

Viết Unit Test

PHẦN 1. JUNIT (JAVA)

JUnit là một framework để viết unit test cho ngôn ngữ Java, đóng vai trò quan trọng trong các ứng dụng test-driven. Để cài đặt môi trường cho JUnit, đầu tiên cần cài đặt bộ JDK (từ 1.5 trở lên), sau đó download tập tin .jar của JUnit tại <https://junit.org/junit4/>.

Giả sử ta có lớp PhanSo và cần viết test case kiểm tra phương thức cộng hai phân số.

```
public class PhanSo {
    private int tuSo;
    private int mauSo;

    public PhanSo(int tu, int mau) {
        if (mau == 0)
            throw new InputMismatchException("Error!!!");

        this.tuSo = tu;
        this.mauSo = mau;
    }

    public PhanSo cong(PhanSo p) {
        int t = this.tuSo * p.mauSo + this.mauSo * p.tuSo;
        int m = this.mauSo * p.mauSo;

        return new PhanSo(t, m);
    }

    // Các phương thức getter và setter
}
```

Viết lớp test case.

```
public class Tester {
    private PhanSo p1 = new PhanSo(1, 2);
    private PhanSo p2 = new PhanSo(2, 3);

    @Test(expected = InputMismatchException.class)
    public void testError() {
        PhanSo ps = new PhanSo(1, 0);
    }

    @Test
    public void testAddFraction1() {
        PhanSo actual = p1.cong(p2);
        PhanSo expected = new PhanSo(7, 6);

        Assert.assertEquals(expected.getTuSo(),
                               actual.getTuSo());
        Assert.assertEquals(expected.getMauSo(),
                               actual.getMauSo());
    }
}
```

Thực thi các test case.

```
public class TestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(Tester.class);

        if (!result.wasSuccessful()) {
            result.getFailures().forEach((failure) -> {
                System.err.println(failure);
            });
        } else {
            System.out.println("success");
        }
    }
}
```

Một số lớp quan trọng để viết unit test trong JUnit

Lớp `org.junit.Assert` cung cấp các phương thức assertion hữu dụng cho viết unit test, các phương thức thông dụng

- o `assertEquals()`: kiểm tra hai đối tượng/giá trị bằng nhau.
- o `assertFalse()`: kiểm tra biểu thức điều kiện là false.
- o `assertTrue()`: kiểm tra biểu thức điều kiện là true.
- o `assertNotNull()`: kiểm tra một đối tượng khác null.
- o `assertNull()`: kiểm tra một đối tượng là null.
- o `assertSame()`: kiểm tra hai đối tượng tham chiếu đến cùng đối tượng không.
- o `assertNotSame()`: kiểm tra hai đối tượng không trỏ đến cùng đối tượng.
- o `assertArrayEquals()`: kiểm tra hai mảng có bằng nhau không.

Lớp `org.junit.TestResult` tập hợp các kết quả thực thi test case.

Các Annotation

- o `@Test`: dùng cho phương thức `public void`, để chỉ định một test case.
 - Tham số `timeout`: chỉ định khoảng thời gian (mili) tối đa thực hiện test case, nếu quá thời gian này, test case được đánh dấu là fail.
 - Tham số `expected`: chỉ định lớp ngoại lệ muốn được ném ra khi thực thi phương thức test case.
- o `@Before`: dùng cho phương thức `public void`, chỉ định phương thức chạy trước mỗi phương thức test case.
- o `@After`: dùng cho phương thức `public void`, chỉ định phương thức chạy sau mỗi phương thức test case.

- `@BeforeClass`: dùng cho phương thức `public static void`, chỉ định phương thức chạy một lần và trước tất cả các phương thức test case trong lớp.
- `@AfterClass`: chỉ định phương thức chạy một lần và sau khi tất cả các phương thức test case hoàn tất.
- `@Ignore`: chỉ định bỏ qua một phương thức khi thực thi test case.

```
public class DemoTester {
    @BeforeClass
    public static void start() {
        System.out.println("Start");
    }

    @Before
    public void startTest() {
        System.out.println("Start Test case");
    }

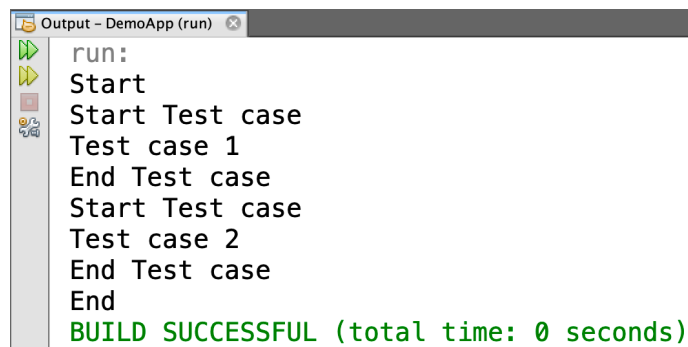
    @Test
    public void test1() {
        System.out.println("Test case 1");
    }

    @Test
    public void test2() {
        System.out.println("Test case 2");
    }

    @After
    public void endTest() {
        System.out.println("End Test case");
    }

    @AfterClass
    public static void end() {
        System.out.println("End");
    }
}
```

Thực thi test này



```
Output - DemoApp (run)
run:
Start
Start Test case
Test case 1
End Test case
Start Test case
Test case 2
End Test case
End
BUILD SUCCESSFUL (total time: 0 seconds)
```

Lớp `org.junit.TestSuite` để thực thi nhiều test cùng lúc.

- `addTest()`: thêm test mới vào suite
- `addTestSuite()`: thêm cùng lúc nhiều test vào suite.

- o `countTestCases()`: đếm số lượng test case được chạy.
- o `getName()`: lấy tên của suite.
- o `setName()`: thiết lập tên cho suite.
- o `run()`: thực thi các test và ghi kết quả vào `TestResult`.
- o `testAt()`: trả về test ở vị trí chỉ định.
- o `testCount()`: đếm số lượng test trong suite.

Lớp trừu tượng `org.junit.TestCase` định nghĩa fixture để chạy nhiều test case, các phương thức thông dụng

- o `setUp()`:
- o `tearDown()`:
- o `run(TestResult tests)`: thực thi các test case và ghi kết quả vào `TestResult`.
- o `countTestCases()`: đếm số lượng test case được thực thi bởi hàm `run()`.
- o `getName()`: lấy tên test case.
- o `setName()`: thiết lập tên cho test case.

Parameterized Tests cho phép chạy test giống nhau với các giá trị khác nhau được thực hiện một cách dễ dàng. Giả sử ta có lớp kiểm tra số nguyên tố sau cần viết các test case kiểm thử.

```
public class PrimeValidation {
    public boolean check(int n) {
        if (n >= 2) {
            for (int i = 2; i <= Math.sqrt(n); i++)
                if (n % i == 0)
                    return false;

            return true;
        }

        return false;
    }
}
```

