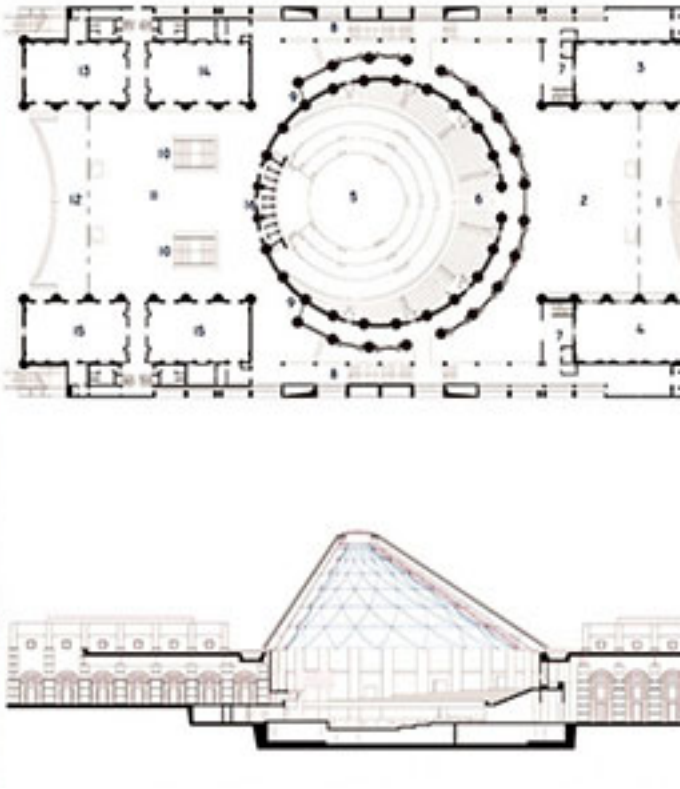
عنوان: **مفاهیم برنامه نویسی ـ مروری بر کلاس و شیء**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۱۹ ۲۱:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

من قصد دارم در قالب چند مطلب برخی از مفاهیم پایه و مهم برنامه نویسی را که پیش نیازی برای درک اکثر مطالب موجود در وب سایت است به زبان ساده بیان کنم تا دایره افرادی که می‌توانند از مطالب ارزشمند این وب سایت استفاده کنند وسعت بیشتری پیدا کند. لازم به توضیح است از آنجا که علاقه ندارم اینجا تبدیل به نسخه فارسی MSDN یا کتاب آنلاین آموزش برنامه نویسی شود این سری آموزش‌ها بیشتر شامل مفاهیم کلیدی خواهند بود.
این مطلب به عنوان اولین بخش از این سری مطالب منتشر می‌شود.

هدف این نوشته بررسی جزییات برنامه نویسی در رابطه با کلاس و شیء نیست. بلکه دریافتن چگونگی شکل گرفتن ایده شیء گرایی و علت مفید بودن آن است.

مشاهده مفاهیم شیء گرایی در پیرامون خود حتماً در دنیای برنامه نویسی شیء گرا بارها با کلمات کلاس و شیء روبرو شده اید. درک صحیح از این مفاهیم بسیار مهم و البته بسیار ساده است. کار را با یک مثال شروع می‌کنیم. به تصویر زیر نگاه کنید.



در سمت راست بخشی از نقشه یک ساختمان و در سمت چپ ساختمان ساخته شده بر اساس این نقشه را می‌بینید. ساختمان همان شیء است. و نقشه ساختمان کلاس آن است چراکه امکان ایجاد اشیائی که تحت عنوان ساختمان طبقه بندی (کلاس بندی) می‌شوند را فراهم می‌کند. به همین سادگی. کلاس‌ها طرح اولیه، نقشه یا قالبی هستند که جزییات یک شی را توصیف می‌کنند.
حتماً با من موافق هستید اگر بگویم:
در نقشه ساختمان نمی‌توانید زندگی کنید اما در خود ساختمان می‌توانید.
از روی یک نقشه می‌توان به تعداد دلخواه ساختمان ساخت.
هنگامی که در یک ساختمان زندگی می‌کنید نیازی نیست تا دقیقاً بدانید چگونه ساخته شده و مثلاً سیم کشی یا لوله کشی‌های آن

