KIỂM THỬ PHẦN MỀM

Bài tập Nunit (tt) – Mocking .NET object with NUnit

1. Mock là gì?

Mock là 1 kỹ thuật giúp cô lập các lớp khỏi sự phụ thuộc lẫn nhau nhằm phục vụ cho việc testing. Việc cô lập này nhằm giúp lập trình viên test được các hàm và lớp mình viết khi các lớp phụ thuộc chưa được cài đặt hoàn toàn.

Ví dụ :
A được giao code lớp Authenticator dùng để kiểm tra việc đăng nhập vào hệ thống.
B được giao code lớp Database dùng để kết nối đến cơ sở dữ liệu trong đó có hàm kiểm tra *tên và mật khẩu* đã tồn tại hay chưa?
Trong trường hợp B chưa code xong lớp Database nhưng A đã code xong lớp Authenticator. Nếu A muốn thực hiện unit test lớp Authenticator thì sao?

- Đợi cho đến khi B đã code xong lớp Database và sử dụng.
- Hoặc là A sẽ sử dụng kỹ thuật Mock để tạo mock object cho lớp Database và sau đó tiến hành test mà không cần đợi B.

2. Ví dụ

Giả sử dự án của ta có 3 lớp sau :

- ***Person*** : lớp chứa các thông tin về 1 người bao gồm id, firstName, lastName
- ***IPersonRepository*** : lớp giao diện cung cấp các hàm giúp truy xuất dữ liệu từ database
- ***PersonService*** : lớp cung cấp các business logic của chương trình. Nó sẽ làm việc với tầng truy xuất dữ liệu, tiến hành tính toán sau đó trả kết quả về cho tầng giao diện.

Trong bối cảnh dự án, chưa có lớp nào hiện thực giao diện IpersonRepository, nhưng ta vẫn muốn thực hiện unit test cho lớp PersonService.

(1) Tạo project mới PersonMock.

(2) Tạo lớp Person:

```
public class Person
{
    public string Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public Person(string id, string fn, string ln)
    {
        Id = id; FirstName = fn; LastName = ln;
    }
}
```

(3) Tạo lớp PersonService

```
public class PersonService
{
    private IPersonRepository personRepon;

    public PersonService(IPersonRepository repos)
    {
        personRepon = repos;
    }

    public List<Person> getAllPeople()
    {
        return personRepon.GetPeople();
    }

    public List<Person> getAllPeopleSorted()
    {
        List<Person> people = personRepon.GetPeople();
        people.Sort(delegate (Person lhp, Person rhp){
            return lhp.LastName.CompareTo(rhp.LastName);
        });
        return people;
    }

    public Person getPerson(string id)
    {
        try
        {
            return personRepon.getPersonById(id);
        }
        catch(ArgumentException)
        {
            return null;
        }
    }

}
```

(4) Tạo giao diện IPersonRepository

```
public interface IPersonRepository
{
    List<Person> GetPeople();
    Person getPersonById(string id);
}
```

(5) Tạo project dùng để test: PersonMock.Unitest

(6) Tạo lớp PersonServiceTest :

Thực hiện add references (Nunit & PersonMock)

C:\Program Files\NUnit 2.5.2\bin\net-2.0\framework\nunit.mocks.dll

```
namespace PersonMock.Unitest
{
    [TestFixture]
    public class PersonServiceTest
    {
        //The dynamic mock proxy that we will use to implement IPersonRepository
        private DynamicMock personRepositoryMock;


       //Set up some testing data
        private Person onePerson = new Person("1", "Wendy", "Whiner");
        private Person secondPerson = new Person("2", "Tony", "Adam");
        private List<Person> peopleList;

        [SetUp]
        public void TestInit()
        {
            peopleList = new List<Person>();
            peopleList.Add(onePerson);
            peopleList.Add(secondPerson);

            //constructor a Mock Object of the IPersonRepository Interface
            personRepositoryMock = new DynamicMock(typeof(IPersonRepository));
        }

        [Test]
        public void TestGetAllPeople()
        {
            //tell that mock object when the "getPeople" method is called to return a
predefined list of people
            personRepositoryMock.ExpectAndReturn("GetPeople", peopleList);
            //Construct a person service with the Mock IPersonRepository
            PersonService service = new PersonService((IPersonRepository)
personRepositoryMock.MockInstance);
            //call a methods and asserts test
            Assert.AreEqual(2, service.getAllPeople().Count);
        }

        [Test]
        public void TestGetAllPeopleSorted()
        {
            //tell that mock object when the "getPeople" method is called to return a
predefined list of people
            personRepositoryMock.ExpectAndReturn("GetPeople", peopleList);

            PersonService service = new
PersonService((IPersonRepository)personRepositoryMock.MockInstance);

            //This method really has "business logic" in it - the sorting of people
            List<Person> people = service.getAllPeopleSorted();
            Assert.IsNotNull(people);
```

```
            Assert.AreEqual(2, people.Count);

            //Make sure the first people returned is the correct one
            Person p = people[0];
            Assert.AreEqual("Adam", p.LastName);
        }

        [Test]
        public void TestGetSinglePersonWithValidId()
        {
            //tell that mock object when the "getPerson" method is called to return a
predefined Person
            personRepositoryMock.ExpectAndReturn("getPersonById", onePerson, "1");
            PersonService service = new
PersonService((IPersonRepository)personRepositoryMock.MockInstance);

            Person p = service.getPerson("1");
            Assert.IsNotNull(p);
            Assert.AreEqual("1", p.Id);
        }

        [Test]
        public void TestGetSinglePersonWithInvalidId()
        {
            //tell that mock object when the "getPerson" method is called to return a
predefined Person
            personRepositoryMock.ExpectAndReturn("getPersonById", new
ArgumentException("Invalid person id"), null);
            PersonService service = new
PersonService((IPersonRepository)personRepositoryMock.MockInstance);

            Assert.IsNull(service.getPerson(null));
        }

    }
}
```