Lớp Test được thiết kế như sau

```
@RunWith(Parameterized.class)
public class PrimeTester {
    private int n;
    private boolean expected;
    private PrimeValidation prime;

    public PrimeTester(int n, boolean expected) {
        this.n = n;
        this.expected = expected;
    }
}
```

```
@Parameterized.Parameters
public static Collection testData() {
    return Arrays.asList(new Object[][] {
        {1, false}, {2, true}, {4, false},
        {7, true}, {16, false}, {19, true}
    });
}

@Before
public void init() {
    this.prime = new PrimeValidation();
}

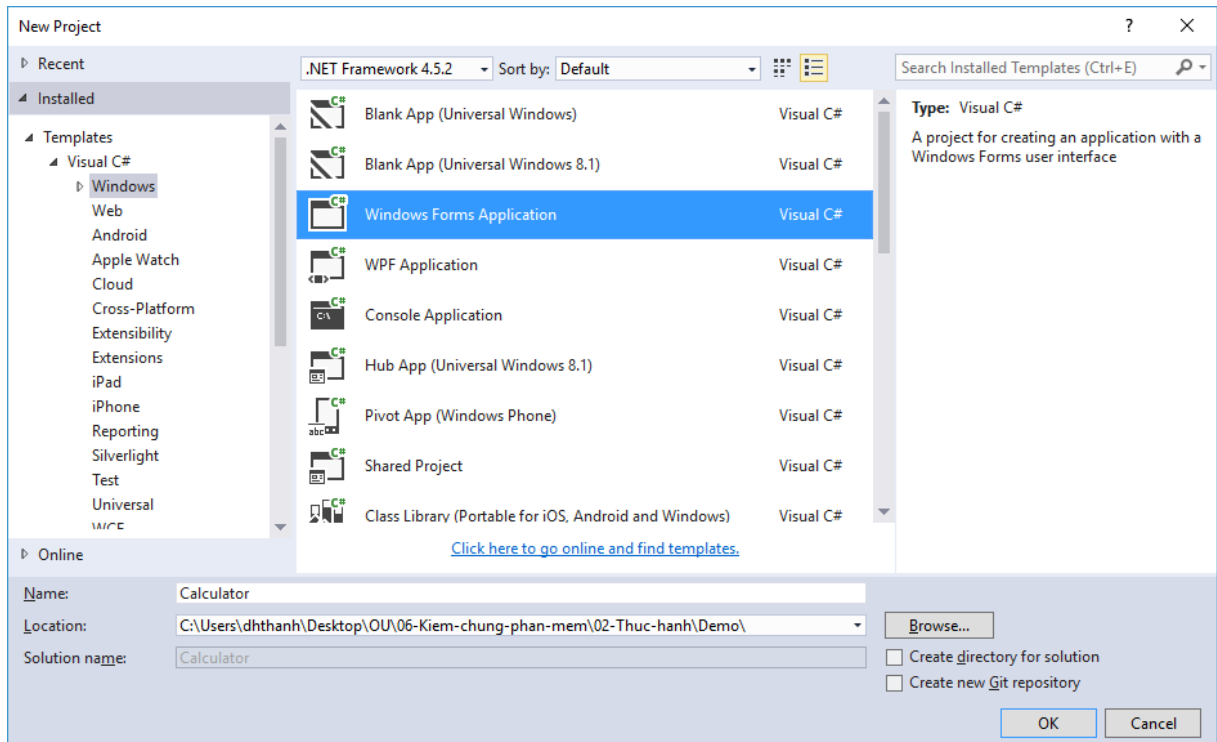
@Test
public void testCase() {
    Assert.assertEquals(this.expected,
        this.prime.check(this.n));
}
}
```

Trong đó bắt buộc:

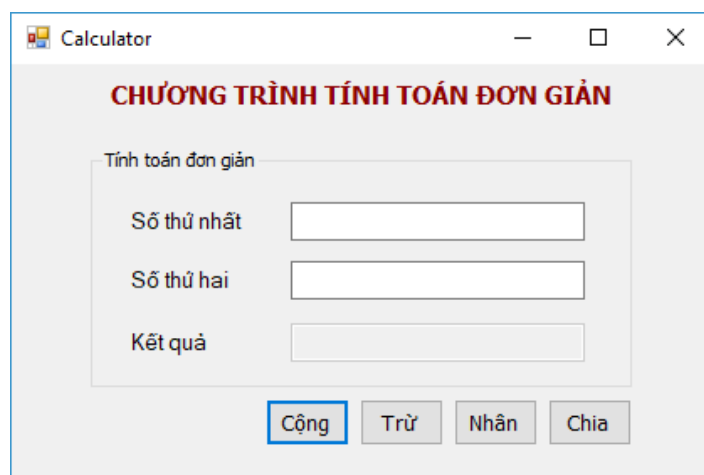
- Gắn annotation `@RunWith(Parameterized.class)` vào lớp test.
- Phương thức tĩnh `testData()` có annotation `@Parameters` trả về một Collection các đối tượng đại diện cho dữ liệu kiểm thử.
- Phương thức khởi tạo public với số tham số tương đương với một dòng trong dữ liệu kiểm thử của phương thức `testData()`.
- Khai báo các thuộc tính của lớp tương ứng với một cột trong mỗi dòng dữ liệu của test data.
- Phương thức test case.

PHẦN 2. 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, 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 Calculation.



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;
}

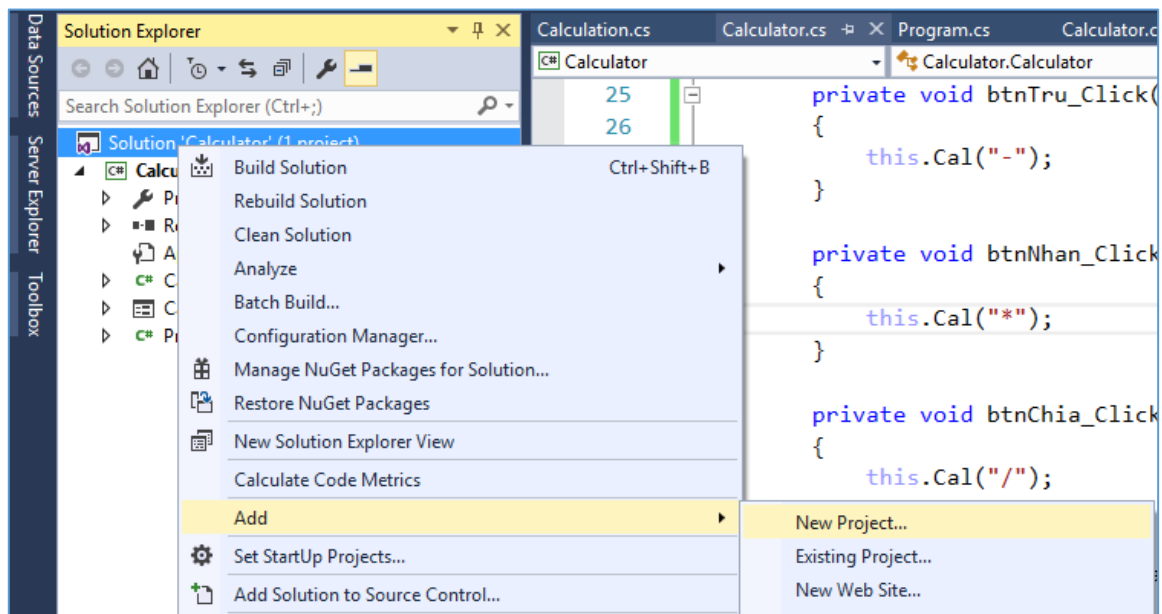
public int Execute(string CalSymbol)
{
    int result = 0;
    switch (CalSymbol)
    {
        case "+":
            result = this.a + this.b;
            break;
        case "-":
            result = this.a - this.b;
            break;
        case "*":
            result = this.a * this.b;
            break;
        case "/":
            result = this.a / this.b;
            break;
    }