چگونه است! تنها کافیست بدانید برای روشن شدن لامپ باید کلید آن را بزنید.
ساختمان دارای ویژگی هایی مانند متراژ، ضخامت دیوار، تعداد پنجره و ابعاد هر یک و ... است که در هنگام ساخت و بر اساس اطلاعات موجود در نقشه تعیین شده اند.
ساختمان دارای کارکرد هایی است. مانند بالا و پایین رفتن آسانسور و یا باز و بسته شدن درب پارکینگ. هر یک از این کارکردها نیز بر اساس اطلاعات موجود در نقشه پیاده سازی و ساخته شده اند.
ساختمان تمام اجزای لازم برای اینکه از آن بتوانیم استفاده کنیم و به عبارتی در آن بتوانیم زندگی کنیم را در خود دارد.
در محیط پیرامون ما تقریباً هر چیزی را می‌توان در یک دیدگاه شیء تصور کرد. به عبارتی هر چیزی که بتوانید به صورت مستقل در ذهن بیاورید و سپس برخی ویژگی‌ها و رفتارها یا کارکردهای آن‌را برشمارید تا آن چیز را قابل شناسایی کند شیء است. مثلاً من به شما می‌گویم موجودی چهار پا دارد، مو... مو... می‌کند و شیر می‌دهد و ... . شما خواهید گفت گاو! و نمی‌گویید گربه. چرا؟ چون توانستید در ذهن خود موجودیتی را به صورت مستقل تصور کنید و از روی ویژگی‌ها و رفتارش آن‌را دقیقاً شناسایی کنید.
سوال: کلاس یا نقشه ایجاد گاو چیست؟ اگر از من بپرسید خواهم گفت طرح اولیه گاو هم ممکن است وجود داشته باشد البته در اختیار خداوند و با سطح دسترسی ملکوت!
اتومبیل، تلویزیون و ... همگی مثال هایی از اشیاء پیرامون ما در دنیای واقعی هستند که حتماً می‌توانید کلاس یا نقشه ایجاد آن‌ها را نیز بدست آورید و یا ویژگی‌ها و کارکردهای آن‌ها را برشمارید.

مفاهیم شیء گرایی در مهندسی نرم افزار مفاهیمی که تاکنون در مورد دنیای واقعی مرور کردیم همان چیزی است که در دنیای برنامه نویسی ـ به عقیده من دنیای واقعی‌تر از دنیای واقعی ـ با آن سر و کار داریم. علت این امر آن است که اصولاً ایده روش برنامه نویسی شیء گرا با مشاهده محیط پیرامون ما به وجود آمده است.
برای نوشتن برنامه جهت حل یک مسئله بزرگ باید بتوان آن مسئله را به بخش‌های کوچکتری تقسیم نمود. در این رابطه مفهوم شیء و کلاس با همان کیفیتی که در محیط پیرامون ما وجود دارد به صورت مناسبی امکان تقسیم یه مسئله بزرگ به بخش‌های کوچکتر را فراهم می‌کند. و سبب می‌شود هماهنگی و تقارن و تناظر خاصی بین اشیاء برنامه و دنیای واقعی بوجود آید که یکی از مزایای اصلی روش شیء گراست.
از آنجا که در یک برنامه اصولاً همه چیز و همه مفاهیم در قالب کدها و دستورات برنامه معنا دارد، کلاس و شیء نیز چیزی بیش از قطعاتی کد نیستند. قطعه کد هایی که بسته بندی شده اند تا تمام کار مربوط به هدفی که برای آن‌ها در نظر گرفته شده است را انجام دهند.
همان طور که در هر زبان برنامه نویسی دستوراتی برای کارهای مختلف مانند تعریف یک متغیر یا ایجاد یک حلقه و ... در نظر گرفته شده است، در زبان‌های برنامه نویسی شیء گرا نیز دستوراتی وجود دارد تا بتوان قطعه کدی را بر اساس مفهوم کلاس بسته بندی کرد.
به طور مثال قطعه کد زیر را در زبان برنامه نویسی سی شارپ در نظر بگیرید.

```
class Player
{
   public string Name;
   public int Age;
   public void Walk()
   {
      // کدهای مربوط به پیاده سازی راه رفتن
   }
   public void Run()
   {
      // کدهای مربوط به پیاده سازی دویدن
   }
}
```

در این قطعه کد با استفاده از کلمه کلیدی class در زبان سی شارپ کلاسی ایجاد شده است که دارای دو ویژگی نام و سن و دو رفتار راه رفتن و دویدن است.
این کلاس به چه دردی می‌خورد؟ کجا می‌توانیم از این کلاس استفاده کنیم؟
پاسخ این است که این کلاس ممکن است برای ما هیچ سودی نداشته باشد و هیچ کجا نتوانیم از آن استفاده کنیم. اما بیایید فرض کنیم برنامه نویسی هستیم که قصد داریم یک بازی فوتبال بنویسیم. به جای آنکه قطعات کد مجزایی برای هر یک از بازیکنان و کنترل رفتار و ویژگی‌های آنان بنویسیم با اندکی تفکر به این نکته پی می‌بریم که همه بازیکنان مشترکات بسیاری دارند و به عبارتی در یک گروه یا کلاس قابل دسته بندی هستند. پس سعی می‌کنیم نقشه یا قالبی برای بازیکن‌ها ایجاد کنیم که دربردارنده ویژگی‌ها و

