|  |
| --- |
| **Công cụ kiểm chứng phần mềm**  **Hướng dẫn thực hành: Unit Testing** |

## Tổng quan về Unit Testing

Unit testing là phương pháp kiểm thử được thực hiện bằng cách sử dụng các đoạn mã có cấu trúc giống như các đối tượng được xây dựng để kiểm tra các thành phần nhỏ trong hệ thống phần mềm như các lớp, hàm, thủ tục, phương thức, …

Sau khi đã soạn ra các đoạn mã, quá trình unit testing có thể được thực hiện tự động, liên tục hoặc định kì được gọi để thăm dò và phát hiện các lỗi kĩ thuật trong suốt quá trình phát triển.

Kĩ thuật unit testing được sử dụng để:

* Kiểm tra mọi đơn vị nhỏ nhất.
* Kiểm tra các trạng thái và ràng buộc của đối tượng ở các mức độ sâu hơn.
* Kiểm tra các qui trình (process) hay rộng hơn là các khung làm việc (workflow).
* Được ứng dụng nhiều trong qui trình phát triển phần mềm XP (Extreme Programming).

Những lợi ích mà kĩ thuật unit testing mang lại gồm:

* Thăm dò, phát hiện lỗi, duy trì sự ổn định.
* Phát hiện các thuật toán thực thi không hiệu quả, các thủ tục chạy vượt quá giới hạn thời gian.
* Phát hiện các lỗi nghiêm trọng có thể xảy ra trong những tình huống hẹp.
* Tạo hàng rào an toàn cho các khối mã.
* Giải phóng chuyên viên QA khỏi các công việc kiểm tra phức tạp.

Các trạng thải trong quá trình unit testing:

* **Pass**: khi một test case được thực thi và kết quả thực thi đúng với kết quả mong đợi.
* **Fail**: khi một test case được thực thi và kết quả thực thi không giống với kết quả mong đợi.
* **Ignore**.

Quá trình thực hiện unit testing:

* Thiết lập các điều kiện.
* Gọi phương thức cần kiểm tra.
* Kiểm tra sự hoạt động của phương thức.
* Dọn dẹp tài nguyên sau khi kết thúc kiểm tra.

Khi thực hiện quá trình unit testing:

* Người thực hiện phải có kiến thức về coding và thiết kế chương trình.
* Kĩ thuật sử dụng là **white-box testing**.
* Cần có đặc tả của module và source code của module đó.
* Phải chuẩn bị trước các test case và test script và sẽ được tái sử dụng.
* Cần được thực hiện càng sớm càng tốt trong quá trình phát triển phần mềm.
* Quá trình unit testing thường được tự động thực thi mỗi khi có sự thay đổi quan trọng.

Tóm lại, unit testing là quá trình viết các đoạn mã để kiểm thử các hàm được viết ra. Quá trình này thường do chính developer viết ra hàm đó thực hiện. Có thể thấy **unit testing** và **debugging** đều là quá trình tìm lỗi của hàm. Tuy nhiên, unit testing khác debugging ở chỗ người ta thường thực hiện unit testing một cách tự động.

## Hướng dẫn sử dụng NUnit

Bộ xUnit là các công cụ hỗ trợ thực hiện unit testing với các ngôn ngữ khác nhau:

* **JUnit** dùng để test các đoạn mã viết bằng ngôn ngữ Java.
* **CppUnit** dùng để test các đoạn mã viết bằng ngôn ngữ C++.
* **NUnit** dùng để test các đoạn mã viết bằng các ngôn ngữ trong .NET Framework.

Trong bài này, ta sẽ tìm hiểu cách sử dụng NUnit để test một đoạn code viết bằng ngôn ngữ C#.

Đầu tiên, ta download NUnit tại địa chỉ <http://www.nunit.org/index.php?p=download>. Có thể download bằng cài đặt .msi hoặc bản nén .zip. Ở đây, giả sử ta download bản nén .zip và tiến hành giải nén vào một thư mục nào đó trên máy.