    return result;
}
}

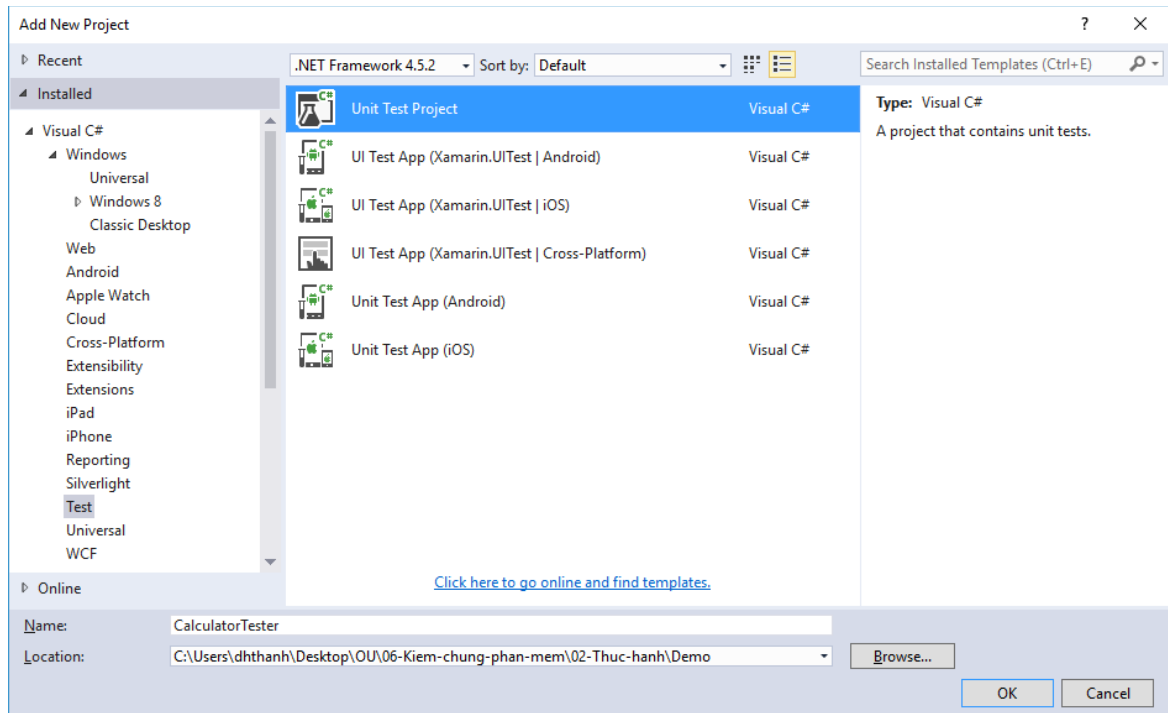
```

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

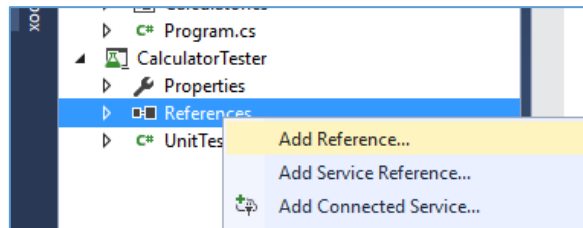
Click chuột phải Solution > Add > New Projects...



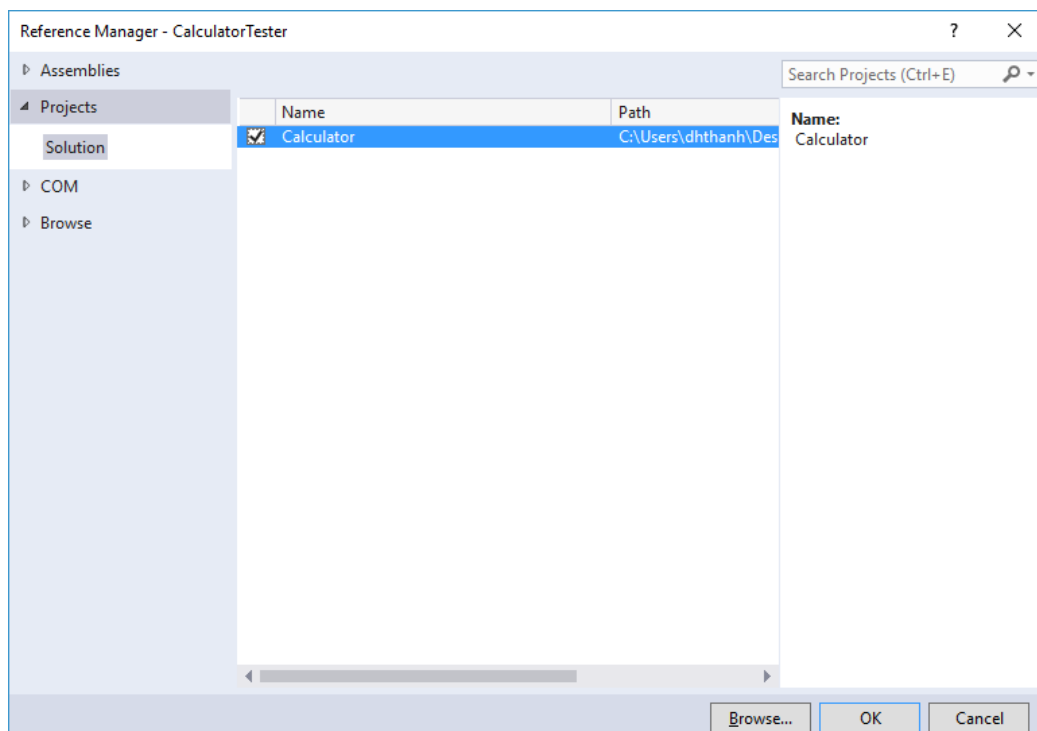
Sau đó chọn loại project là “Unit Test Project” và đặt tên là CalculatorTester



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("/");
        }
    }
}
```

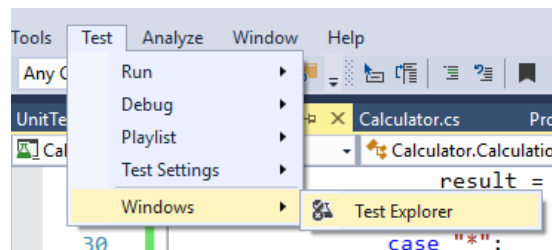
Trong đó:

- `[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.

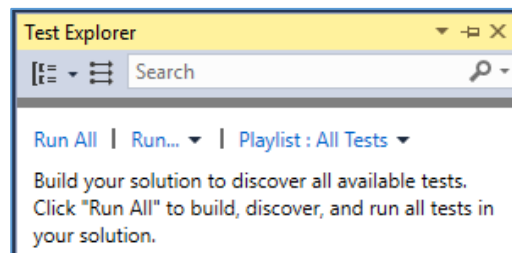
- `[ExpectedException(typeof(DivideByZeroException))]`: kết quả mong muốn là xuất hiện ngoại lệ `DivideByZeroException` khi thực hiện phép chia cho 0.
- `[Timeout]`: thiết lập timeout khi thực thi test case.
- `[Ignore]`: bỏ qua tạm thời test case khi thực thi.
- Các phương thức của `Assert.AreEqual`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

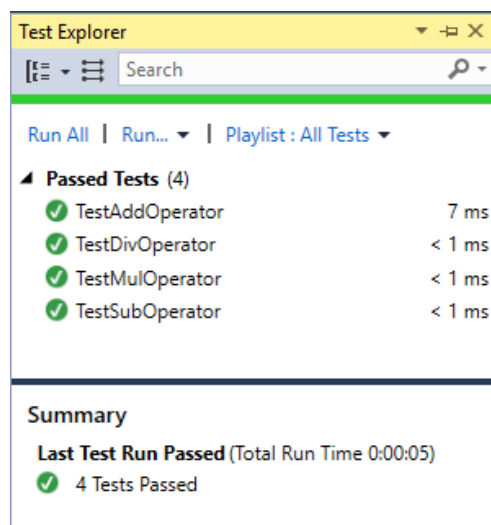
- Mở cửa sổ Test Explorer: menu `Test > Windows > Test Explorer`



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



- Kết quả chạy các test case như sau:

