عنوان: UnitTest برای کلاس های Abstract با استفاده از Rhino Mocks
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۱/۱۱/۲۳ ۱۲:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Rhino Mocks, abstract

در این پست قصد دارم کلاس زیر رو براتون آزمایش کنم:

```
public abstract class myabstractclass
{
    public abstract string dosomething( string input );

    public double round( double number , int decimals )
    {
        return math.round( number , decimals );
    }
}
```

در کلاس بالا که abstract هستش، متدی دارم که abstract است و بدنه‌ای نداره و از متد بعدی به اسم round برای گرد کردن اعداد استفاده می‌شه. برای تست کلاس بالا و اطمینان از درست بودن متدها باید به روش زیر عمل نمود:

روش اول:

در این روش ابتدا باید کلاسی نوشت تا کلاس abstract بالا رو پیاده سازی کنه:

```
public class mynewclass : myabstractclass
   {
       public override string dosomething( string input )
       {
           return input;
       }
   }
```

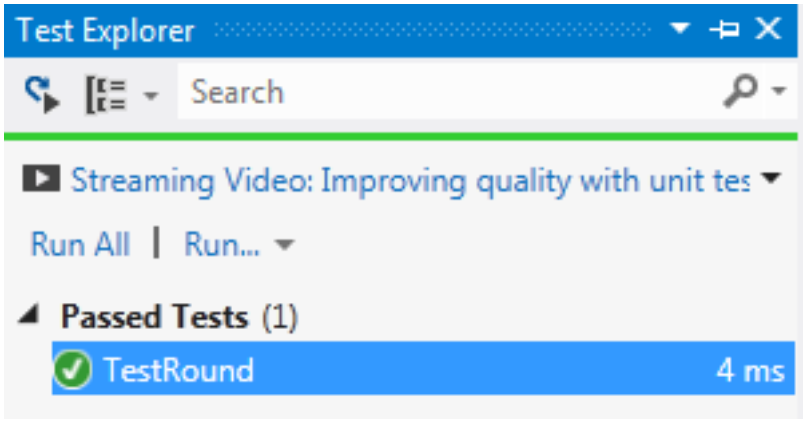
بعد می‌شه خیلی راحت برای کلاس دوم متدهای تست رو نوشت. به روش زیر:

```
[testclass]
    public class mytest
    {
        [testmethod]
        public void testround()
        {
            mynewclass mynewclass = new mynewclass();

            var result = mynewclass.round( 5.55 , 1 );

            assert.areequal( 5.6 , result );
        }
    }
```

که بعد از اجرا نتیجه زیر رو خواهید دید



البته روش بالا خیلی مورد پسند من نیست.

در روش دوم که من خیلی بیشتر بهش علاقه دارم دیگه نیازی به استفاده از یک کلاس دوم برای پیاده سازی کلاس abstract نیست. بلکه در این روش از ابزار rhinomocks برای این کار استفاده می‌کنیم . استفاده از rhino mocks به چندین روش امکان پذیره که امروز 2 روش اونو براتون توضیح میدم:
در روش اول از mockrepository استفاده می‌کنیم و در روش دوم از روش aaa یا arrange-act-assert

استفاده از mockrepository :
ابتدا کدهای مربوطه رو می‌نویسم:

```
[testmethod]
       public void testwithmockrepository()
       {
           var mockrepository = new rhino.mocks.mockrepository();
           var mock = mockrepository.partialmock<myabstractclass>();

           using ( mockrepository.record() )
           {
               expect.call( mock.dosomething( arg<string>.is.anything ) ).return( "hi..."
).repeat.once();
           }
           using ( mockrepository.playback() )
           {
               assert.areequal( "hi..." , mock.dosomething( "salam" ) );
           }
       }
```