Sau khi giải nén xong, có 3 file mà chúng ta cần quan tâm, các file này nằm trong thư mục bin:

* File **nunit.framework.dll**
* File **nunit.exe**
* File **nunit-console.exe**

Qui tắc thực hiện NUnit:

* Tạo project sử dụng NUnit riêng.
* Add reference đến file nunit.framework.dll
* Verify các hàm cần kiểm tra thông qua lệnh Assert.

Để thực thi quá trình unit testing với NUnit, có 2 cách:

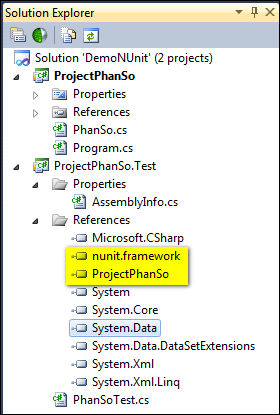
* **Sử dụng ứng dụng giao diện Windows (GUI Runner).** Các thao tác thực hiện sẽ khá đơn giản, tuy nhiên khó thực hiện tự động được.
* **Sử dụng giao diện dòng lệnh (Command Line).** Các thao tác sẽ phức tạp hơn vì phải nhớ và gõ các câu lệnh, tuy nhiên có thể thực hiện tự động.

Sau đây, ta sẽ làm thử một ví dụ về Unit Testing các **hàm dựng, cộng, trừ, chia và ToString** của lớp **phân số**.

Tạo project với ngôn ngữ C#, đặt tên là **ProjectPhanSo**. Ở đây, ta chọn loại project là Console Application. Tùy theo bài, có thể là Windows Application, Console Application, Class Library, đều được.

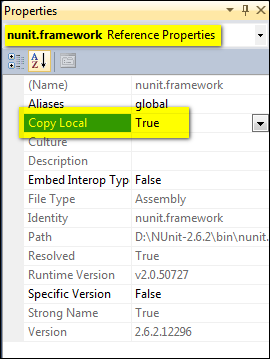
Add thêm một project thứ 2, đặt tên là **ProjectPhanSo.Test**. Project này là ta sẽ để là Class Library (project này sẽ build ra file .dll và sẽ dùng nunit.exe để chạy).

Hai project này sẽ nằm cùng một solution (**DemoNUnit**).



Trong ProjectPhanSo.Test, add reference đến 2 chỗ:

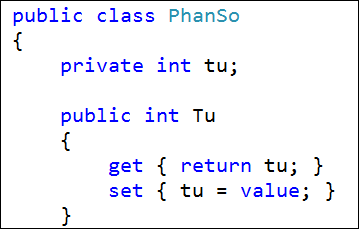
* File **nunit.framework.dll**: để ta có thể sử dụng các hàm của NUnit cung cấp. Nhớ chọn thuộc tính **CopyLocal = true** cho **nunit.framework.dll**.

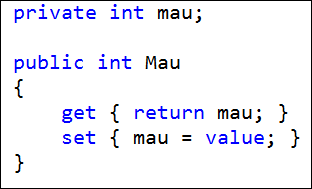


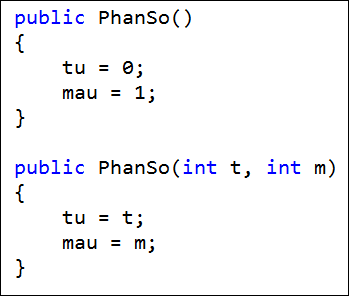
**ProjectPhanSo**: để ta có thể sử dụng lớp PhanSo và test lớp này.

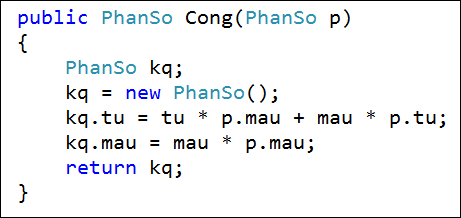
Trong **ProjectPhanSo**:

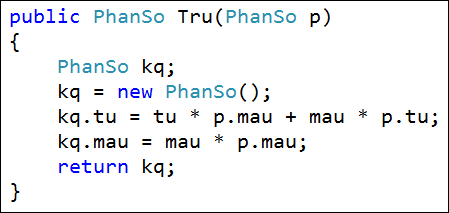
* Khai báo lớp PhanSo
* 2 thuộc tính: tử số và mẫu số.
* 2 property cho 2 thuộc tính.
* 2 hàm dựng mặc định và 2 tham số
* 3 hàm: cộng, trừ, chia, ToString.