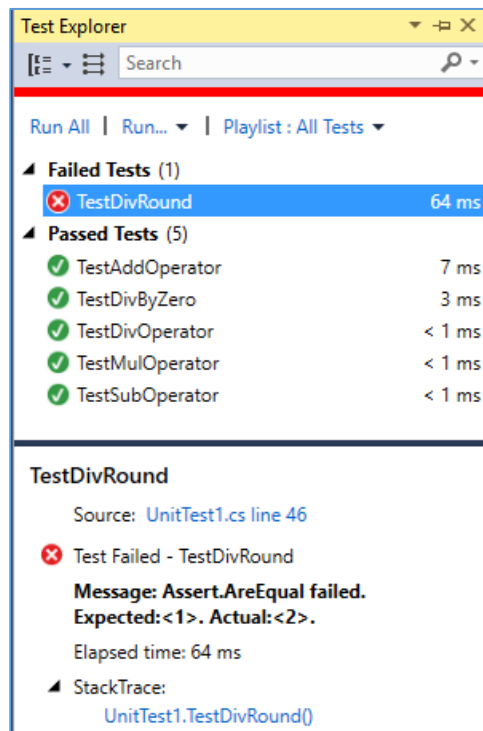


- Thêm 1 test case kiểm tra làm tròn trong kết quả phép chia.

```
[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:

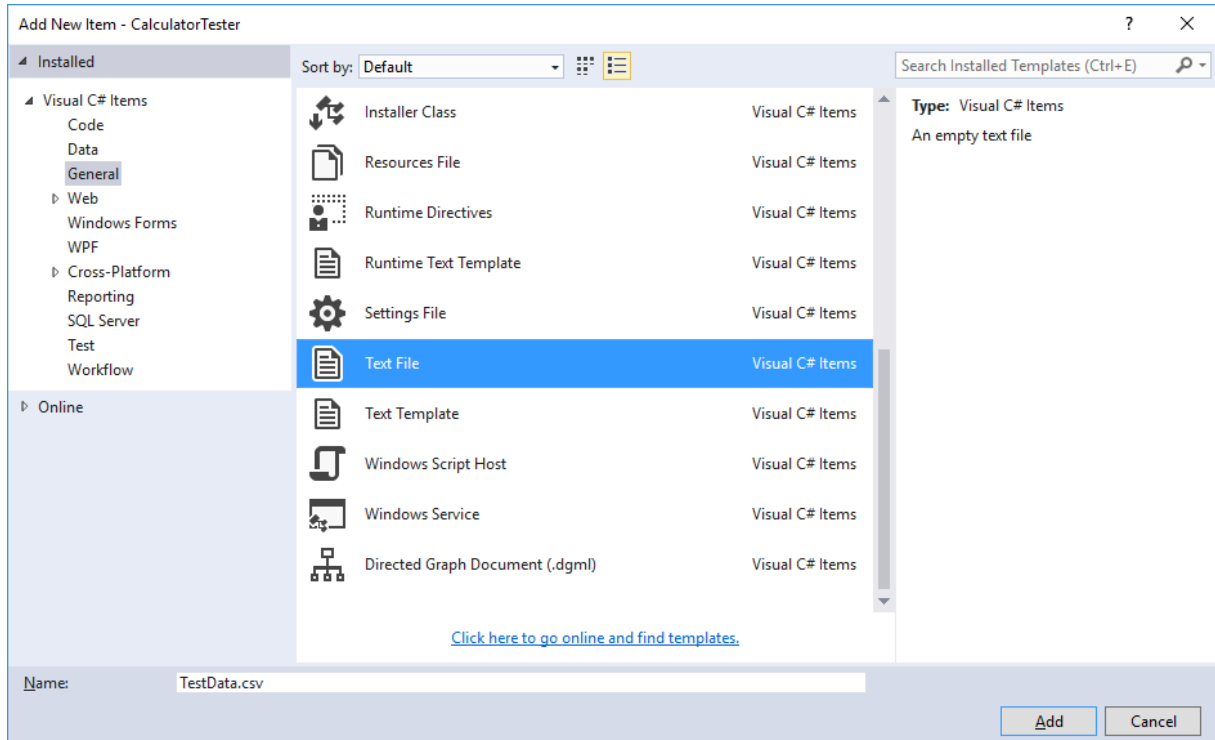
<code>[ExpectedException(typeof(<expected_exception>))]</code>
--

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

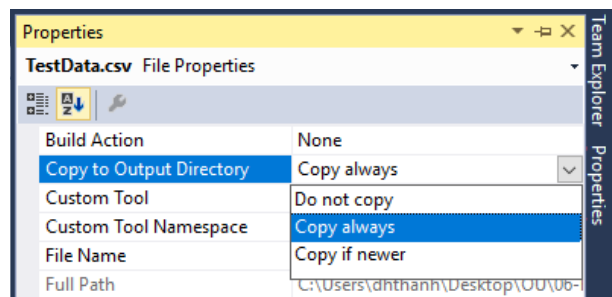
<code>[TestMethod, Ignore]</code>

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.



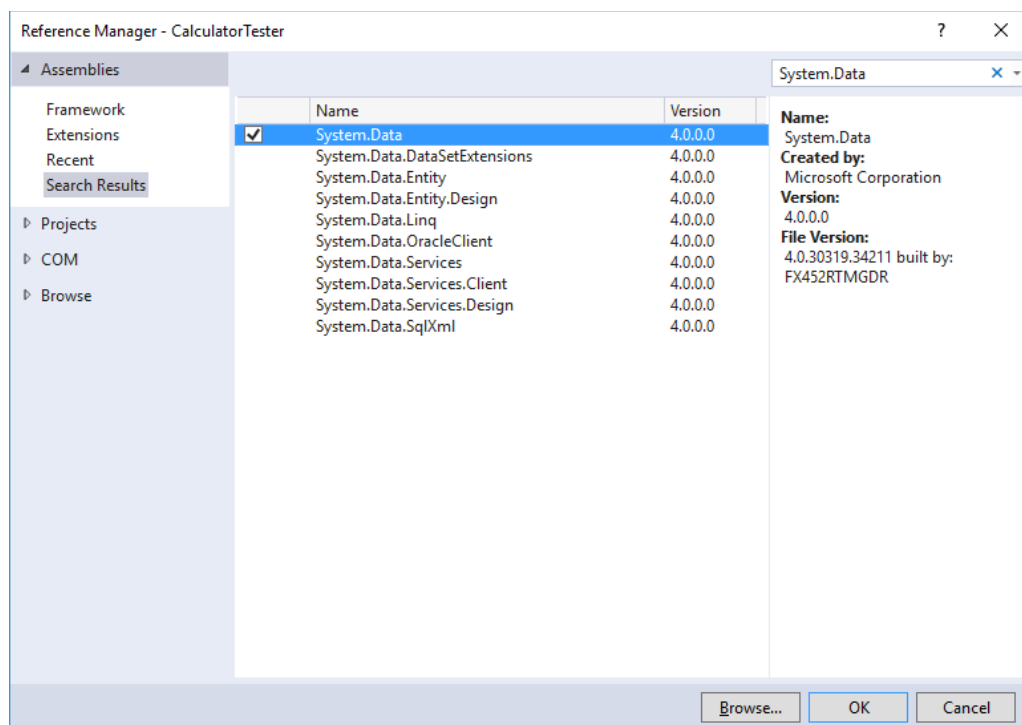
Chuột phải tập tin TestData.csv chọn Properties, thiết lập thuộc tính “Copy to Output Directory” thành “Copy always” để tập tin này sẽ được sao chép vào thư mục bin khi build project.



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

TestData.csv	
1	a,b,expected
2	3,4,7
3	-7,6,-1
4	-3,-7,-10
5	-5,5,0
6	

Thêm reference System.Data vào project test



Tạo đối tượng TestContext trong lớp unit test như sau:

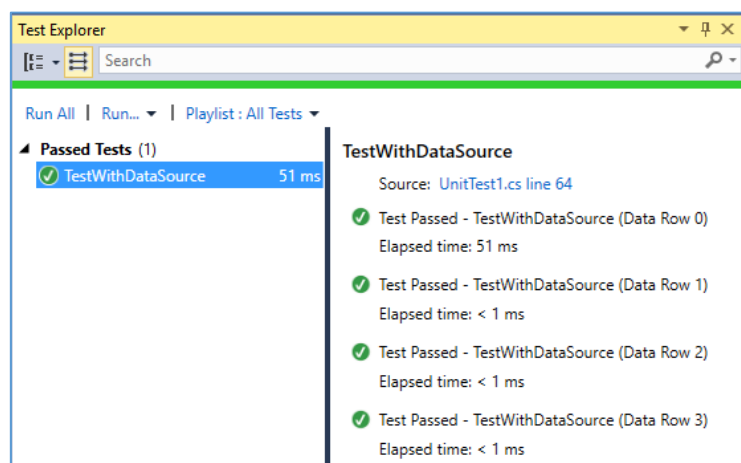
```
public TestContext TestContext { get; set; }
```

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());

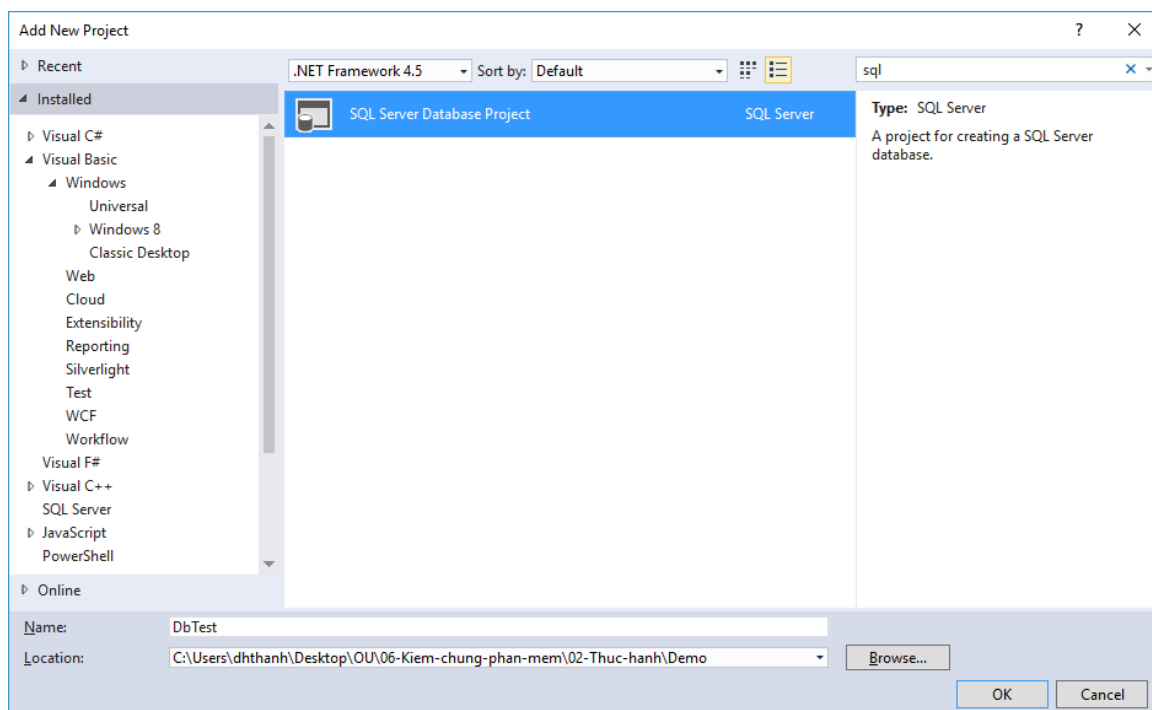
    Calculation c = new Calculation(a, b);
    int actual = c.Execute("+");
    Assert.AreEqual(expected, actual);
}
```