رفتارهای آن‌ها باشد.
همان طور که در نقشه ساختمان نمی‌توانیم زندگی کنیم این کلاس هم هنوز آماده انجام کارهای واقعی نیست. چراکه برخی مقادیر هنوز برای آن تنظیم نشده است. مانند نام بازیکن و سن و ....
و همان طور که برای سکونت لازم است ابتدا یک ساختمان از روی نقشه ساختمان بسازیم برای استفاده واقعی از کلاس یاد شده نیز باید از روی آن شیء بسازیم. به این فرآیند وهله سازی یا نمونه سازی نیز می‌گویند. یک زبان برنامه نویسی شیء گرا دستوراتی را برای وهله سازی نیز در نظر گرفته است. در C# کلمه کلیدی new این وظیفه را به عهده دارد.

```
Player objPlayer = new Player();
objPlayer.Name = “Ali Karimi”;
objPlayer.Age = 30;
objPlayer.Run();
```

وقتی فرآیند وهله سازی صورت می‌گیرد یک نمونه یا شیء از آن کلاس در حافظه ساخته می‌شود که در حقیقت می‌توانید آنرا همان کدهای کلاس تصور کنید با این تفاوت که مقداردهی‌های لازم صورت گرفته است. به دلیل تعیین مقادیر لازم، حال شیء تولید شده می‌تواند به خوبی اعمال پیش بینی شده را انجام دهد. توجه نمایید در اینجا پیاده سازی داخلی رفتار دویدن و اینکه مثلاً در هنگام فراخوانی آن چه کدی باید اجرا شود تا تصویر یک بازیکن در حال دویدن در بازی نمایش یابد مد نظر و موضوع بحث ما نیست. بحث ما چگونگی سازماندهی کدها توسط مفهوم کلاس و شیء است. همان طور که مشاهده می‌کنید ما تمام جزییات بازیکن‌ها را یکبار در کلاس پیاده سازی کرده ایم اما به تعداد دلخواه می‌توانیم از روی آن بازیکن‌های مختلف را ایجاد کنیم. همچنین به راحتی رفتار دویدن یک بازیکن را فراخوانی میکنیم بدون آنکه پیاده سازی کامل آن در اختیار و جلوی چشم ما باشد.
تمام آنچه که بازیکن برای انجام امور مربوط به خود نیاز دارد در کلاس بازیکن کپسوله می‌شود. بدیهی است در یک برنامه واقعی ویژگی‌ها و رفتارهای بسیار بیشتری باید برای کلاس بازیکن در نظر گرفته شود. مانند پاس دادن، شوت زدن و غیره.
به این ترتیب ما برای هر برنامه می‌توانیم مسئله اصلی را به تعدادی مسئله کوچکتر تقسیم کنیم و وظیفه حل هر یک از مسائل کوچک را به یک شیء واگذار کنیم. و بر اساس اشیاء تشخیص داده شده کلاس‌های مربوطه را بنویسیم. برنامه نویسی شیء گرا سبب می‌شود تا مسئله توسط تعدادی شیء که دارای نمونه‌های متناظری در دنیای واقعی هستند حل شود که این امر زیبایی و خوانایی و قابلیت نگهداری و توسعه برنامه را بهبود می‌دهد.
احتمالاً تاکنون متوجه شده اید که برای نگهداری ویژگی‌های اشیاء از متغیرها و برای پیاده سازی رفتارها یا کارکردهای اشیاء از توابع استفاده میکنیم.
با توجه به این که هدف این مطلب بررسی مفهوم شیء گرائی بود و نه جزییات برنامه نویسی، بنابراین بیان برخی مفاهیم در این رابطه را که بیشتر در مهندسی نرم افزار معنا دارند تا در دنیای واقعی در مطالب بعدی بررسی می‌کنیم.