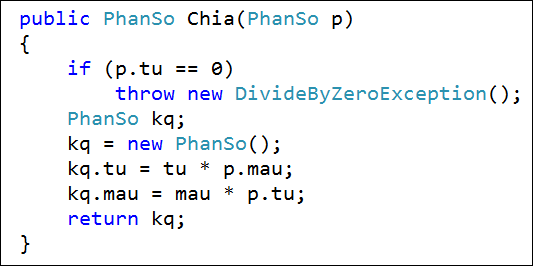


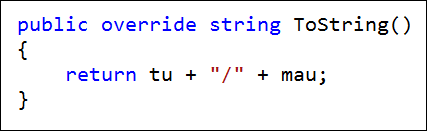








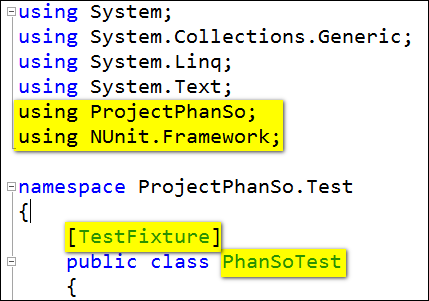




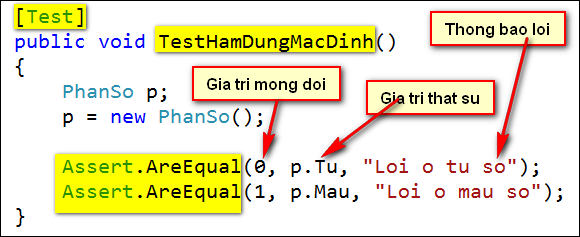
Trong **ProjectPhanSo.Test**:

* Tạo 1 class để test các hàm của lớp PhanSo. Chẳng hạn đặt tên class là PhanSoTest
* Using các namespace cần thiết.
* Thêm **[TestFixture]** vào trước khai báo lớp **PhanSoTest.**
* Viết các hàm để test các hàm của lớp PhanSo. Lưu ý:
  + Với mỗi hàm của lớp PhanSo, ta có thể viết nhiều hàm để test hàm đó.
* Thêm **[Test]** vào trước mỗi hàm.

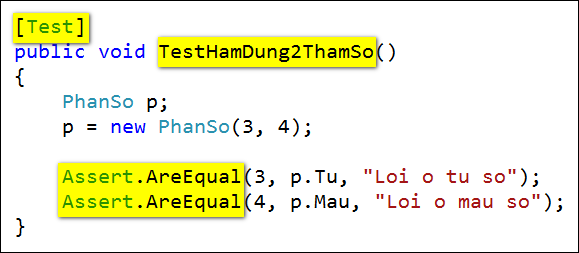
Trong mỗi hàm, ta sẽ gọi hàm tương ứng cần test, hàm này sẽ thực thi và trả về kết quả. Ta sẽ dùng các hàm của lớp **Assert** do NUnit cung cấp để kiểm tra xem giá trị trả về của hàm và giá trị mong đợi có giống nhau hay không?



