عنوان: **C# 6 - Null-conditional operators**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۴/۰۷/۱۲ ۱۴:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, C# 6.0

برنامه نویس‌های سی‌شارپ پیشتر با null-coalescing operator یا ?? آشنا شده بودند. برای مثال

```
string data = null;
var result = data ?? "value";
```

در این حالت اگر data یا سمت چپ عملگر، نال باشد، مقدار value (سمت راست عملگر) بازگشت داده خواهد شد؛ که در حقیقت خلاصه شده‌ی چند سطر ذیل است:

```
if (data == null)
{
    data = "value";
}
var result = data;
```

در سی شارپ 6، جهت تکمیل عملگرهای کار با مقادیر نال و بالا بردن productivity برنامه نویس‌ها، عملگر دیگری به نام Null-conditional operator و یا .? به این مجموعه اضافه شده‌است. در این حالت ابتدا مقدار سمت چپ عملگر بررسی خواهد شد. اگر مقدار آن مساوی نال بود، در همینجا کار خاتمه یافته و نال بازگشت داده می‌شود. در غیر اینصورت کار بررسی زنجیره‌ی جاری ادامه خواهد یافت.
برای مثال بسیاری از نتایج بازگشتی از متدها، چند سطحی هستند:

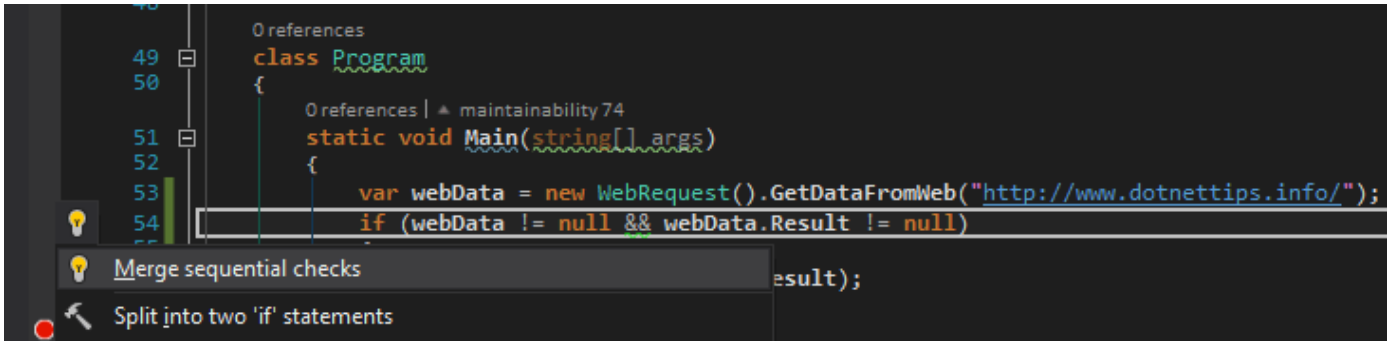
```
class Response
{
    public string Result { set; get; }
    public int Code { set; get; }
}


class WebRequest
{
    public Response GetDataFromWeb(string url)
    {
        // ...
        return new Response { Result = null };
    }
}
```

در اینجا روش مرسوم کار با کلاس درخواست اطلاعات از وب به صورت ذیل است:

```
var webData = new WebRequest().GetDataFromWeb("http://www.dotnettips.info/");
if (webData != null && webData.Result != null)
{
    Console.WriteLine(webData.Result);
}
```

چون می‌خواهیم به خاصیت Result دسترسی پیدا کنیم، نیاز است دو مرحله وضعیت خروجی متد و همچنین خاصیت Result آن‌را جهت مشخص سازی نال نبودن آن‌ها، بررسی کنیم و اگر برای مثال خاصیت Result نیز خود متشکل از یک کلاس دیگر بود که در آن برای مثال StatusCode نیز ذکر شده بود، این بررسی به سه سطح یا بیشتر نیز ادامه پیدا می‌کرد.
در این حالت اگر اشاره‌گر را به محل && انتقال دهیم، افزونه‌ی ReSharper پیشنهاد یکی کردن این بررسی‌ها را ارائه می‌دهد:



