عنوان: **آشنایی با mocking frameworks (چارچوب‌های تقلید) - قسمت اول**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۸/۰۲/۱۵ ۲۱:۵۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing

این مطلب در ادامه‌ی مطالب آزمون‌های واحد یا unit testing است.
نوشتن آزمون واحد برای کلاس‌هایی که با یک سری از الگوریتم‌ها ، مسایل ریاضی و امثال آن سر و کار دارند، ساده است. عموما این نوع کلاس‌ها وابستگی خارجی آنچنانی ندارند؛ اما در عمل کلاس‌های ما ممکن است وابستگی‌های خارجی بسیاری پیدا کنند؛ برای مثال کار با دیتابیس، اتصال به یک وب سرویس، دریافت فایل از اینترنت، خواندن اطلاعات از انواع فایل‌ها و غیره.
مطابق اصول آزمایشات واحد، یک آزمون واحد خوب باید ایزوله باشد. نباید به مرزهای سیستم‌های دیگر وارد شده و عملکرد سیستم‌های خارج از کلاس را بررسی کند.
این مثال ساده را در نظر بگیرید:
فرض کنید برنامه شما قرار است از یک وب سرویس لیستی از آدرس‌های IP یک کشور خاص را دریافت کند و در یک دیتابیس محلی آن‌ها را ذخیره نماید. به صورت متداول این کلاس باید اتصالی را به وب سرویس گشوده و اطلاعات را دریافت کند و همچنین آن‌ها را خارج از مرز کلاس در یک دیتابیس ثبت کند. نوشتن آزمون واحد برای این کلاس مطابق اصول مربوطه غیر ممکن است. اگر کلاس آزمون واحد آن‌را تهیه نمائید، این آزمون، integration test نام خواهد گرفت زیرا از مرزهای سیستم باید عبور نماید.

همچنین یک آزمون واحد باید تا حد ممکن سریع باشد تا برنامه نویس از انجام آن بر روی یک پروژه بزرگ منصرف نگردد و ایجاد این اتصالات در خارج از سیستم، بیشتر سبب کندی کار خواهند شد.

برای این ایزوله سازی روش‌های مختلفی وجود دارند که در ادامه به آن‌ها خواهیم پرداخت:

روش اول: استفاده از اینترفیس‌ها
با کمک یک اینترفیس می‌توان مشخص کرد که یک قطعه از کد "چه کاری" را قرار است انجام دهد؛ و نه اینکه "چگونه" باید آن‌را به انجام رساند.
یک مثال ساده از خود دات نت فریم ورک، اینترفیس IComparable است:

```
public static string GetComparisonText(IComparable a, IComparable b)
   {
       if (a.CompareTo(b) == 1)
           return "a is bigger";
       if (a.CompareTo(b) == -1)
           return "b is bigger";
       return "same";
   }
```

در این مثال چون از IComparable استفاده شده، متد ما از هر نوع داده‌ای جهت مقایسه می‌تواند استفاده کند. تنها موردی که برای آن مهم خواهد بود این است که a راهی را برای مقایسه با b ارائه دهد.

اکنون با توجه به این توضیحات، برای ایزوله کردن ارتباط با دیتابیس و وب سرویس در مثال فوق، می‌توان اینترفیس‌های زیر را تدارک دید:

```
public interface IEmailSource
   {
       IEnumerable<string> GetEmailAddresses();
   }

   public interface IEmailDataStore
   {
       void SaveEmailAddresses(IEnumerable<string> emailAddresses);

   }
```

در اینجا استفاده و تعریف اینترفیس‌ها چندین خاصیت را به همراه خواهد داشت :
الف) به این صورت تنها مشخص می‌شود که چه کاری را قصد داریم انجام دهیم و کاری به پیاده سازی آن نداریم.
ب) ساخت کلاس بدون وجود یا دسترسی به یک دیتابیس میسر می‌شود. این مورد خصوصا در یک کار تیمی که قسمت‌های مختلف کار به صورت همزمان در حالت پیشرفت و تهیه است حائز اهمیت می‌شود.
ج) با توجه به اینکه در اینجا به پیاده سازی توجهی نداریم، می‌توان از این اینترفیس‌ها جهت تقلید دنیای واقعی استفاده کنیم. (که در اینجا mocking نام گرفته است)

جهت تقلید رفتار و عملکرد این دو اینترفیس، به کلاس‌های تقلید زیر خواهیم رسید:

```
public class MockEmailSource : IEmailSource
{
   public IEnumerable<string> EmailAddressesToReturn { get; set; }
   public IEnumerable<string> GetEmailAddresses()
   {
       return EmailAddressesToReturn;
   }
}

public class MockEmailDataStore : IEmailDataStore
{
   public IEnumerable<string> SavedEmailAddresses { get; set; }
   public void SaveEmailAddresses(IEnumerable<string> emailAddresses)
   {
       SavedEmailAddresses = emailAddresses;
   }

}
```

تا اینجا اولین قدم در مورد ایزوله سازی کلاس‌هایی که به مرز سیستم‌های دیگر وارد می‌شوند، برداشته شد. اما به مرور زمان مدیریت این اینترفیس‌ها و افزودن رفتارهای جدید به کلاس‌های مشتق شده از آن‌ها مشکل می‌شود. به همین جهت تا حد ممکن از پیاده سازی دستی آن‌ها خودداری شده و روش پیشنهادی استفاده از mocking frameworks است.

ادامه دارد ....