1. Giải thích ví dụ

- Để sử dụng được mock trong Nunit ta phải add thư viện nunit.mocks.dll và using package Nunit.Mock;

```
using NUnit.Framework;
using NUnit.Mocks;
```

- NUnit cung cấp lớp DynamicMock dùng để tạo mock.

```
//The dynamic mock proxy that we will use to implement IPersonRepository
private DynamicMock personRepositoryMock;
```

- Đối tượng mock và các dữ liệu dùng để test được khởi tạo trong hàm TestInit :

```
[SetUp]
public void TestInit()
{
    peopleList = new List<Person>();
    peopleList.Add(onePerson);
```

```
            peopleList.Add(secondPerson);

            //constructor a Mock Object of the IPersonRepository Interface
            personRepositoryMock = new
DynamicMock(typeof(IPersonRepository));
        }
```

Lưu ý, tham số để khởi tạo 1 đối tượng DynamicMock là kiểu của lớp hoặc giao diện muốn tạo đối tượng giả lập. Ở đây ta muốn tạo 1 đối tượng giả lập có giao diện IPersonRepository

- Để sử dụng mock, ta có 2 bước sau :
    - Bước 1: Bảo mock biết đối tượng giả lập sẽ có những hàm gì và kết quả trả ra của hàm đó là gì ?

```
//tell that mock object when the "getPeople" method is called to return a
predefined list of people
personRepositoryMock.ExpectAndReturn("GetPeople", peopleList);
```

Ở đây ta nói mock rằng đối tượng giả lập sẽ có hàm tên GetPeople và kết quả trả về của hàm này là peopleList.

    - Bước 2: Sử dụng mock :

```
//Construct a person service with the Mock IPersonRepository
PersonService service = new PersonService((IPersonRepository)
personRepositoryMock.MockInstance);
```

Sau khi đã thiết lập mock xong ta có thể lấy đối tượng giả lập bằng câu lệnh : (IPersonRepository) personRepositoryMock.MockInstance

Câu lệnh trên sẽ cho ta 1 đối tượng giả lập đối tượng IPersonRepository. Khi đối tượng service gọi hàm GetPeople từ đối tượng này, đều sẽ cho kết quả là peopleList.

Tham khảo

[1]. **http://www.zorched.net/2007/03/10/mocking-net-objects-with-nunit/**

[2]. **http://www.typemock.com/Docs/Unit_Test_Patterns_for_NET_Development_Part-1.php**

[3]. **http://www.typemock.com/Docs/Unit_Test_Patterns-2_Mock_Types.php**

[4]. **http://martinfowler.com/articles/mocksArentStubs.html**

Bài tập

Lớp **IntegerSet** là một lớp đại diện cho 1 tập hợp các số nguyên, có các phương thức sau **<Sử dụng mã nguồn kèm theo>:**

Phương thức max,min, avg, search : dùng để tìm max, min, trung bình, và tìm kiếm 1 phần tử trong mảng. Hàm search trả về 1 nếu tìm thấy và 0 nếu không tìm thấy.

Các phương thức này sử dụng lớp Algorithm để cài đặt.

Agorithm là 1 lớp interface có các phương thức sau :

- int max(IntegerSet s)
- int min(IntegerSet s)
- double avg(IntegerSet s)
- int search (IntegerSet s)

Sinh viên không cài đặt lớp Algorithm

Yêu cầu :

Thực hiện unit testing cho các hàm max, min, avg, và search của lớp IntegerSet sử dụng Mocks như hướng dẫn phần đầu.