عنوان: **آشنایی با Refactoring - قسمت 2**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۷/۱۳ ۰۰:۰۵:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Refactoring

قسمت دوم آشنایی با Refactoring به معرفی روش « **استخراج متدها** » اختصاص دارد. این نوع Refactoring بسیار ساده بوده و مزایای بسیاری را به همراه دارد؛ منجمله:

- بالا بردن خوانایی کد؛ از این جهت که منطق طولانی یک متد به متدهای کوچکتری با نام‌های مفهوم شکسته می‌شود.
- به این ترتیب نیاز به مستند سازی کدها نیز بسیار کاهش خواهد یافت. بنابراین در یک متد، هر جایی که نیاز به نوشتن کامنت وجود داشت، یعنی باید همینجا آن قسمت را جدا کرده و در متد دیگری که نام آن، همان خلاصه کامنت مورد نظر است، قرار داد.
- این نوع جدا سازی منطق‌های پیاده سازی قسمت‌های مختلف یک متد، در آینده نگهداری کد نهایی را نیز ساده‌تر کرده و انجام تغییرات بر روی آن را نیز تسهیل می‌بخشد؛ زیرا اینبار بجای هراس از دستکاری یک متد طولانی، با چند متد کوچک و مشخص سروکار داریم.

برای نمونه به مثال زیر دقت کنید:

```
using System.Collections.Generic;

namespace Refactoring.Day2.ExtractMethod.Before
{
    public class Receipt
    {
        private IList<decimal> Discounts { get; set; }
        private IList<decimal> ItemTotals { get; set; }

        public decimal CalculateGrandTotal()
        {
            // Calculate SubTotal
            decimal subTotal = 0m;
            foreach (decimal itemTotal in ItemTotals)
                subTotal += itemTotal;

            // Calculate Discounts
            if (Discounts.Count > 0)
            {
                foreach (decimal discount in Discounts)
                    subTotal -= discount;
            }

            // Calculate Tax
            decimal tax = subTotal * 0.065m;
            subTotal += tax;

            return subTotal;
        }
    }
}
```

همانطور که از کامنت‌های داخل متد CalculateGrandTotal مشخص است، این متد سه کار مختلف را انجام می‌دهد؛ جمع اعداد، اعمال تخفیف، اعمال مالیات و نهایتا یک نتیجه را باز می‌گرداند. بنابراین بهتر است هر عمل را به یک متد جداگانه و مشخص منتقل کرده و کامنت‌های ذکر شده را نیز حذف کنیم. نام یک متد باید به اندازه‌ی کافی مشخص و مفهوم باشد و آنچنان نیازی به مستندات خاصی نداشته باشد:

```
using System.Collections.Generic;

namespace Refactoring.Day2.ExtractMethod.After
{
    public class Receipt
```

```
    {
        private IList<decimal> Discounts { get; set; }
        private IList<decimal> ItemTotals { get; set; }

        public decimal CalculateGrandTotal()
        {
            decimal subTotal = CalculateSubTotal();
            subTotal = CalculateDiscounts(subTotal);
            subTotal = CalculateTax(subTotal);
            return subTotal;
        }

        private decimal CalculateTax(decimal subTotal)
        {
            decimal tax = subTotal * 0.065m;
            subTotal += tax;
            return subTotal;
        }

        private decimal CalculateDiscounts(decimal subTotal)
        {
            if (Discounts.Count > 0)
            {
                foreach (decimal discount in Discounts)
                    subTotal -= discount;
            }
            return subTotal;
        }

        private decimal CalculateSubTotal()
        {
            decimal subTotal = 0m;
            foreach (decimal itemTotal in ItemTotals)
                subTotal += itemTotal;
            return subTotal;
        }
    }
}
```

بهتر شد! عملکرد کد نهایی، تغییری نکرده اما کیفیت کد ما بهبود یافته است (همان مفهوم و معنای Refactoring). خوانایی کد افزایش یافته است. نیاز به کامنت نویسی به شدت کاهش پیدا کرده و از همه مهم‌تر، اعمال مختلف، در متدهای خاص آن‌ها قرار گرفته‌اند.
به همین جهت اگر حین کد نویسی، به یک متد طولانی برخوردید (این مورد بسیار شایع است)، در ابتدا حداقل کاری را که جهت بهبود کیفیت آن می‌توانید انجام دهید، «استخراج متدها» است.

ابزارهای کمکی جهت پیاده سازی روش «استخراج متدها»:

- ابزار Refactoring توکار ویژوال استودیو پس از انتخاب یک قطعه کد و سپس کلیک راست و انتخاب گزینه‌ی Refactor-> Extract method، این عملیات را به خوبی می‌تواند مدیریت کند و در وقت شما صرفه جویی خواهد کرد.
- افزونه‌های ReSharper و همچنین CodeRush نیز چنین قابلیتی را ارائه می‌دهند؛ البته توانمندی‌های آن‌ها از ابزار توکار یاد شده بیشتر است. برای مثال اگر در میانه کد شما جایی return وجود داشته باشد، گزینه‌ی Extract method ویژوال استودیو کار نخواهد کرد. اما سایر ابزارهای یاده شده به خوبی از پس این موارد و سایر موارد پیشرفته‌تر بر می‌آیند.

نتیجه گیری:
نوشتن کامنت، داخل بدنه‌ی یک متد مزموم است؛ حداقل به دو دلیل:
- ابزارهای خودکار مستند سازی از روی کامنت‌های نوشته شده، از این نوع کامنت‌ها صرفنظر خواهند کرد و در کتابخانه‌ی شما مدفون خواهند شد (یک کار بی‌حاصل).
- وجود کامنت در داخل بدنه‌ی یک متد، نمود آشکار ضعف شما در کپسوله سازی منطق مرتبط با آن قسمت است.

و ... «لطفا» این نوع پیاده سازی‌ها را خارج از فایل code behind هر نوع برنامه‌ی winform/wpf/asp.net و غیره قرار دهید. تا حد امکان سعی کنید این مکان‌ها، استفاده کننده‌ی «نهایی» منطق‌های پیاده سازی شده توسط کلاس‌های دیگر باشند؛ نه اینکه خودشان

محل اصلی قرارگیری و ابتدای تعریف منطق‌های مورد نیاز قسمت‌های مختلف همان فرم مورد نظر باشند. «لطفا» یک فرم درست نکنید با 3000 سطر کد که در قسمت code behind آن قرار گرفته‌اند. code behind را محل «نهایی» ارائه کار قرار دهید؛ نه نقطه‌ی آغاز تعریف منطق‌های پیاده سازی کار. این برنامه نویسی چندلایه که از آن صحبت می‌شود، فقط مرتبط با کار با بانک‌های اطلاعاتی نیست. در همین مثال، کدهای فرم برنامه، باید نقطه‌ی نهایی نمایش عملیات محاسبه مالیات باشند؛ نه اینکه همانجا دوستانه یک قسمت مالیات حساب شود، یک قسمت تخفیف، یک قسمت جمع بزند، همانجا هم نمایش بدهد! بعد از یک هفته می‌بینید که code behind فرم در حال انفجار است! شده 3000 سطر! بعد هم سؤال می‌پرسید که چرا اینقدر میل به «بازنویسی» سیستم این اطراف زیاد است! برنامه نویس حاضر است کل کار را از صفر بنویسد، بجای اینکه با این شاهکار بخواهد سرو کله بزند! هر چند یکی از روش‌های برخورد با این نوع کدها جهت کاهش هراس نگهداری آن‌ها، شروع به Refactoring است.