá trị x nào đó. Chương trình nhập vào số nguyên n là bậc đa thức và $n + 1$ số nguyên a_i ($0 \le a_i \le n$), với a_i là hệ số của x^i và giá trị biến nguyên x và tính giá trị biểu thức đó.

```
class Polynomial
{
    private int n;
    private List<int> a;

    public Polynomial(int n, List<int> a)
    {
        if (a.Count() != n + 1)
            throw new ArgumentException("Invalid Data");

        this.n = n;
        this.a = a;
    }

    public int Cal(double x)
    {
        int result = 0;
        for (int i = 0; i <= this.n; i++)
        {
            result += (int)(a[i] * Math.Pow(x, i));
        }

        return result;
    }
}
```

Bài 3: Viết unit test kiểm tra chương trình chuyển đổi số nguyên dương cơ số 10 sang cơ số k bất kỳ ($2 \le k \le 16$).

```
public class Radix
{
    private int number;

    public Radix(int number)
    {
        if (number < 0)
            throw new ArgumentException("Incorrect Value");

        this.number = number;
    }

    public string ConvertDecimalToAnother(int n, int radix = 2)
    {
        if (radix < 2 || radix > 16)
            throw new ArgumentException("Invalid Radix");

        List<string> result = new List<string>();
        while (n > 0)
        {
            int value = n % radix;
            if (value < 10)
                result.Add(value.ToString());
            else
            {
                switch (value)
                {
                    case 10: result.Add("A"); break;
                    case 11: result.Add("B"); break;
                    case 12: result.Add("C"); break;
                    case 13: result.Add("D"); break;
                    case 14: result.Add("E"); break;
                    case 15: result.Add("F"); break;
                }
            }
            n /= radix;
        }

        result.Reverse();
        return String.Join("", result.ToArray());
    }
}
```

Bài 4: Viết lớp `Diem` để thao tác với điểm trong không gian hai chiều bao gồm hai thuộc tính hoành độ và tung độ. Lớp `HinhChuNhat` chứa thông tin của một hình chữ nhật, biết một hình chữ nhật được xác định bởi 2 điểm là tọa độ điểm trên bên trái và tọa độ điểm dưới bên phải:



Lớp `HinhChuNhat` có hai phương thức thực hiện các chức năng sau:

- Tính diện tích hình chữ nhật
- Kiểm tra hai hình chữ nhật có giao nhau hay không?

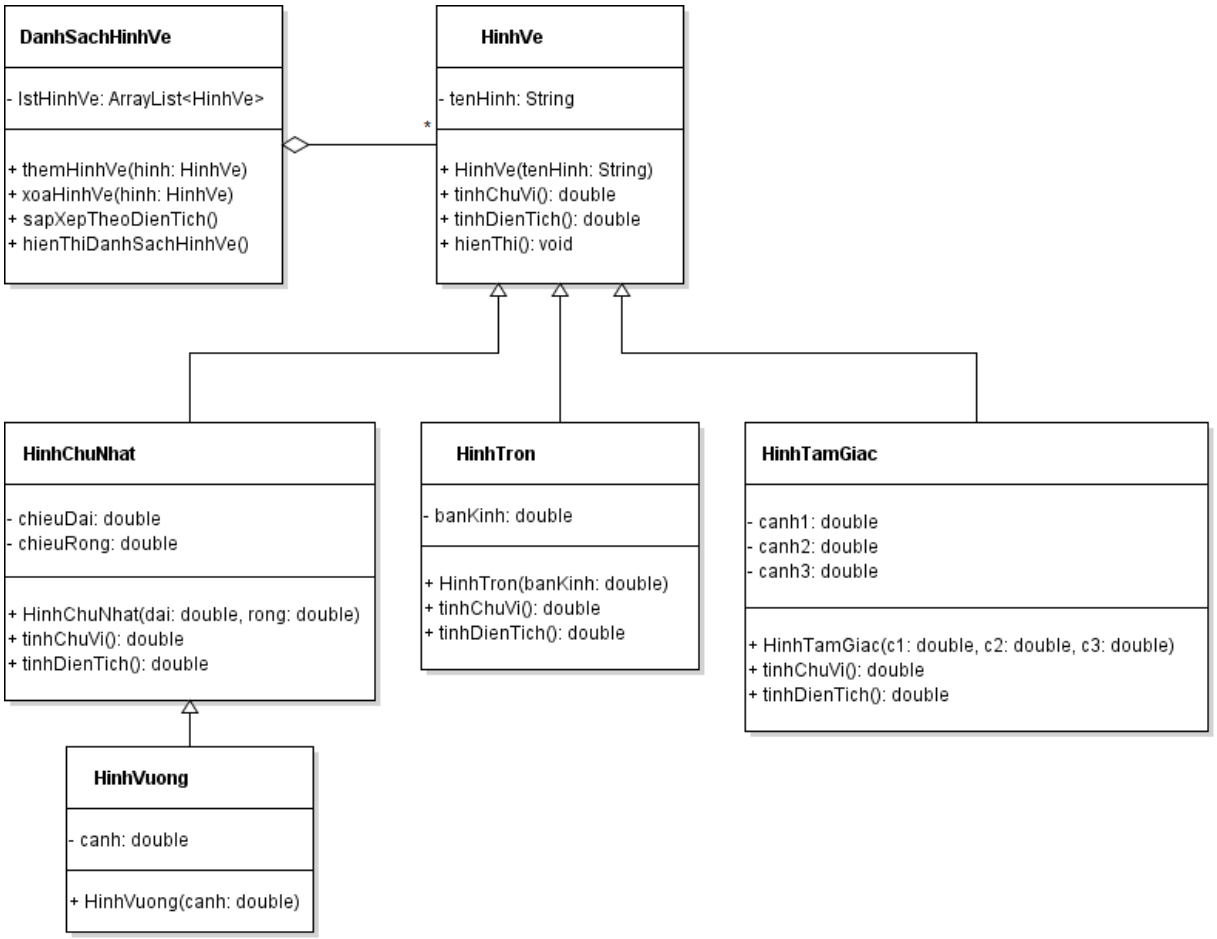
Viết unit test để kiểm tra các chức năng của chương trình trên.

Bài 5: Một trung tâm gia sư cần quản lý thông tin học viên, một học viên bao gồm thông tin: mã số học viên, họ tên, quê quán, điểm của ba môn học chính. Vào cuối khoá học, trung tâm muốn tìm ra một số học viên có thành tích học tập tốt để trao học bổng khuyến khích. Một học viên được đánh giá là tốt nếu điểm trung bình ba môn học chính từ 8.0 trở lên và không có môn nào trong ba môn chính điểm dưới 5.

Viết chương trình cho phép nhập danh sách học viên và xác định danh sách học viên có thể nhận học bổng.

Viết unit test để kiểm thử các chức năng của chương trình trên.

Bài 6: Viết chương trình hiện thực hóa sơ đồ lớp bên dưới, bao gồm tính diện tích, chu vi của một hình bất kỳ, cụ thể trong sơ đồ lớp là hình chữ nhật, hình vuông, hình tròn, hình tam giác.



Viết unit test để kiểm tra các chức năng trong chương trình trên.