همانطور که در کدهای نوشته شده بالا می‌بینید ابتدا یک mockrepository ساخته شده، سپس از نمونه اون کلاس برای ساخت partialmock کلاس myabstractclass استفاده کردم. در این روش متدهای expect حتما باید بین بلاک record نوشته شوند تا بتونیم از اون‌ها در بلاک playback استفاده کنیم. نتیجه اجرای این متد تست هم مثل متد تست قبلی درست است.
در مورد expect.call باید بگم که از این کلاس برای شبیه سازی رفتار یک متد استفاده میشه (مثلا در مواقعی که یک متد برای انجام عملیات باید به دیتا بیس وصل شده و یک query را اجرا کنه) برای اینکه در تست، از این عملیات صرف نظر بشه از mockها استفاده کرده و رفتار متد رو به روش بالا شبیه سازی می‌کنیم. البته کار کردن با rhino mocks به صورت بالا دیگه از مد افتاده و جدیدا از روش aaa استفاده میشه که اونو در پایین توضیح می‌دم:

```
[testmethod]
      public void testwithaaa()
      {
          var mock = mockrepository.generatepartialmock<myabstractclass>();

          mock.expect( x => x.dosomething( arg<string>.is.anything ) ).return( "hi..."
).repeat.once();//arange

          var result = mock.dosomething( "salam" );//act
```

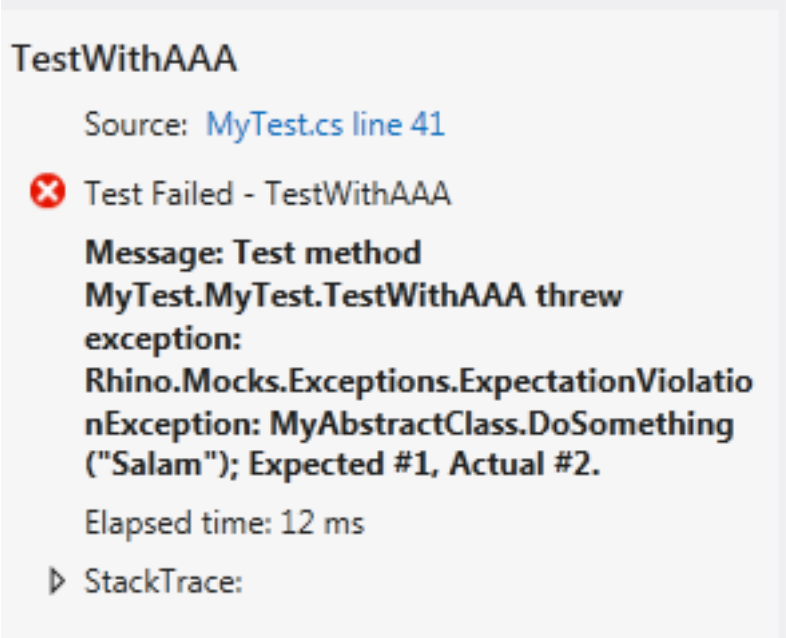
```
            assert.areequal( "hi..." , result );//assert
        }
```

توی این روش دیگه خبری از record و playback نیست و همانطور که مشخصه از سه مرحله arrange-act-assert تشکیل شده که هر مرحله رو براتون مشخص کردم. مزیت استفاده از این روش اینه که اولا تعداد خطوط کمتری برای کد نویسی نیاز داره و دوما سرعت اجراش از روش قبلی خیلی بیشتره. در مورد repeat.once هم بگم که این دستور نشون می‌ده فقط یک بار اجازه انجام عملیات act رو دارید. یعنی اگر کد هارو به صورت زیر تغییر بدیم با خطا روبرو می‌شیم:

```
[testmethod]
        public void testwithaaa()
        {
            var mock = mockrepository.generatepartialmock<myabstractclass>();

            mock.expect( x => x.dosomething( arg<string>.is.anything ) ).return( "hi..."
).repeat.once();//arange

            var result = mock.dosomething( "salam" );//act
            var result2 = mock.dosomething( "bye" );//act
            assert.areequal( "hi..." , result );//assert
        }
```



خطای آن هم واضح داره میگه که expected#1 هستش در حالی که actual#2 (تعداد دفعات حقیقی از دفعات مورد انتظار بیشتره)

توی پست‌های بعدی (اگه وقت بشه) حتما در مورد rhino mocks بیشتر توضیح میدم