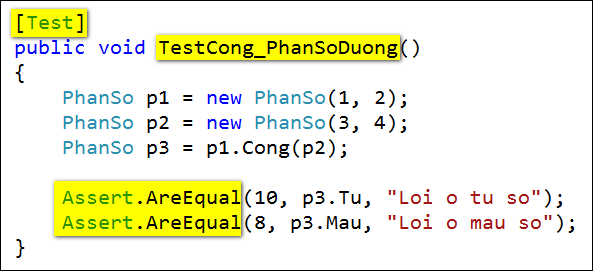
Hàm để test hàm dựng mặc định của lớp phân số:



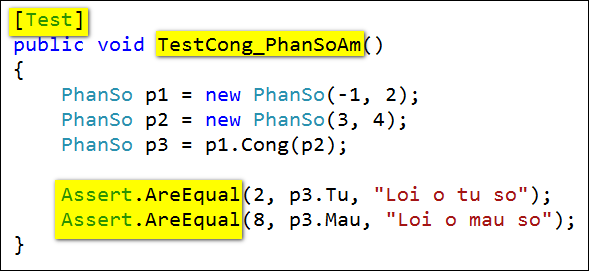
Hàm để test hàm dựng 2 tham số:



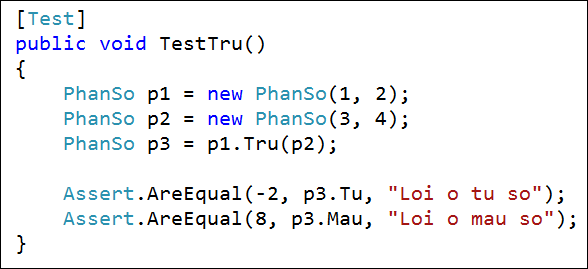
Hàm để test hàm cộng với phân số dương:



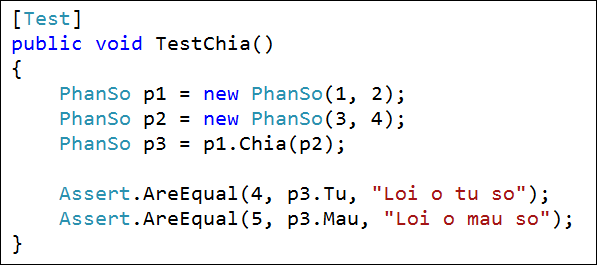
Hàm để test hàm cộng với phân số âm:



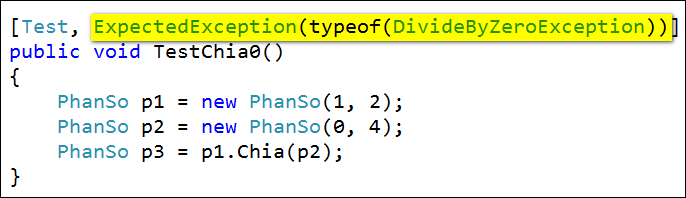
Hàm để test hàm trừ:



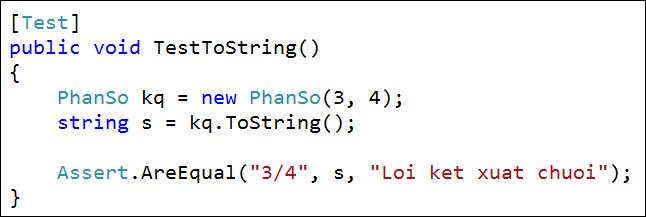
Hàm để test hàm chia:



Hàm để test hàm chia phân số bằng 0. **Lưu ý về cách để test xem là hàm chia của lớp phân số có throw ra DivideByZeroException**.



Hàm để test hàm ToString:

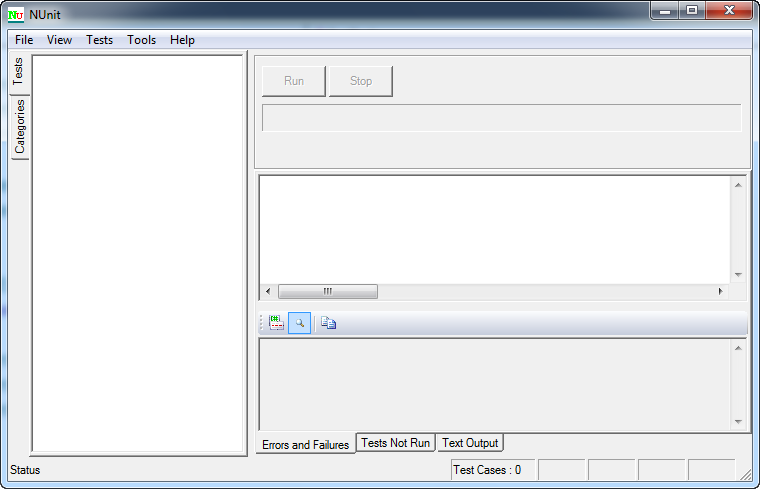


Sau khi hoàn thành, ta biên dịch toàn bộ solution. **ProjectPhanSo.Test** sẽ cho ra file **ProjectPhanSo.Test.dll**.