به این ترتیب تمام چند سطح بررسی نال، به یک عبارت بررسی .? دار، خلاصه خواهد شد:

```
if (webData?.Result != null)
{
    Console.WriteLine(webData.Result);
}
```

در اینجا ابتدا بررسی می‌شود که آیا webData نال است یا خیر؟ اگر نال بود همینجا کار خاتمه پیدا می‌کند و به بررسی Result نمی‌رسد. اگر نال نبود، ادامه‌ی زنجیره تا به انتها بررسی می‌شود.
البته باید دقت داشت که برای تمام سطوح باید از .? استفاده کرد (برای مثال response?.Results?.Status)؛ در غیر اینصورت همانند سابق در صورت استفاده‌ی از دات معمولی، به یک null reference exception می‌رسیم.

کار با متدها و Delegates

این عملگر جدید مقایسه‌ی با نال را بر روی متدها (علاوه بر خواص و فیلدها) نیز می‌توان بکار برد. برای مثال خلاصه شده‌ی فراخوانی ذیل:

```
if (x != null)
{
   x.Dispose();
}
```

با استفاده از Null Conditional Operator به این صورت است:

```
x?.Dispose();
```

و یا بکار گیری آن بر روی delegates (روش قدیمی):

```
var copy = OnMyEvent;
if (copy != null)
{
   copy(this, new EventArgs());
}
```

نیز با استفاده از متد Invoke به نحو ذیل قابل انجام است و نکته جالب یک سطر کد ذیل علاوه بر ساده شدن آن:

```
OnMyEvent?.Invoke(this, new EventArgs());
```

Thread-safe بودن آن نیز می‌باشد. زیرا در این حالت کامپایلر delegate را به یک متغیر موقتی کپی کرده و سپس فراخوانی‌ها را انجام می‌دهد. اگر انجام این کپی موقت صورت نمی‌گرفت، در حین فراخوانی آن از طریق چندین ترد مختلف، ممکن بود یکی از

مشترکین delegate از آن قطع اشتراک می‌کرد و در این حالت فراخوانی تردی دیگر در همان لحظه، سبب کرش برنامه می‌شد.

استفاده از Null Conditional Operator بر روی Value types

الف) مقایسه با نال

کد ذیل را درنظر بگیرید:

```
var code = webData?.Code;
```

در اینجا Code یک value type از نوع int است. در این حالت با بکارگیری Null Conditional Operator، خروجی این حاصل، از نوع Nullable<int> و یا int? درنظر گرفته خواهد شد و با توجه به اینکه عبارات 0<null و همچنین 0>null هر دو false هستند، مقایسه‌ی این خروجی با 0 بدون مشکل انجام می‌شود. برای مثال مقایسه‌ی ذیل از نظر کامپایلر یک عبارت معتبر است و بدون مشکل کامپایل می‌شود:

```
if (webData?.Code > 0)
{

}
```

ب) بازگشت مقدار پیش فرض دیگری بجای نال

اگر نیاز بود بجای null مقدار پیش فرض دیگری را بازگشت دهیم، می‌توان از null-coalescing operator سابق استفاده کرد:

```
int count = response?.Results?.Count ?? 0;
```

در این مثال خاصیت CountT در اصل از نوع int تعریف شده‌است؛ اما بکارگیری .? سبب Nullable شدن آن خواهد شد. بنابراین امکان بکارگیری عملگر ?? یا null-coalescing operator نیز بر روی این متغیر وجود دارد.