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

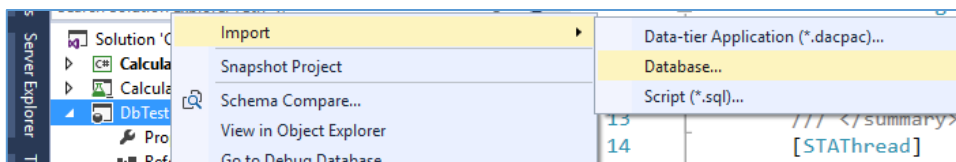


SQL SERVER UNIT TEST

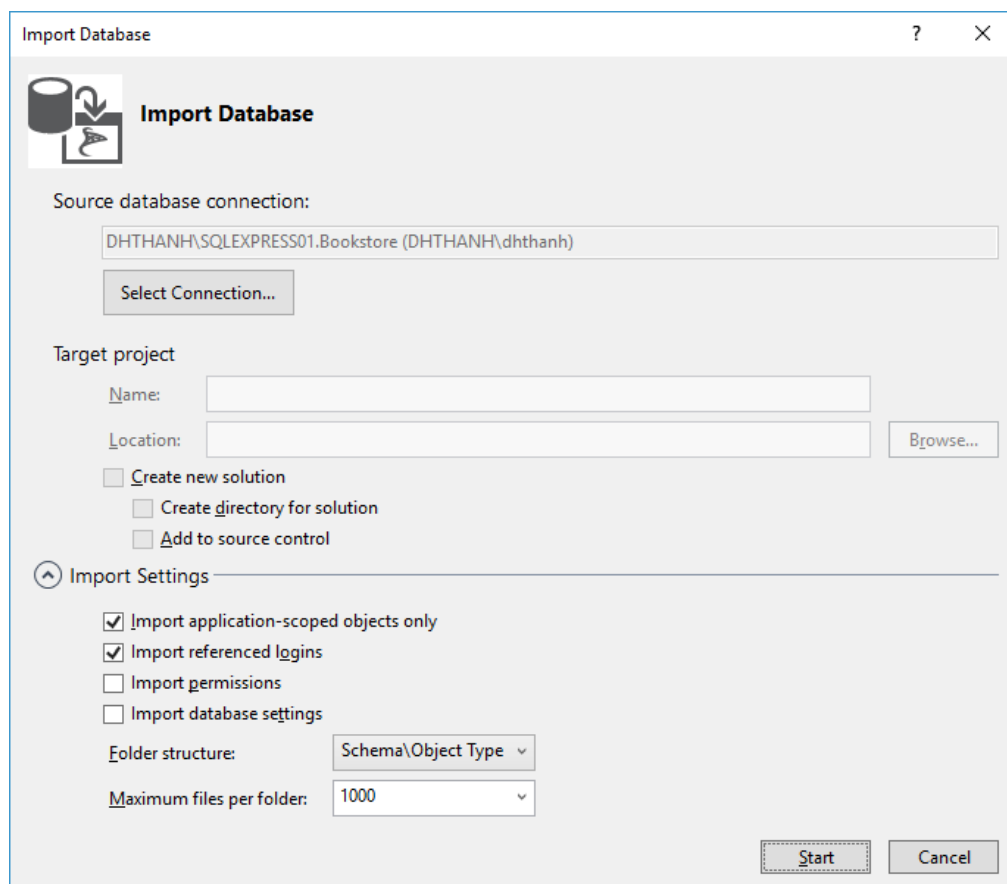
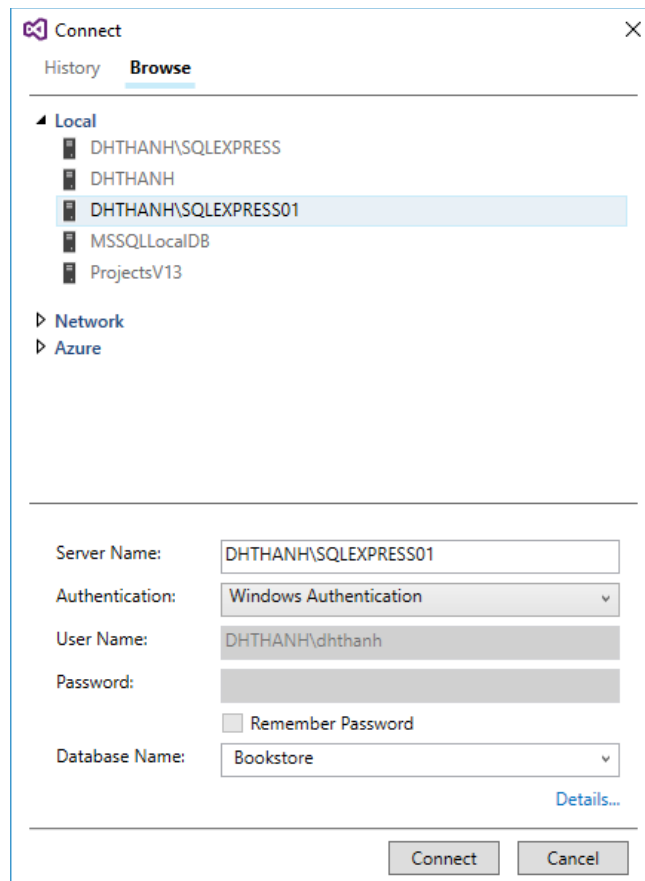
Tạo project “SQL Server Database Project” trước khi viết Unit Test.



