Aug 31

Moqing quick reference (Cheat Sheet)

This post is just a quick reference for myself, when I've spent a while away from TDD and want to remember some of the tricks.

Initial construction

```
var mockFoo = new Mock<IFoo>(MockBehavior.Strict);
```

Accessing the mock object as IFoo

```
IFoo foo = mockFoo.Object;
```

Initializing the inversion of control container (StructureMap)

Generally use Configure, not Initialize. Configure is additive, Initialize is not. Here's a complete test method showing the different ways we typically initialize.

```
[TestMethod()]
public void TestMethod()
{
    var mockOrange = new Mock<IOrange>(MockBehavior.Strict);
    MyDefaultHandlerForPear pearConcretion = new MyDefaultHandlerForPear();
    ObjectFactory.Configure( x=>
    {
       // use mock object
      x.For<IOrange>().Use(mockOrange.Object);
     // use new class when asked
      x.For<IApple>().Use<MyDefaultHandlerForApple>();
     // use an existing instance that implements IPear
      x.For<IPear>().Use(pearConcretion);
```

```
    });
}
```

Setting up void calls

```
public interface IOrange
{
  void Peel();
}
[TestMethod()]
public void TestOrange()
{
  var mockOrange = new Mock<IOrange>(MockBehavior.Strict);
  mockOrange.Setup(x => x.Peel());
}
```

Setting up simple return responses

```
public interface IOrange { int CountSlices(); }
...
var mockOrange = new Mock<IOrange>(MockBehavior.Strict);
mockOrange.Setup(x => x.CountSlices()).Returns(5);
```

Setting up open parameterized calls

When methods have parameters, you can allow calls to be made regardless of if the parameter's values are rubbish by using the convention It.IsAny<T>().

This becomes more meaningful when you use MockBehaviour.Strict, as you must call Setup for all functions that are going to be called, even if you're not really interested if they work.

```
public interface IOrange { int CountSlices(bool containingPips); }
...
 mockOrange = new Mock<IOrange>(MockBehavior.Strict);
 mockOrange.Setup(x => x.CountSlices(It.IsAny<bool>())).Returns(5);
```

Setting up closed calls

If you want to ensure only specific parameter values or objects are passed, your setup can explicitly allow these. From the above snippet..

```
mockOrange.Setup(x => x.CountSlices(false)).Returns(2);
mockOrange.Setup(x => x.CountSlices(true)).Returns(3);
```

Setting up a function callback

If you want to just call a function, use the anonymous lambda within the return statement.

```
mockOrange.Setup(x => x.CountSlices(false)).Returns(()=>{ CallMyFunction(); retu
```

Setting up intelligent behaviour using parameters

Extending the above syntax, if your goal is to stimulate your class under test with conditional behaviours, you can provide fake implementations inline. This is achieved by re-specifying the functions parameters in the lambda expression.

```
public interface IOrange { int CountSlices(bool containingPips, int juicePercentage); }
[TestMethod()]
public void TestOrange()
{
  var mockOrange = new Mock<IOrange>(MockBehavior.Strict);
  mockOrange.Setup(x => x.CountSlices( It.IsAny<bool>(), It.IsAny<int>()))
    .Returns( ( bool containingPips, int juicePercentage) =>
    {
      if( containingPips && juicePercentage>50 )
        return 2;
      else
        return 4;
    } );
}
```

Private method calling on classes

If you've got yourself into a position where you need to call a private method of a class under test, you can do so, without making them public.

My opinion – if you're doing this, you've not architected your classes correctly (single responsibility principle).

See this stack exchange post on how to achieve this.

Verifying frequency of calls

```
public interface IOrange { int CountSlices(bool containingPips); }
[TestMethod()]
public void TestOrange()
{
   var mockOrange = new Mock<IOrange>(MockBehavior.Strict);
```

```
    mockOrange.Setup(x => x.CountSlices( It.IsAny<bool>())).Returns( 4 );
    // do something that would cause CountSlices to be called
    // verify Count slices called once with any bool
    mockOrange.Verify(x => x.CountSlices(It.IsAny<bool>()), Times.Once());
    // verify Count slices called twice with true
    mockOrange.Verify(x => x.CountSlices(true), Times.Exactly(2));
    // verify Count slices never called with false
    mockOrange.Verify(x => x.CountSlices(false), Times.Never());
}
```


Related

Allowing IIS 7.5 Applications to communicate with SQL server via Windows Authentication
2013/05/11
In "ASP.NET"

Pragmatic SOLID – Part 5 – The Dependency Inversion Principle
2013/12/14
In "Object Orientated Design"

Pragmatic SOLID Part 2 - The Open/Closed Principle
2013/11/09
In "CSharp"

This entry was posted on 2013/08/31, in CSharp and tagged C++, callbacks, IoC, It.IsAny, Mock, Moq, Parameterized function calls, StructureMap, TDD. Bookmark the permalink.

1 Comment

← Windows Phone 8 and 7 – Locking the aspect ratio to portrait or landscape

Pragmatic SOLID Part 1 – The Single Responsibility Principle →

ONE THOUGHT ON "MOQING QUICK REFERENCE (CHEAT SHEET)"

REPLY

ytd2525 says:

2013/09/02 at 09:48

Reblogged this on ytd2525.