Exemples en C# pour 'Les simulacres ne sont pas des bouchons'

par Bruno Orsier (Site Web de Bruno Orsier)

Date de publication : 9/9/2007

Dernière mise à jour : 24/10/2007

Cette page montre comment réaliser en C# les trois premiers exemples de l'article de Martin Fowler **"Les simulacres ne sont pas des bouchons"**.
Le code source du projet peut être téléchargé **ici**

Les tests traditionnels

Réalisation détaillée

Ici nous appliquons directement ce qui a déjà été vu dans le tutoriel sur les pentaminos, et mettons donc en place un projet Visual C# Express avec Nunit, comme **indiqué précédemment**.

Puis nous nous laissons guider par les tests, ce qui est facile puisqu'ils sont déjà écrits en Java. Nous obtenons ainsi la classe de test suivante :

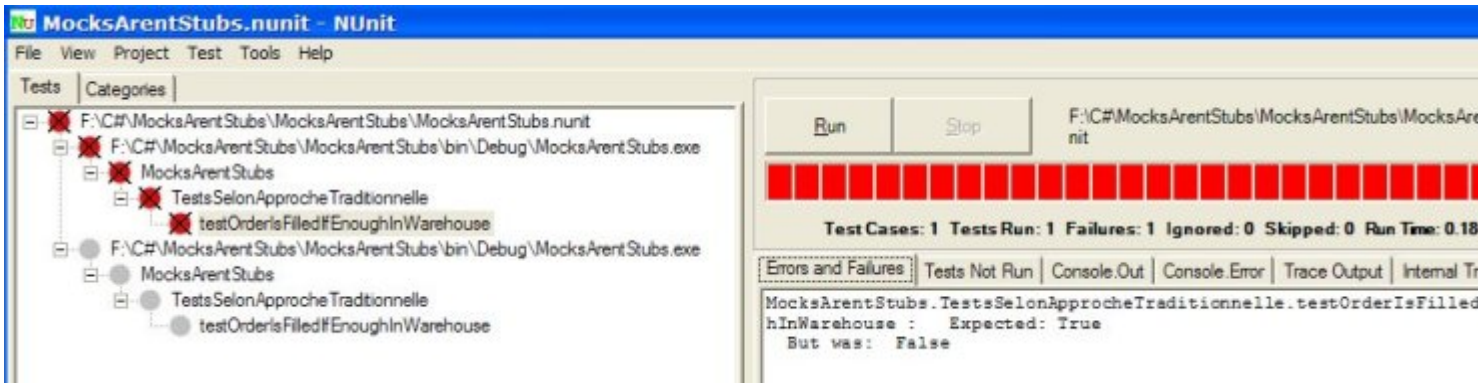
```
[TestFixture]
public class TestsSelonApprocheTraditionnelle
{
    private static String TALISKER = "Talisker";
    private static String HIGHLAND_PARK = "Highland Park";
    private Warehouse warehouse;

    [SetUp]
    public void setUp()
    {
        warehouse = new Warehouse() ;
        warehouse.add(TALISKER, 50);
        warehouse.add(HIGHLAND_PARK, 25);
    }

    [Test]
    public void testOrderIsFilledIfEnoughInWarehouse()
    {
        Order order = new Order(TALISKER, 50);
        order.fill(warehouse);

        Assert.IsTrue(order.isFilled());
        Assert.AreEqual(0, warehouse.getInventory(TALISKER));
    }
}
```

Pour parvenir à compiler, ces tests nous conduisent au code suivant :

```
public class Warehouse
{
    public void add(string itemIdentification, int orderAmount)
    {
    }
    public int getInventory(string itemIdentification)
    {
        return -1;
    }
}

public class Order
{
    public Order(string itemIdentification, int orderAmount)
    {
    }
    public void fill(Warehouse warehouse)
    {
    }
    public Boolean isFilled()
    {
        return false;
```

```
        }
    }
```