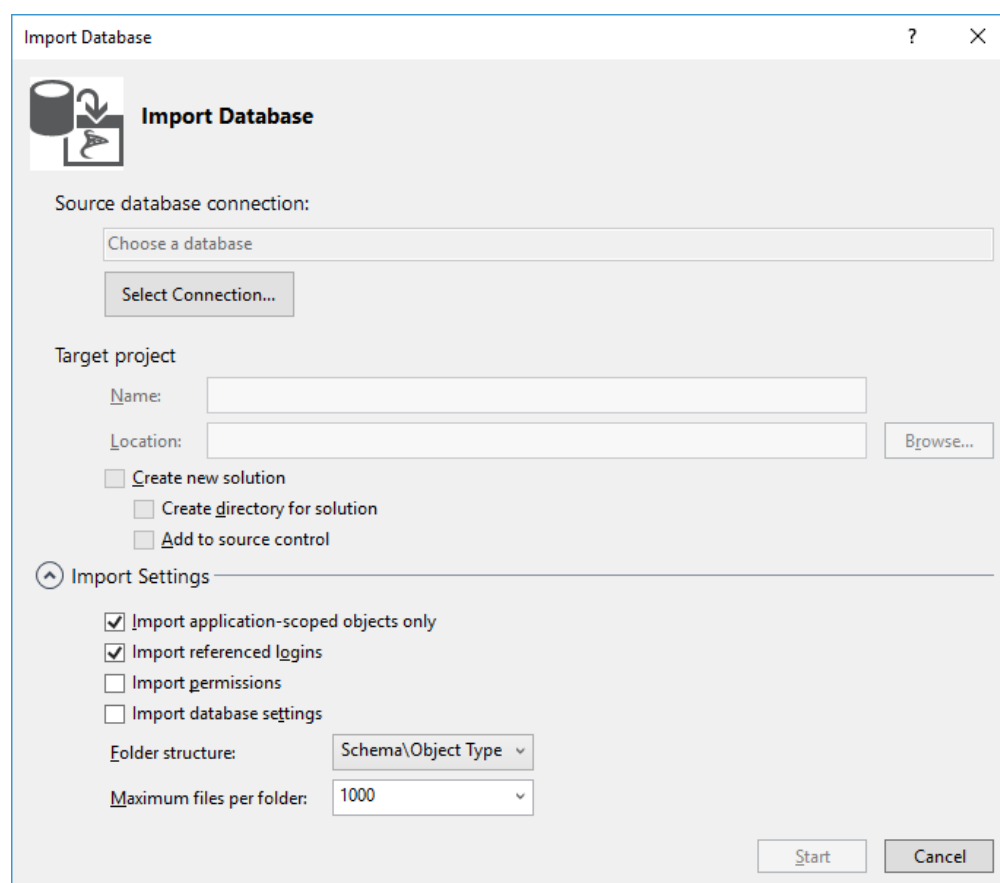
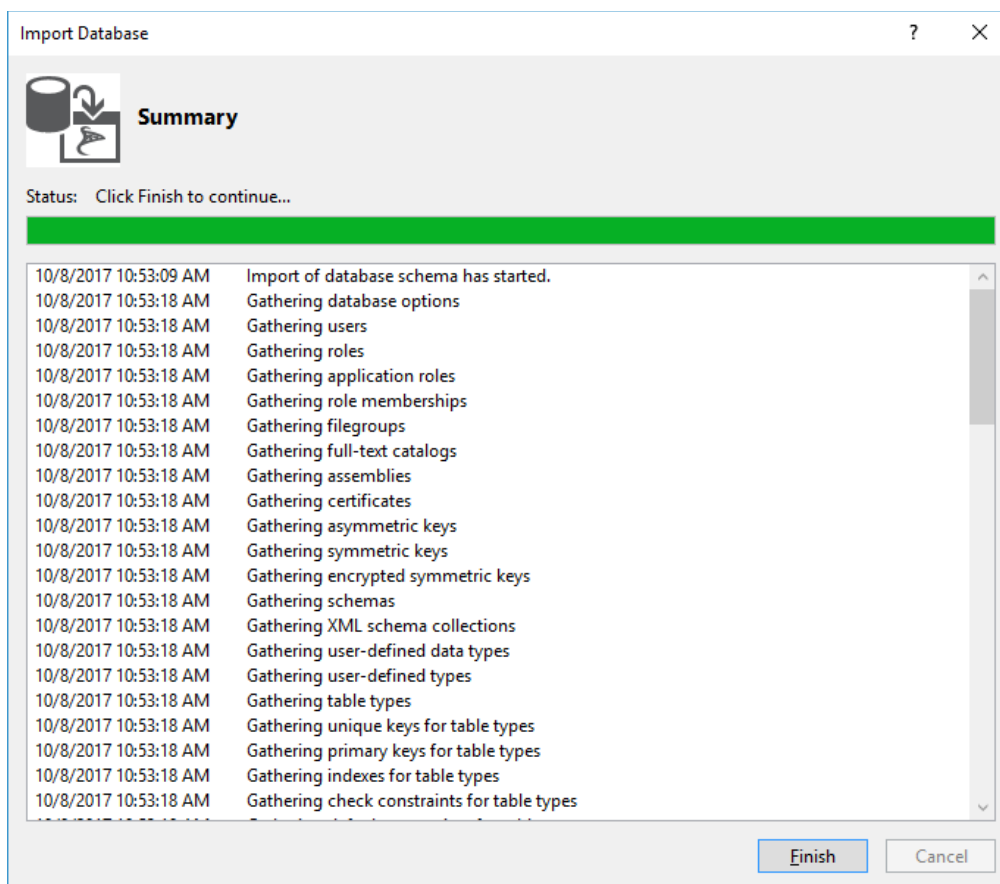
Nạp CSDL vào project: click chuột phải vào project CSDL đã tạo, chọn Import > Database...



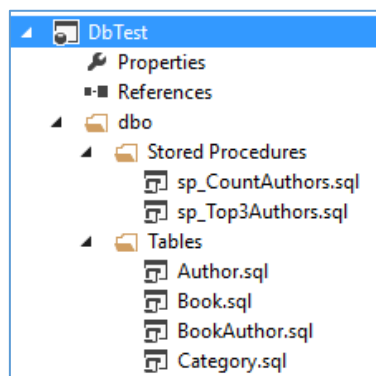
Một cửa sổ “Import Database” hiện ra, click nút “Select Connection...” để chọn CSDL kết nối.



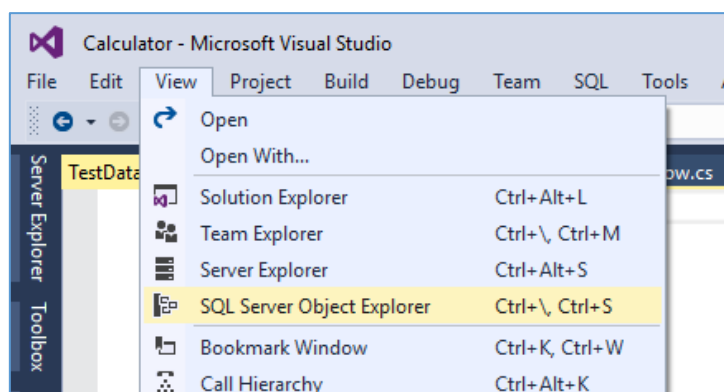
Cơ sở dữ liệu được import vào như sau:



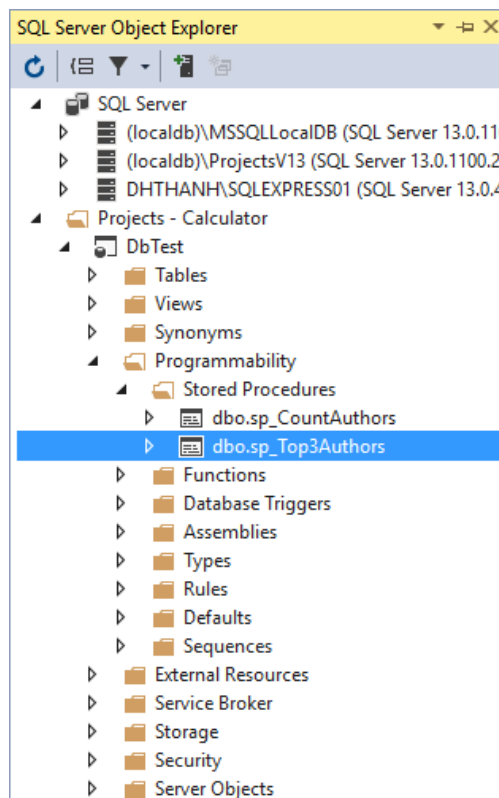
Sau khi import CSDL thì project SQL Server đã tạo sẽ có cấu trúc như sau:



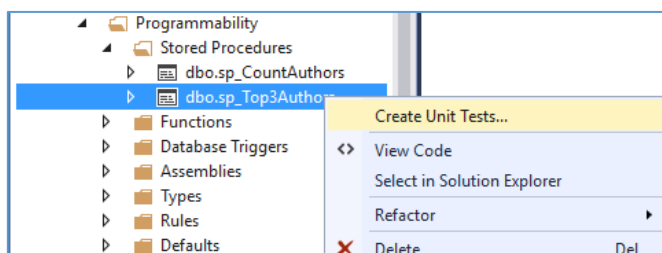
Tạo unit test cho CSDL, mở cửa sổ SQL Server Object Explorer: VIEW > SQL Server Object Explorer



