MOCK AVEC LE FRAMEWORK MOQ

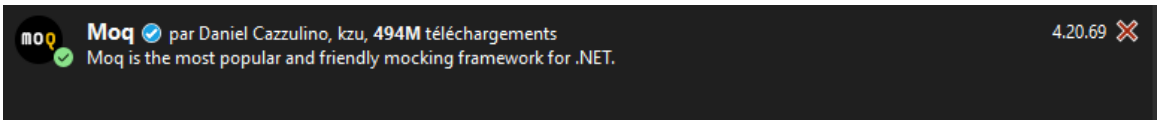
BOUNATIROU Rodolphe

CRM MULHOUSE

Mise en place

Installation

Pour utiliser le framework Moq, vous pouvez générer un projet de test unitaire (MSTest).
Dans ce dernier, il vous faudra ajouter par le biais du gestionnaire de packages nugGet, le framework Moq.



Pré requis

Le mock étant un substitut à un objet, pour la redéfinition de méthodes, au même titre que l'héritage ou la réalisation, il y'a des condition pré-requises.

Les méthodes substituées par le mock doivent être **virtual** ou **abstract**.
Ces méthodes doivent également être publiques afin de permettre leur accès par le mock.

Instanciation et modification

Vous trouverez les classes métier sur lequel s'appuiera le Mock en annexe.

Comme la méthode DireCoucou() se base sur de l'aléatoire, comment faire en sorte de tester cette méthode, alors que son résultat dépend de cet aléatoire ?

La valeur aléatoire provient de la fonction LancerAleatoire() de notre TestClass. Via un Mock, en redéfinissant le retour de cette méthode, on pourra tester DireCoucou() avec des données fixes et non aléatoire.

Instanciation du mock

Pour instancier un mock, il y'a deux méthodes :

1. Mock.of<T>()
2. new Mock<T>() ;

Avec Mock.of<T>(), l'objet de type T sera instancié au moment de la création de la ligne,
en revanche new Mock<T> instanciera l'objet T au moment du premier accès à sa propriété Object.

En fonction de ce choix la manière de redéfinir la méthodes va être différente

```
var mock = Mock.Of<TestClass>();
Mock.Get(mock).Setup(m => m.LancerAleatoire()).Returns(1);
```

```
var mock = new Mock<TestClass>();
mock.Setup(m => m.LancerAleatoire()).Returns(1);
```

Ces deux fonctions vont permettre, dans tout les cas, de redéfinir la fonction LancerAleatoire() de notre TestClass. (Ici avec un retour fixe à 1).

Ainsi on est certain que notre nombre obtenu sera désormais impair et que la chaine de caractère qu'on obtiendra via la fonction DireCoucou sera « Coucou »

Pour s'en assurer, il ne nous reste plus qu'à utiliser une méthode Assert.
Toujours décliné en deux versions selon le choix effectué à l'instanciation de notre mock.

```
Assert.AreEqual(mock.DireCoucou(), "Coucou");
```

Pour la version Mock.of<T>()

```
Assert.AreEqual(mock.Object.DireCoucou(), "Coucou");
```

Pour la version new Mock<T>()

Limitations observées

Les limitations constatées à l'utilisation des mock sont donc le fait qu'il ne redéfinit que les méthodes et attributs qui lui sont accessible, mais également qu'il doit redéfinir l'ensemble des méthodes virtual qui lui sont fournies, même si pour le test cela ne parait pas nécessaire, sous peine de lever une exception, ou rater des tests dut à des données nulles à la place des méthodes non redéfinies.

Annexe classe metier

```
public class Alea
{
    private static Random rnd;
    1 référence
    public Alea()
    {
        rnd = new Random();
    }

    1 référence
    public static Random GetInstance()
    {
        if (rnd != null)
            new Alea();
        return rnd;
```

```
public class TestClass
{
    private int prop;

    0 références
    public int Prop { get => prop; set => prop = value; }
    2 références | 1/1 ayant réussi
    public virtual int LancerAleatoire()
    {
        return Alea.GetInstance().Next(0,11);
    }

    1 référence | 1/1 ayant réussi
    public string DireCoucou()
    {
        int des = LancerAleatoire();
        return (des % 2 == 0 ? "Bonjour" : "Coucou");
    }
}
```