عنوان: آشنایی با Refactoring - قسمت 14
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۹/۰۵ ۱۲:۵۶:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Refactoring

در بسیاری از زبان‌های برنامه نویسی امکان null بودن Reference types وجود دارد. به همین جهت مرسوم است پیش از استفاده از آن‌ها، بررسی شود آیا شیء مورد استفاده نال است یا خیر و سپس برای مثال متد یا خاصیت مرتبط با آن فراخوانی گردد؛ در غیر اینصورت برنامه با یک استثناء خاتمه خواهد یافت.
مشکلی هم که با این نوع بررسی‌ها وجود دارد این است که پس از مدتی کد موجود را تبدیل به مخزنی از انبوهی از if و else ها خواهند کرد که هم درجه‌ی پیچیدگی متدها را افزایش می‌دهند و هم نگهداری آن‌ها را در طول زمان مشکل می‌سازند. برای حل این مساله، الگوی استانداردی وجود دارد به نام null object pattern ؛ به این معنا که بجای بازگشت دادن null و یا سبب بروز یک exception شدن، بهتر است واقعا مطابق شرایط آن متد یا خاصیت، «هیچ‌کاری» را انجام نداد. در ادامه، توضیحاتی در مورد نحوه‌ی پیاده سازی این «هیچ‌کاری» را انجام ندادن، ارائه خواهد شد.

الف) حین معرفی خاصیت‌ها از محافظ استفاده کنید.

برای مثال اگر قرار است خاصیتی به نام Name را تعریف کنید که از نوع رشته است، حالت امن آن رشته بجای null بودن، «خالی» بودن است. به این ترتیب مصرف کننده مدام نگران این نخواهد بود که آیا الان این Name نال است یا خیر. مدام نیاز نخواهد داشت تا if و else بنویسد تا این مساله را چک کند. نحوه پیاده سازی آن هم ساده است و در ادامه بیان شده است:

```
private string name = string.Empty;
public string Name
{
    get { return this.name; }
    set
    {
        if (value == null)
        {
            this.name = "";
            return;
        }
        this.name = value;
    }
}
```

دقیقا در زمان انتساب مقداری به این خاصیت، بررسی می‌شود که آیا مثلا null است یا خیر. اگر بود، همینجا و نه در کل برنامه، مقدار آن «خالی» قرار داده می‌شود.

ب) سعی کنید در متدها تا حد امکان null بازگشت ندهید.

برای نمونه اگر متدی قرار است لیستی را بازگشت دهد:

```
public IList<string> GetCultures()
{
    //...
}
```

و حین تهیه این لیست، عضوی مطابق منطق پیاده سازی آن یافت نشد، null را بازگشت ندهید؛ یک new List خالی را بازگشت

دهید. به این ترتیب مصرف کننده دیگری نیازی به بررسی نال بودن خروجی این متد نخواهد داشت.

ج) از متدهای الحاقی بجای if و else استفاده کنید.

پیاده سازی حالت الف زمانی میسر خواهد بود که طراح اصلی ما باشیم و کدهای برنامه کاملا در اختیار ما باشند. اما در شرایطی که امکان دستکاری کدهای یک کتابخانه پایه را نداریم چه باید کرد؟ مثلا دسترسی به تعاریف کلاس XElement دات نت فریم ورک را نداریم (یا حتی اگر هم داریم، تغییر آن تا زمانیکه در کدهای پایه اعمال نشده باشد، منطقی نیست). در این حالت می‌شود یک یا چند extension method را طراحی کرد:

```
public static class LanguageExtender
{
        public static string GetSafeStringValue(this XElement input)
        {
            return (input == null) ? string.Empty : input.Value;
        }

        public static DateTime GetSafeDateValue(this XElement input)
        {
            return (input == null) ? DateTime.MinValue : DateTime.Parse(input.Value);
        }
}
```

به این ترتیب می‌توان امکانات کلاس پایه‌ای را بدون نیاز به دسترسی به کدهای اصلی آن مطابق نیازهای خود تغییر و توسعه داد.