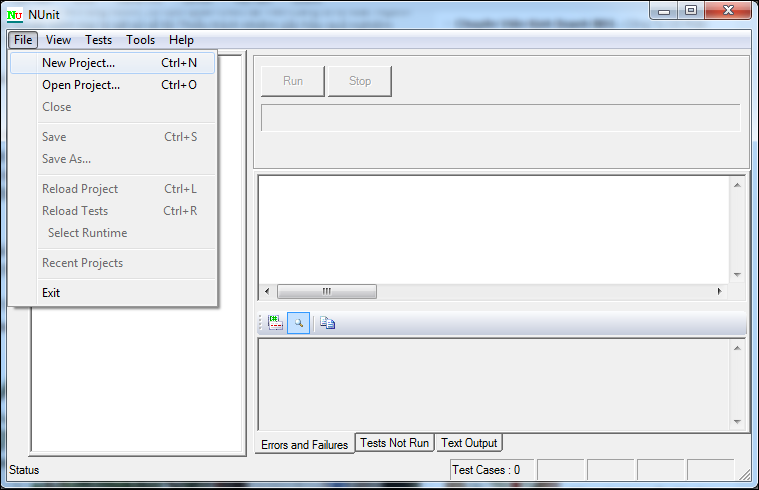
Như đã nói, để test bằng NUnit có 2 cách.

* Sử dụng nunit-console. Đơn giản là ta mở **Command Line** và chạy dòng lệnh **nunit-console.exe ProjectPhanSo.Test.dll**. Dĩ nhiên, ta phải chỉ đúng đường dẫn. Để có thể gọi nunit-console.exe ở mọi nơi thì phải **add đường dẫn thư mục bin của NUnit vào phần Path của Environment Variables**.
* Sử dụng màn hình giao diện của NUnit thì khá đơn giản.

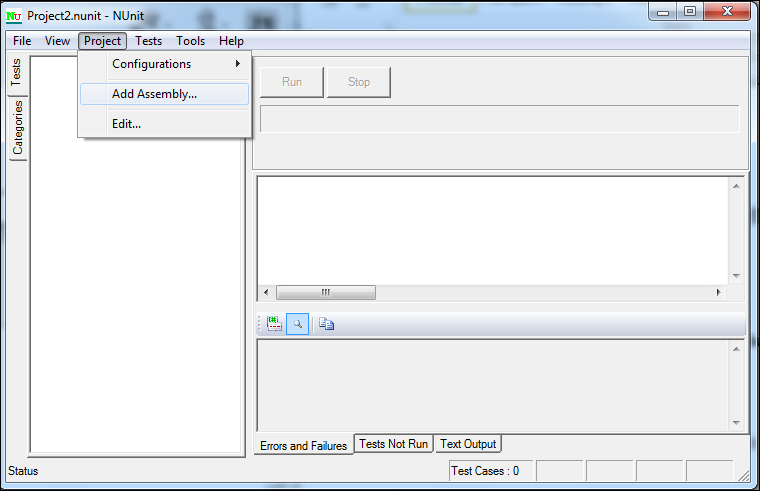
Chạy file nunit-x86.exe (hoặc nunit.exe).