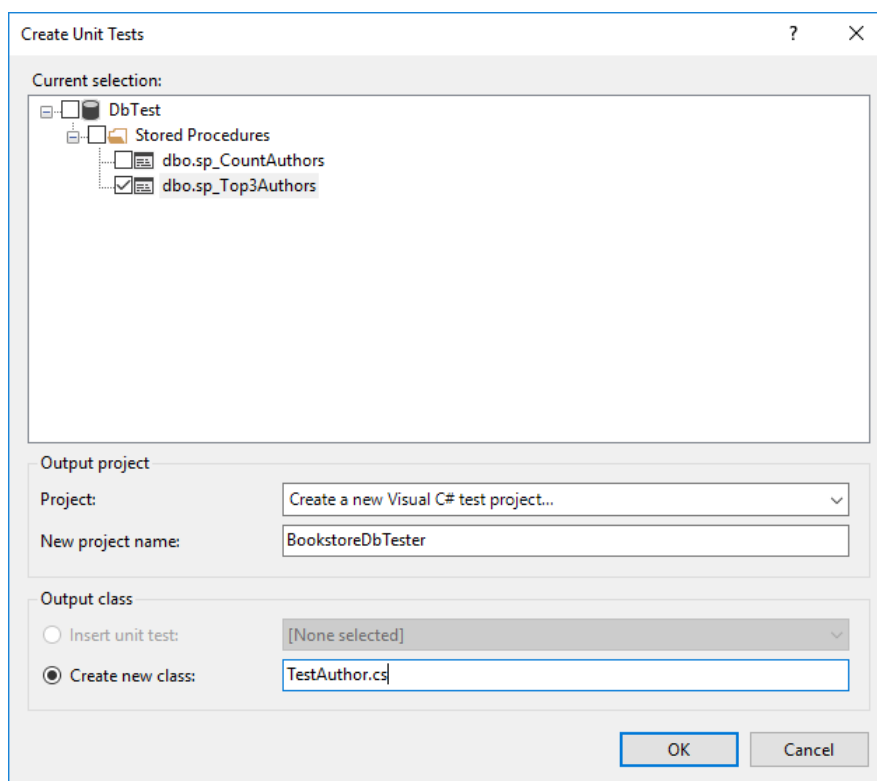
Mở rộng thư mục Projects trong cửa sổ và tìm các store procedure để tạo unit test.



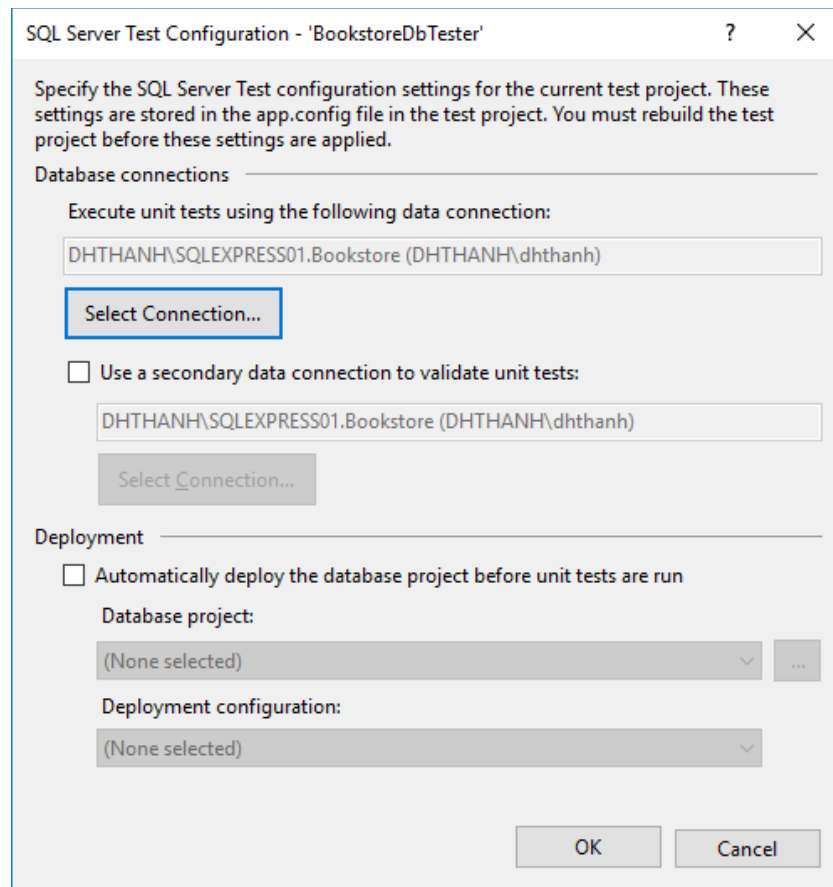
Click phải chuột vào store procedure muốn tạo unit test và chọn Create Unit Tests...



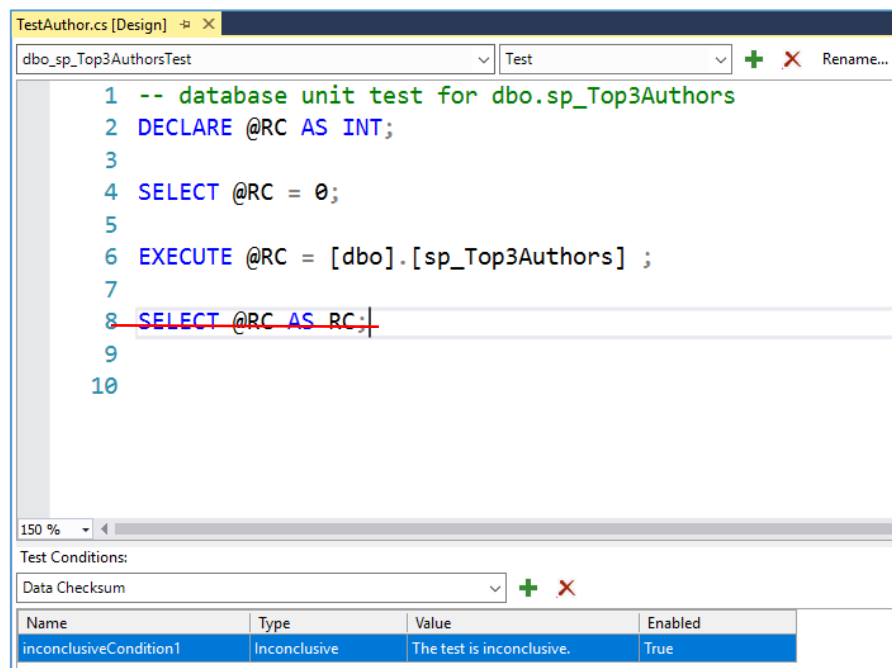
Tiếp theo chọn project chứa unit test hoặc tạo project unit test mới và đặt tên cho lớp chứa unit test.



Chọn database kết hợp để test

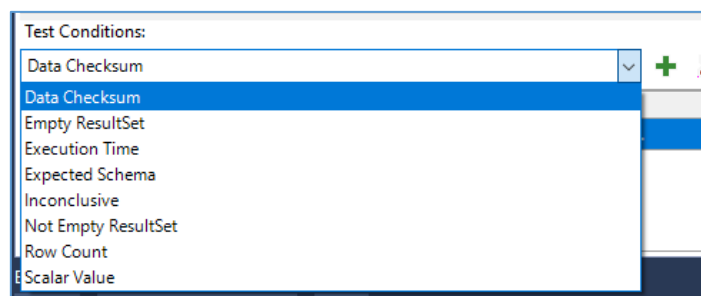


Một pane mặc định sẽ được tạo ra các assertion test cho store procedure. (xóa dòng code 8)

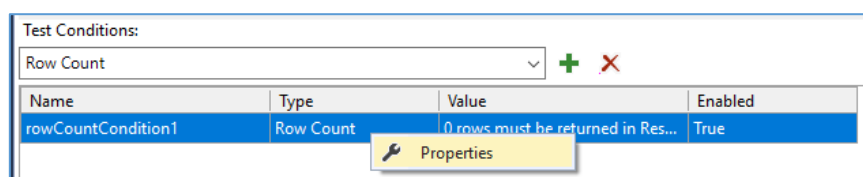


Có 8 loại Assertion test trong dropdown list

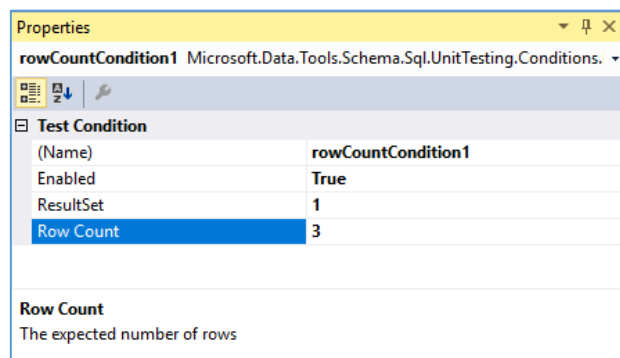
- Data Checksum: fail nếu giá trị checksum của kết quả trả về khác với giá trị checksum mong đợi.
- Empty Resultset: fail nếu result set trả về khác rỗng.
- Execution Time: fail nếu thời gian thực thi dài hơn thời gian mong muốn.
- Expected Schema: fail nếu các cột và các kiểu dữ liệu của result set không khớp với điều kiện test chỉ định.
- Not Empty Result ResultSet: Fail nếu kết quả result set là rỗng.
- Row Count: fail nếu số lượng dòng kết quả trả về không đúng số lượng mong muốn.
- Scalar Value: fail nếu một giá trị chỉ định trong result set khác với một giá trị mong muốn.
- Inconclusive.