نظرات خوانندگان

نویسنده: Hesam
تاریخ: ۱۳۹۲/۰۱/۱۹ ۲۲:۴۵

بسیار عالی :)

نویسنده: من
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۴:۵۶

شروع خوبی بود، آفرین!
من هنوز وقتی جلسه‌ی اول کلاس میپرسم که پراید یک کلاس هست و یا یک آبجکت. همه میگن آبجکتی از کلاس ماشین!
این در حالیست که خیلی از این افراد سالهاست برنامه نویسی میکنند و هنوز نمیدونند heap یا stack چیه

نویسنده: علي
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۴

حالا با توجه به توضیحات بالا که گفته شد « *در نقشه ساختمان نمی‌توانید زندگی کنید اما در خود ساختمان می‌توانید.* » با یک پراید می‌شود رانندگی کرد اما با ماشین که بیشتر یک مفهوم است خیر. نه؟

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۲۲

تشکر از نظر شما. متوجه صحبت شما هستم ولی این توضیح برای دوستان دیگه می‌تونه مفید باشه.
پراید به عنوان رده ای از اتومبیل‌ها کلاس است. اما اگر به کسی یک اتومبیل پراید در حال عبور را نشان دهید که دارای پلاک و ... است، آن شیء ای است از کلاس پراید. اگر در یک پارکینگ تعدادی اتومبیل باشد و از کسی بخواهیم بر اساس نوع گروه بندی یا کلاس بندی کند، بعد از چند دقیقه خواهیم دید اتومبیل‌های هم نوع را جدا کرده. مثلا همه اتومبیل‌های از نوع پراید را کنار هم قرار داده. این همان مفهوم کلاس بندی است. برای تولید اتومبیلی از نوع پراید کارخانه یک نقشه یا طرحی ایجاد می‌کند که بسیاری مشخصات و چگونگی ساخت را در خود دارد. چگونگی انجام برخی کارکردها مانند حرکت را دارد. این طرح کلاس نامیده می‌شود. اما آیا می‌توان برای آن پلاک در نظر گرفت؟ خیر چون فقط نقشه ایجاد اتومبیل است (یا به عبارتی فقط مفهوم است). همچنین کلاس پراید احتمالاً از کلاس اتومبیل یا ماشین برخی خصوصیات و رفتارها را با ارث برده است.

بیاد داشته باشیم هر فردی در هر سطحی از دانش هم مسلماً بسیاری چیزها را هنوز نمی‌داند و دانسته‌های ما در مقابل نادانسته‌ها قطره ای بیش نیست. تا جایی که می‌توان گفت همه ما از نظر نادانسته‌ها برابریم. تفاوت در دانسته هاست. کسانی که سال‌ها برنامه نویسی می‌کنند هم حتما دانسته هایی دارند که می‌توانند این کار را ادامه دهند. پس کافی است آنچه نمی‌دانند را سعی کنیم به اشتراک بگذاریم تا بدانند و از دانسته هایشان استفاده کنیم.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۰ ۱۵:۲۵

جمله با پراید می‌شود رانندگی کرد ابهام دارد. ابهام آن به این صورت رفع می‌شود که من می‌دانم منظور شما از پراید به عنوان یک اسم عام و یک مفهوم و **نام رده یا کلاسی از اتومبیل ها** نیست. بلکه منظور شما با اتومبیل پرایدی است که دارای یک پلاک مشخص است و مثلاً کسی به تازگی گنجی پیدا کرده و رفته یک پراید خریده! آن اتومبیل پراید مشخص یک شیء است از کلاس پراید. بله با آن شی می‌توان رانندگی کرد. اما با مفهوم یا نقشه یا کلاس یا رده یا گروه یا طرح تولید خودروی پراید یا هر ماشین دیگری نمی‌توان رانندگی کرد.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۱/۲۱ ۱۵:۴۸

ممنون، خیلی خوب بررسی کرده بودید و مثالهایی که هم ارائه کردید به دلم نشست، امیدوارم برای سایر مباحث هم به همین صورت ادامه بدید. البته به نظر بنده نیازی هم نیست خیلی روی این مبحث تناظر بین دنیای واقعی و دنیای شی گرا حساس باشیم، چون اگر به همین نحو بخواهیم این دو را با هم مقایسه کنیم شاید بعضی جاها با مشکل مواجه بشیم. این تناظر فقط برای اینه که دید و تجسمی از این مفهوم داشته باشیم تا بهتر بتوانیم آن را بپذیریم.
یادمون نره که هدف ما یادگیری مفهوم شی گرایی است نه یادگیری تناظر آن با دنیای واقعی، این تناظر فقط یک ابزاره برای دسترسی سریعتر به این هدف!

