عنوان: **آشنایی با Refactoring - قسمت 11**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۸/۰۱ ۲۳:۰۸:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Refactoring

قسمت یازدهم آشنایی با Refactoring به توصیه‌هایی جهت بالا بردن خوانایی تعاریف مرتبط با اعمال شرطی می‌پردازد.

الف) شرط‌های ترکیبی را کپسوله کنید

عموما حین تعریف شرط‌های ترکیبی، هدف اصلی از تعریف آن‌ها پشت انبوهی از && و || گم می‌شود و برای بیان مقصود، نیاز به نوشتن کامنت خواهند داشت. مانند:

```
using System;

namespace Refactoring.Day11.EncapsulateConditional.Before
{
    public class Element
    {
        private string[] Data { get; set; }
        private string Name { get; set; }
        private int CreatedYear { get; set; }

        public string FindElement()
        {
            if (Data.Length > 1 && Name == "E1" && CreatedYear > DateTime.Now.Year - 1)
                return "Element1";

            if (Data.Length > 2 && Name == "RCA" && CreatedYear > DateTime.Now.Year - 2)
                return "Element2";

            return string.Empty;
        }
    }
}
```

برای بالا بردن خوانایی این نوع کدها که برنامه نویس در همین لحظه‌ی تعریف آن‌ها دقیقا می‌داند که چه چیزی مقصود اوست، بهتر است هر یک از شرط‌ها را تبدیل به یک خاصیت با معنا کرده و جایگزین کنیم. برای مثال مانند:

```
using System;

namespace Refactoring.Day11.EncapsulateConditional.After
{
    public class Element
    {
        private string[] Data { get; set; }
        private string Name { get; set; }
        private int CreatedYear { get; set; }

        public string FindElement()
        {
            if (hasOneYearOldElement)
                return "Element1";

            if (hasTwoYearsOldElement)
                return "Element2";

            return string.Empty;
        }

        private bool hasTwoYearsOldElement
        {
```

```
            get { return Data.Length > 2 && Name == "RCA" && CreatedYear > DateTime.Now.Year - 2; }
        }

        private bool hasOneYearOldElement
        {
            get { return Data.Length > 1 && Name == "E1" && CreatedYear > DateTime.Now.Year - 1; }
        }
    }
}
```

همانطور که ملاحظه می‌کنید پس از این جایگزینی، خوانایی متد FindElement بهبود یافته است و برنامه نویس اگر 6 ماه بعد به این کدها مراجعه کند نخواهد گفت: «من این کدها رو نوشتم؟!»؛ چه برسد به سایرینی که احتمالا قرار است با این کدها کار کرده و یا آن‌ها را نگهداری کنند.

ب) از تعریف خواص Boolean با نام‌های منفی پرهیز کنید

یکی از مواردی که عموما علت اصلی بروز بسیاری از خطاها در برنامه است، استفاده از نام‌های منفی جهت تعریف خواص است. برای مثال در کلاس مشتری زیر ابتدا باید فکر کنیم که مشتری‌های علامتگذاری شده کدام‌ها هستند که حالا علامتگذاری نشده‌ها به این ترتیب تعریف شده‌اند.

```
namespace Refactoring.Day11.RemoveDoubleNegative.Before
{
    public class Customer
    {
        public decimal Balance { get; set; }

        public bool IsNotFlagged
        {
            get { return Balance > 30m; }
        }
    }
}
```

همچنین از تعریف این نوع خواص در فایل‌های کانفیگ برنامه‌ها نیز جدا پرهیز کنید؛ چون عموما کاربران برنامه‌ها با این نوع نامگذاری‌های منفی، مشکل مفهومی دارند.

Refactoring قطعه کد فوق بسیار ساده است و تنها با معکوس کردن شرط و نحوه‌ی نامگذاری خاصیت IsNotFlagged پایان می‌یابد:

```
namespace Refactoring.Day11.RemoveDoubleNegative.After
{
    public class Customer
    {
        public decimal Balance { get; set; }

        public bool IsFlagged
        {
            get { return Balance <= 30m; }
        }
    }
}
```