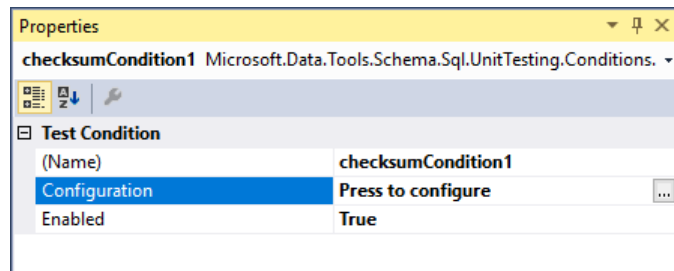
Tạo Row Count: trong dropdown list Test Conditions chọn Row Count và click vào nút **+** bên cạnh, sau đó click chuột phải vào test vừa tạo > chọn Properties.



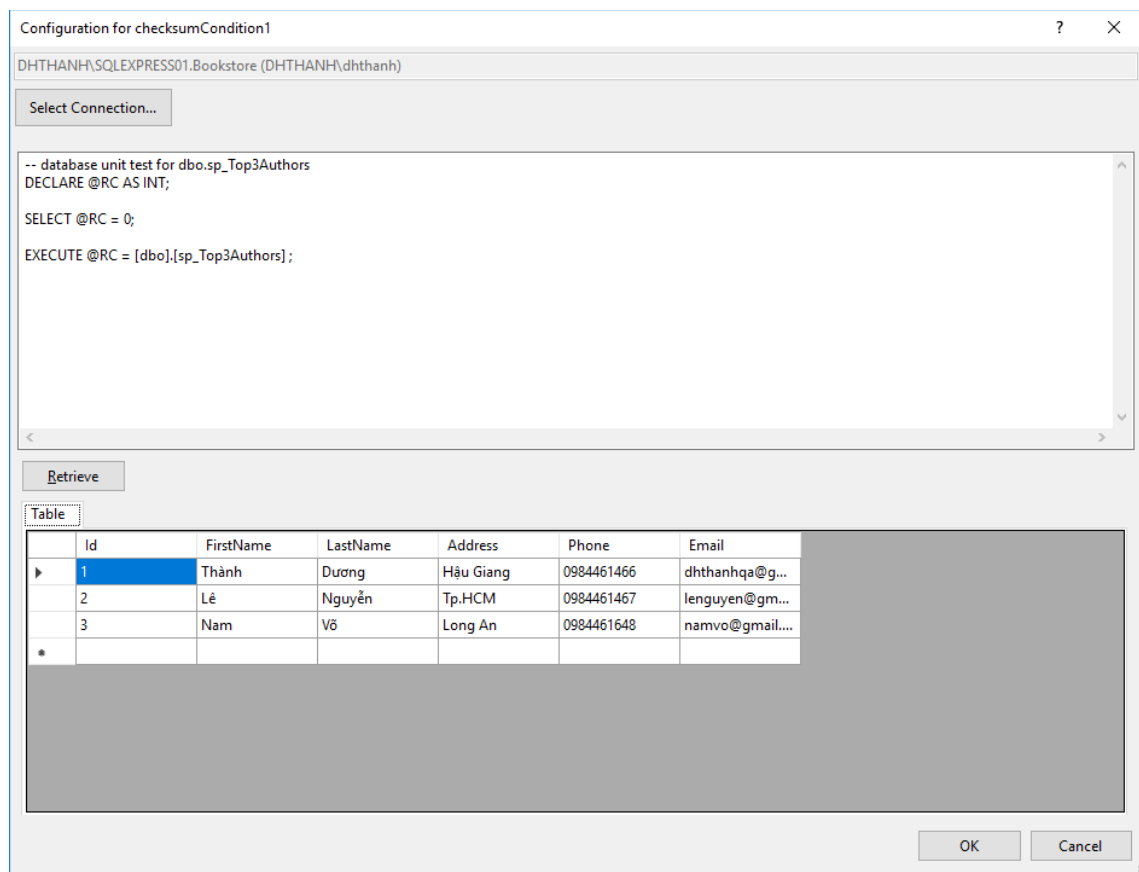
Và thiết lập thuộc tính Row Count là số dòng mong muốn trả về.



Tạo Data Checksum: trong dropdown list Test Conditions chọn Data Checksum và click nút **+** bên cạnh, click chuột phải vào test vừa tạo > chọn Properties > Click vào nút **...** của Configuration để cấu hình test.

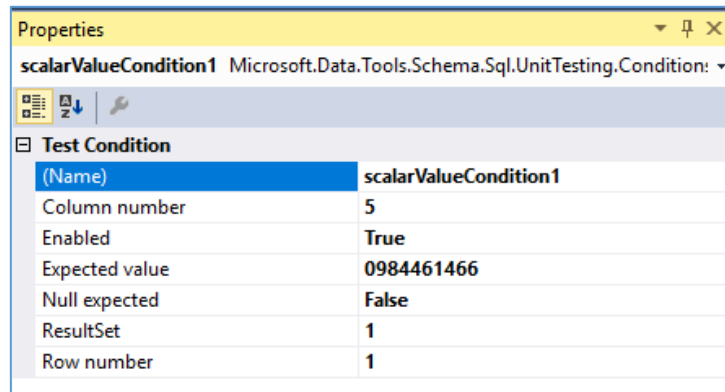


Chọn kết nối và ấn nút “Retrieve”, khi đó một checksum mong muốn (expected) sẽ được tạo ra để so sánh xác nhận với quả.



Name	Type	Value	Enabled
rowCountCondition1	Row Count	3 rows must be returned in ResultSet 1.	True
checksumCondition1	Data Checksum	ResultSet is expected to have checksum -771168943	True

Tạo Scalar Value: trong dropdown list Test Conditions chọn Scalar Value và click nút **+** bên cạnh, click chuột phải vào test vừa tạo > chọn Properties và thiết lập các thuộc tính như sau:



Điều này có nghĩa là kết quả mong muốn ở dòng 1, cột 5 có giá trị là “0984461466”.

Bài 1: Cho hàm tính x^n bằng đệ quy, trong đó x là số thực và n là số nguyên bất kỳ, biết x^n được tính như sau:

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

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

Viết unit test kiểm thử hàm Power với đặc tả yêu cầu như trên.

Bài 2: Một 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 ($0 \leq i \leq n$), với a_i là hệ số của x^i và giá trị biến nguyên x . Nếu người dùng nhập không đủ $n + 1$ hệ số cho đa thức hoặc nhập n âm thì ném ra ngoại lệ `ArgumentException`, với nội dung lỗi là “Invalid Data”.

Viết Unit Test kiểm tra đoạn chương trình hiện thực hóa yêu cầu trên.

```
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: Cho chương trình chuyển đổi số nguyên dương cơ số 10 sang cơ số nguyên k bất kỳ (với $2 \leq k \leq 16$). Viết các Unit Test kiểm thử đoạn chương trình này.


```
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 radix = 2)
    {
        int n = this.number;

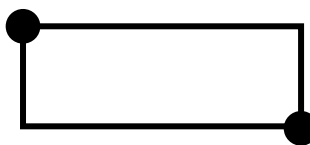
        if (radix < 2 || radix > 16)
            throw new ArgumentException("Invalid Radix");

        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:

Điểm trên bên trái



Điểm dưới bên phải

Lớp `HìnhChuNhat` 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 thử 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.

- a) *Viết chương trình cho phép nhập danh sách học viên và xác định danh sách học viên có thể nhận học bổng.*
- b) *Viết các Unit Test để kiểm thử các chức năng của chương trình trên.*