Et de plus nous obtenons la barre rouge :



si bien qu'il ne reste plus qu'à écrire le code suivant (implémentation évidente) pour passer au vert :

```
    public class Warehouse
    {
        private Hashtable AvailableItems = new Hashtable();

        public void add(string itemIdentification, int orderAmount)
        {
            AvailableItems.Add(itemIdentification, orderAmount);
        }

        public void remove(string itemIdentification, int orderAmount)
        {
            AvailableItems[itemIdentification] = (int) AvailableItems[itemIdentification] -
orderAmount;
        }


        public int getInventory(string itemIdentification)
        {
            return (int) AvailableItems[itemIdentification]; ;
        }
    }

    public class Order
    {
        private string Identification;
        private int Amount;
        private Boolean Filled;

        public Order(string itemIdentification, int orderAmount)
        {
            Identification = itemIdentification;
            Amount = orderAmount;
        }

        public void fill(Warehouse warehouse)
        {
            warehouse.remove(Identification, Amount);
        }
        public Boolean isFilled()
        {
            return true;
```

```
        }
    }
```

C'est une implémentation minimum pour passer au vert : par exemple il n'y a pas de gestion d'exception, et c'est donc une implémentation très optimiste. Mais dans une optique TDD, aucun test ne nous conduit à une telle gestion, donc nous n'écrivons pas de tel code. Par contre nous n'avons pas encore programmé toute la logique de gestion de l'entrepôt, et pour cela il nous faut le deuxième test :

```
        [Test]
        public void testOrderDoesNotRemoveIfNotEnough()
        {
            Order order = new Order(TALISKER, 51);
            order.fill(warehouse);
            Assert.IsFalse(order.isFilled());
            Assert.AreEqual(50, warehouse.getInventory(TALISKER));
        }
```

Ce code compile, et nous obtenons la barre rouge comme prévu. Pour avoir la barre verte il faut principalement modifier la méthode *fill* :

```
        public void fill(Warehouse warehouse)
        {
            if (warehouse.getInventory(Identification) >= Amount)
            {
                warehouse.remove(Identification, Amount);
                Filled = true;
            }
        }
```

Code final

*

Voici le code complet ainsi obtenu. Tout d'abord le code de test :

```
    [TestFixture]
    public class TestsSelonApprocheTraditionnelle
    {
        private static String TALISKER = "Talisker";
        private static String HIGHLAND_PARK = "Highland Park";
        private Warehouse warehouse;

        [SetUp]
        public void setUp()
        {
            warehouse = new Warehouse() ;
            warehouse.add(TALISKER, 50);
            warehouse.add(HIGHLAND_PARK, 25);
        }

        [Test]
        public void testOrderIsFilledIfEnoughInWarehouse()
        {
            Order order = new Order(TALISKER, 50);
            order.fill(warehouse);

            Assert.IsTrue(order.isFilled());
            Assert.AreEqual(0, warehouse.getInventory(TALISKER));
        }
```

```
        [Test]
        public void testOrderDoesNotRemoveIfNotEnough()
        {
            Order order = new Order(TALISKER, 51);
            order.fill(warehouse);
            Assert.IsFalse(order.isFilled());
            Assert.AreEqual(50, warehouse.getInventory(TALISKER));
        }

    }
```