ج) دسترسی به مقدار Value یک متغیر nullable

نمونه‌ی دیگر آن قطعه کد ذیل است:

```
int? x = 10;
//var value = x?.Value; // invalid
Console.WriteLine(x?.ToString());
```

در اینجا برخلاف متغیر Code که از ابتدا nullable تعریف نشده‌است، متغیر x نال پذیر است. اما باید دقت داشت که با تعریف .? دیگر نیازی به استفاده از خاصیت Value این متغیر nullable نیست؛ زیرا .? سبب محاسبه و بازگشت خروجی آن می‌شود. بنابراین در این حالت، سطر دوم غیرمعتبر است (کامپایل نمی‌شود) و سطر سوم معتبر.

کار با indexer property و بررسی نال

اگر به عنوان بحث دقت کرده باشید، یک s جمع در انتهای **s** Null-conditional operator ذکر شده‌است. به این معنا که این عملگر مقایسه‌ی با نال، صرفا یک شکل و فرم .? را ندارد. مثال ذیل در حین کار با آرایه‌ها و لیست‌ها بسیار مشاهده می‌شود:

```
if (response != null && response.Results != null && response.Results.Addresses != null
  && response.Results.Addresses[0] != null && response.Results.Addresses[0].Zip == "63368")
{

}
```

در اینجا به علت بکارگیری indexer بر روی Addresses، دیگر نمی‌توان از عملگر .? که صرفا برای فیلدها، خواص، متدها و delegates طراحی شده‌است، استفاده کرد. به همین منظور، عملگر بررسی نال دیگری به شکل [...]? برای این بررسی طراحی

شده‌است:

```
if(response?.Results?.Addresses?[0]?.Zip == "63368")
{

}
```

به این ترتیب 5 سطح بررسی نال فوق، به یک عبارت کوتاه کاهش می‌یابد.

موارد استفاده‌ی ناصحیح از عملگرهای مقایسه‌ی با نال

خوب، عملگر .? کار مقایسه‌ی با نال را خصوصا در دسترسی‌های چند سطحی به خواص و متدها بسیار ساده می‌کند. اما آیا باید در همه جا از آن استفاده کرد؟ آیا باید از این پس کلا استفاده از دات را فراموش کرد و بجای آن از .? در همه جا استفاده کرد؟
مثال ذیل را درنظر بگیرید:

```
public void DoSomething(Customer customer)
{
    string address = customer?.Employees
                  ?.SingleOrDefault(x => x.IsAdmin)?.Address?.ToString();
    SendPackage(address);
}
```

در این مثال در تمام سطوح آن از .? بجای دات استفاده شده‌است و بدون مشکل کامپایل می‌شود. اما این نوع فراخوانی سبب خواهد شد تا یک سری از مشکلات موجود کاملا مخفی شوند؛ خصوصا اعتبارسنجی‌ها. برای مثال در این فراخوانی اگر مشتری نال باشد یا اگر کارمندانی را نداشته باشد، آدرسی بازگشت داده نمی‌شود. بنابراین حداقل دو سطح بررسی و اعتبارسنجی عدم وجود مشتری یا عدم وجود کارمندان آن در اینجا مخفی شده‌اند و دیگر مشخص نیست که علت بازگشت نال چه بوده‌است.
روش بهتر انجام اینکار، بررسی وضعیت customer و انتقال مابقی زنجیره‌ی LINQ به یک متد مجزای دیگر است:

```
public void DoSomething(Customer customer)
{
   Contract.Requires(customer != null);
   string address = customer.GetAdminAddress();
   SendPackage(address);
}
```

نظرات خوانندگان

نویسنده: امیر ح کریمی
تاریخ: ۱۳۹۴/۰۷/۱۹ ۱۴:۳۶

با سلام
برای چک کردن مقادیر نال پی در پی واقعا کاربردی است
البته موردی که ابتدای مطلب اومده اشکال کوچکی دارد :

```
string data = null;
var result = data ?? "value";
```

و

```
if (data == null)
{
    data = "value";
}
var result = data;
```

یکی نیستند چون در کد دوم مقدار data تغییر می‌کند(در صورتیکه برابر نال باشد).