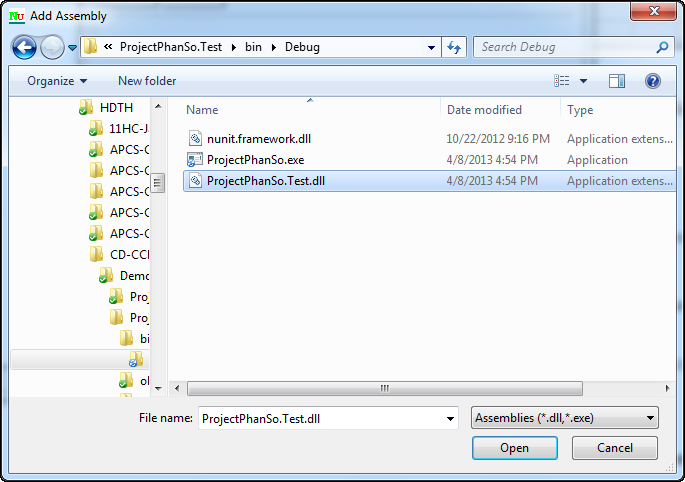


Tạo mới project

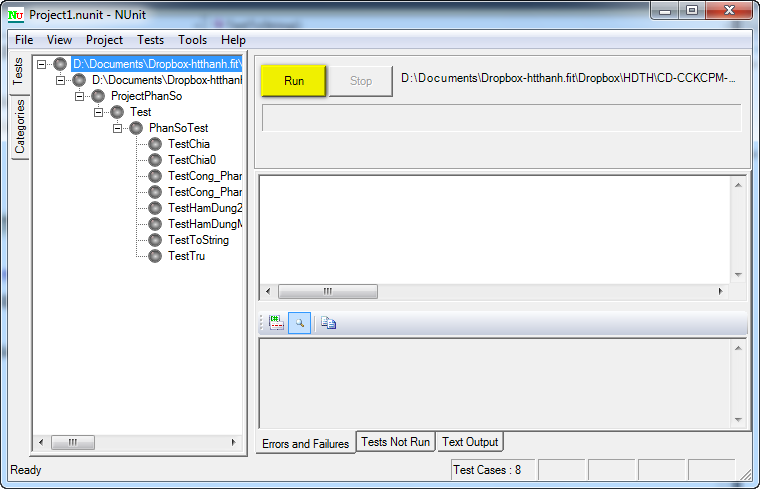


Chọn menu **Project – Add Assembly** và chỉ đường dẫn đến file **ProjectPhanSo.Test.dll**.



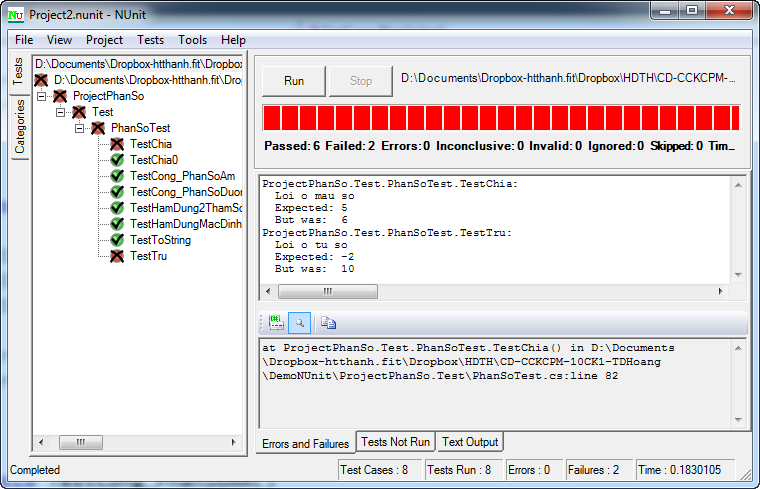


Click nút Run và sẽ ra kết quả.