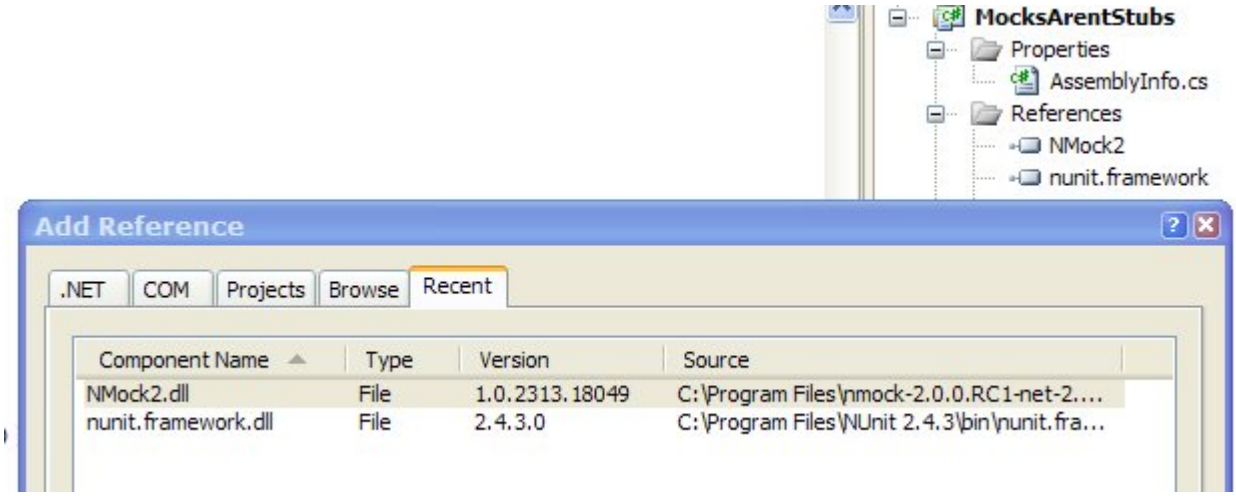
Puis le code nécessaire à l'exécution des tests :

```
    public class Warehouse
    {
        private Hashtable AvailableItems = new Hashtable();

        public void add(string itemIdentification, int orderAmount)
        {
            AvailableItems.Add(itemIdentification, orderAmount);
        }

        public void remove(string itemIdentification, int orderAmount)
        {
           AvailableItems[itemIdentification] = (int) AvailableItems[itemIdentification] -
orderAmount;
        }


        public int getInventory(string itemIdentification)
        {
            return (int) AvailableItems[itemIdentification]; ;
        }
    }

    public class Order
    {
        private string Identification;
        private int Amount;
        private Boolean Filled;

        public Order(string itemIdentification, int orderAmount)
        {
            Identification = itemIdentification;
            Amount = orderAmount;
            Filled = false;
        }

        public void fill(Warehouse warehouse)
        {
            if (warehouse.getInventory(Identification) >= Amount)
            {
                warehouse.remove(Identification, Amount);
                Filled = true;
            }
        }
        public Boolean isFilled()
        {
            return Filled;
        }
    }
```

Exemple avec objet simulacre

Mise en place

Pour réaliser cet exemple, nous avons besoin d'un outil permettant de définir automatiquement nos objets simulacres (il est bien sûr possible de les définir manuellement, sans outil, mais cela est assez fastidieux). Nous allons utiliser ici **NMock** version 2, car elle est inspirée de jMock que Martin Fowler utilise dans son exemple. Après l'avoir téléchargée (prendre la version pour .net 2.0) et installée (pour la version Release Candidate 1 utilisée ici, il n'y a pas d'installateur et il faut copier le dossier *bin* à un endroit approprié, par exemple *C:\program files\nmock-2.0.0.RC1-net-2.0*), il faut ajouter la référence *NMock2* au projet Visual C# Express :



et penser à importer l'espace de noms correspondant dans les fichiers sources :

```
using NMock2;
```

L'utilisation de NMock2 va ensuite nous obliger à définir une interface, ce qui est assez normal car il nous faut un mécanisme pour pouvoir utiliser à la fois notre véritable objet (en production) et notre simulacre (en test). Donc voici notre interface :

```
public interface IWarehouse
{
    void add(string itemIdentification, int orderAmount);
    void remove(string itemIdentification, int orderAmount);
    int getInventory(string itemIdentification);
}

public class Warehouse : IWarehouse
{
 ...
}
```

Le test Java se transcrit alors assez directement en C#, avec toutefois deux particularités. D'une part pour exprimer l'ordre dans lequel nous attendons les méthodes il faut utiliser *using(mocks.Ordered) { ... }* ; d'autre part la création de l'objet simulacre est un peu différente car il faut utiliser cette fabrique de simulacres appellée *Mockery* :

```
        [Test]
        public void testFillingRemovesInventoryIfInStock()
        {
            //setup - data
            Order order = new Order(TALISKER, 50);

            Mockery mocks = new Mockery();
            IWarehouse warehouseMock = mocks.NewMock<IWarehouse>();

            //setup - expectations
            using (mocks.Ordered)
            {
                Expect.Once.On(warehouseMock)
                    .Method("hasInventory")
                    .With(TALISKER,50)
                    .Will(Return.Value(true));
                Expect.Once.On(warehouseMock)
                    .Method("remove")
                    .With(TALISKER, 50);
            }
            //exercise
            order.fill(warehouseMock);

            //verify
            mocks.VerifyAllExpectationsHaveBeenMet();
        }
    }
```