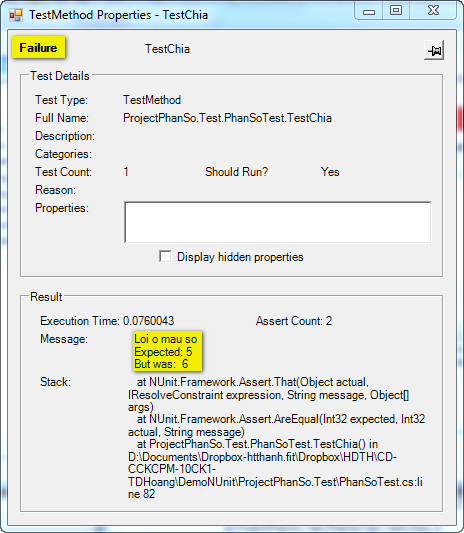


Ta thấy:

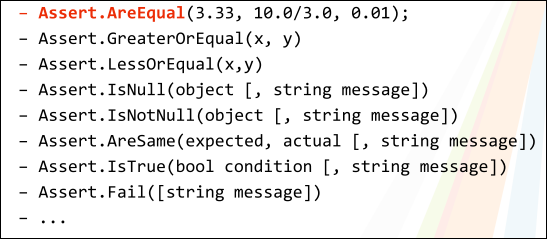
* Các hàm chia cho phân số 0, cộng phân số dương, cộng phân số âm, hàm dựng mặc định, hàm dựng 2 tham số, hàm ToString cho kết quả đúng khi chạy với test case.
* Các hàm trừ, chia cho kết quả sai khi chạy với test case.

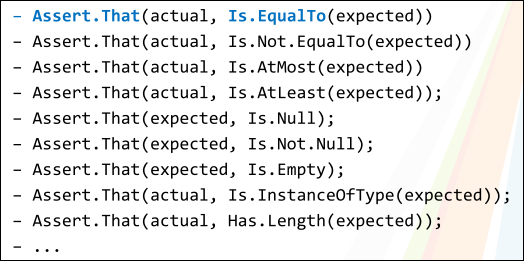


Ta có thể click phải vào một hàm và chọn **Properties** để xem chi tiết kết quả.



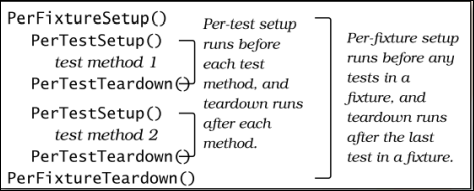
Ngoài hàm **Assert.AreEqual** ở trên, trong NUnit còn một số hàm khác có thể tham khảo như:

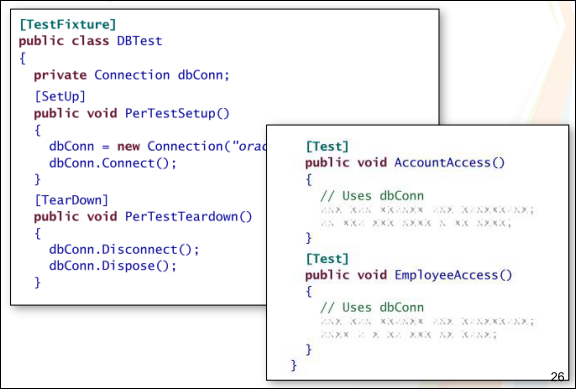


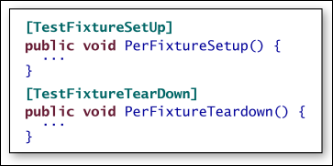


Ngoài 2 thuộc tính **[TestFixture]** và **[Test]**, ta còn có một số thuộc tính khác:

* **[TestFixtureSetUp]**: trong hàm được gán thụôc tính này, ta sẽ qui định các bước khởi tạo cần thiết trước khi test (tạo đối tượng, cấp vùng nhớ, mở kết nối đến CSDL, …).
* **[TestFixtureTearDown]**: trong hàm được gán thuộc tính này, ta sẽ qui định rằng sau khi test xong, cần thực hiện các bước dọn dẹp gì(hủy đối tượng, đóng kết nối CSDL, …).
* **[SetUp]**: hàm được gán thuộc tính này sẽ thực thi ngay trước khi thực thi một hàm test.
* **[TearDown]**: hàm được gán thuộc tính này sẽ thực thi ngay sau khi thực thi một hàm test.







## Kiểm thử nhiều test case trên cùng một hàm [Test] dùng nhãn [TestCase]