Le test échoue toutefois, car nous n'avons pas défini de telle méthode *hasInventory* (aucun test ne nous avait conduit à réaliser une telle méthode jusqu'à présent) :

```
MocksArentStubs.OrderInteractionTester.testFillingRemovesInventoryIfInStock :
System.ArgumentException :
mock object warehouse does not have a method matching hasInventory
```

Définir cette méthode ne suffit toujours pas, nous échouons cette fois pour une autre raison, plus intéressante car justement le test montre que nous n'avons pas encore utilisé cette méthode :

```
MocksArentStubs.OrderInteractionTester.testFillingRemovesInventoryIfInStock :
NMock2.Internal.ExpectationException :
unexpected invocation of warehouse.remove("Talisker", <50>)
Expected:
  Ordered:
    1 time: warehouse.hasInventory(equal to "Talisker", equal to <50>), will return <True> [called
 0 times]
    1 time: warehouse.remove(equal to "Talisker", equal to <50>) [called 0 times]
```

Il reste en effet à modifier la méthode *fill* pour qu'elle utilise *hasInventory* :

```
        public void fill(IWarehouse warehouse)
        {
            if (warehouse.hasInventory(Identification,Amount))
            {
                warehouse.remove(Identification, Amount);
                Filled = true;
            }
        }
```

Cela permet d'obtenir la barre verte. Comme NMock2 possède une méthode *WithAnyArguments*, le deuxième test est facile à écrire, et est présenté ci-dessous.

Code final

*

Voici le code complet correspondant à l'exemple de Martin Fowler :

```
[TestFixture]
public class OrderInteractionTester
{
    private static String TALISKER = "Talisker";

    [Test]
    public void testFillingRemovesInventoryIfInStock()
    {
        //setup - data
        Order order = new Order(TALISKER, 50);

        Mockery mocks = new Mockery();
        IWarehouse warehouseMock = mocks.NewMock<IWarehouse>();

        //setup - expectations
        using (mocks.Ordered)
        {
            Expect.Once.On(warehouseMock)
                .Method("hasInventory")
                .With(TALISKER,50)
                .Will(Return.Value(true));
            Expect.Once.On(warehouseMock)
                .Method("remove")
                .With(TALISKER, 50);
        }
        //exercise
        order.fill(warehouseMock);

        //verify
        mocks.VerifyAllExpectationsHaveBeenMet();
    }

    [Test]
    public void testFillingDoesNotRemoveIfNotEnoughInStock()
    {
        Order order = new Order(TALISKER, 50);

        Mockery mocks = new Mockery();
        IWarehouse warehouseMock = mocks.NewMock<IWarehouse>();

        Expect.Once.On(warehouseMock)
            .Method("hasInventory")
            .WithAnyArguments()
            .Will(Return.Value(false));

        order.fill(warehouseMock);

        mocks.VerifyAllExpectationsHaveBeenMet();
    }
```

Exemple avec TypeMock

Ici j'ai choisi de ne pas utiliser **easymock.net** car son développement semble être arrêté. A la place j'utilise **TypeMock.NET**, qui offre une version gratuite ainsi qu'une version commerciale. TypeMock va nous permettre d'illustrer la métaphore **enregistrer/rejouer** que Martin Fowler avait illustrée avec EasyMock.

Mise en place

Après avoir téléchargé et installé TypeMock, il suffit d'ajouter la référence au projet (sélectionner TypeMock.NET for .NET 2.0 dans la liste des références), et d'inclure l'espace de noms correspondant dans le fichier source :

```
using TypeMock;
```

Le code de test est alors particulièrement simple et direct à écrire, puisqu'on indique directement les appels souhaités pour définir les attentes :

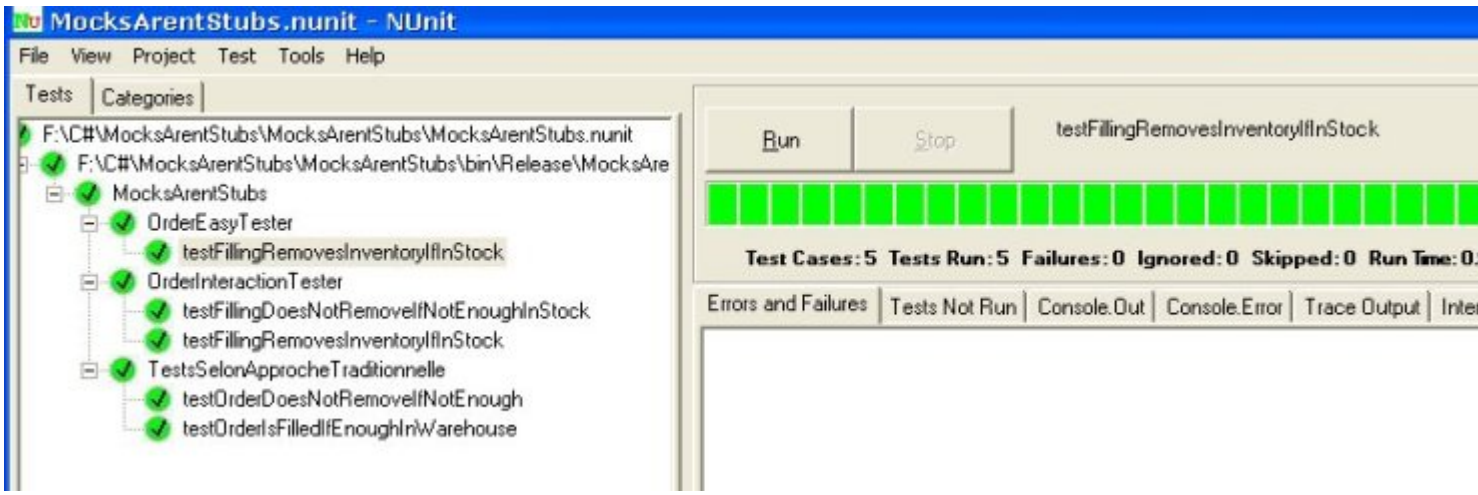
```
        [Test]
        public void testFillingRemovesInventoryIfInStock()
        {
            //setup - data
            Order order = new Order(TALISKER, 50);
            Warehouse warehouseMock = new Warehouse();

            //setup - expectations
            using (RecordExpectations recorder = RecorderManager.StartRecording())
            {
                warehouseMock.hasInventory(TALISKER, 50);
                recorder.Return(true);
                warehouseMock.remove(TALISKER, 50);
            }

            //exercise
            order.fill(warehouseMock);

            //verify
            MockManager.Verify();
        }
```

Le code est même beaucoup plus simple qu'avec EasyMock car il n'est pas nécessaire de manipuler des objets "contrôle".

Attention, on ne peut pas lancer NUnit directement, il faut le lancer par l'intermédiaire d'un programme fourni par TypeMock, comme suit (à taper en ligne de commande) :

```
"C:\Program Files\TypeMock\TypeMock.NET\TMockRunner.exe" "C:\Program Files\NUnit
 2.4.3\bin\nunit.exe"
```

On constate alors que nous obtenons la barre verte pour tous les tests que nous avons écrits jusque là :



Code final

*

Voici le code de test final :

```
[TestFixture]
public class OrderEasyTester
{
    private static String TALISKER = "Talisker";

    [Test]
    public void testFillingRemovesInventoryIfInStock()
    {
        //setup - data
        Order order = new Order(TALISKER, 50);
        Warehouse warehouseMock = new Warehouse();

        //setup - expectations
        using (RecordExpectations recorder = RecorderManager.StartRecording())
        {
            warehouseMock.hasInventory(TALISKER, 50);
            recorder.Return(true);
            warehouseMock.remove(TALISKER, 50);
        }

        //exercise
        order.fill(warehouseMock);

        //verify
        MockManager.Verify();
    }
}
```

Remerciements

Merci à **Alp** pour sa relecture de cet article.