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۱ ۱۷:۳۵

سلام و ممنون از نظر شما.
اتفاق جالبی افتاد و آن این بود که هم اکنون داشتم در OneNote بخشی برای مطلب بعدی می‌نوشتم. دقیقاً داشتم پاراگرافی را می‌نوشتم که جلوی این که ذهن خواننده به سمتی برود که گویی "الزاماً هر مورد در مهندسی نرم افزار را باید پس از یافتن مصداق آن در محیط اطراف یاد گرفت" را بگیرم.
دقیقاً صحیح است. تاکید بر این تناظر در این بخش به دلیل یافتن درک عمیق‌تر از شیء گرایی و علت مفید بودن آن و چگونگی شکل گیری ایده آن است. این درک عمیق‌تر امکان استفاده بهتر و صحیح‌تر این مفاهیم در برنامه را فراهم می‌کند. و سبب می‌شود برنامه نویس شیءگرایی را ابزاری برای حل مسئله بیاد نه راه و روشی که همه میگن خوبه پس باید رعایت کرد. حال آنکه چون درک دقیقی از آن ندارد در حقیقت مسئله را با آن روشی که بهتر بلد است حل می‌کنند و فقط تعدادی کلاس و شیء در برنامه وجود دارد.

نویسنده: آریانا
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۵۶

بسیار بسیار عالی بود مرسی

نویسنده: سحابی
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۴:۵۹

سلام
برای یادگیری Desing pattern منابعی وجود دارد ؟ لطفا در صورت امکان اگر منابع و یا لینک کارآمدی معرفی کنید .

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۱۴ ۱۷:۱۵

برچسب‌های سایت رو مرور کنید به اندازه کافی در این زمینه مطلب هست. مثلا اینجا

عنوان: **مفاهیم برنامه نویسی ـ مروری بر فیلدها، متدها و ساخت اشیاء**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۱/۲۳ ۲۰:۱۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

شکستن یک مسئله بزرگ به تعدادی مسئله کوچک‌تر راهکار موثری برای حل آن است. این امر در برنامه نویسی نیز که هدف آن چیزی جز حل یک مسئله نیست همواره مورد توجه بوده است. به همین دلیل روش هایی که به کمک آن‌ها بتوان یک برنامه بزرگ را به قطعات کوچکتری تقسیم کرد تا هر قطعه کد مسئول انجام کار خاصی باشد پیشتر به زبان‌های برنامه نویسی اضافه شده اند. یکی از این ساختارها تابع (Function) نام دارد. برنامه ای که از توابع برای تقسیم کدهای برنامه استفاده می‌کند یک برنامه ساخت‌یافته می‌گوییم.

در مطلب پیشین به پیرامون خود نگاه کردیم و اشیاء گوناگونی را مشاهده کردیم که در حقیقت دنیای ما را تشکیل داده اند و فعالیت‌های روزمره ما با استفاده از آن‌ها صورت می‌گیرد. ایده ای به ذهنمان رسید. اشیاء و مفاهیم مرتبط به آن می‌تواند روش بهتر و موثرتری برای تقسیم کدهای برنامه باشد. مثلاً اگر کل کدهای برنامه که مسئول حل یکی از مسئله‌های کوچک یاد شده است را یکجا بسته بندی کنیم و اصولی که از اشیاء واقعی پیرامون خود آموختیم را در مورد آن رعایت کنیم به برنامه بسیار با کیفیت‌تری از نظر خوانایی، راحتی در توسعه، اشکال زدایی ساده‌تر و بسیاری موارد دیگر خواهیم رسید.

توسعه دهندگان زبان‌های برنامه نویسی که با ما در این مورد هم عقیده بوده اند دست به کار شده و دستورات و ساختارهای لازم برای پیاده کردن این ایده را در زبان برنامه نویسی قرار دادند و آن را زبان برنامه نویسی شیء گرا نامیدند. حتی جهت برخورداری از قابلیت استفاده مجدد از کد و موارد دیگر به جای آنکه کدها را در بسته هایی به عنوان یک شیء خاص قرار دهیم آن‌ها را در بسته هایی به عنوان قالب یا نقشه ساخت اشیاء خاصی که در ذهن داریم قرار می‌دهیم. یعنی مفهوم کلاس یا رده که پیشتر اشاره شد. به این ترتیب یک بار می‌نویسیم و بارها استفاده می‌کنیم. مانند همان مثال بازیکن در بخش نخست . هر زمان که لازم باشد با استفاده از دستورات مربوطه از روی کدهای کلاس که نقشه یا قالب ساخت اشیاء هستند شیء مورد نظر را ساخته و در جهت حل مسئله مورد نظر به کار می‌بریم.

حال برای آنکه به طور عملی بتوانیم از ایده شیء گرایی در برنامه هایمان استفاده کنیم و مسائل بزرگ را حل کنیم لازم است ابتدا مقداری با جزییات و دستورات زبان در این مورد آشنا شویم.

تذکر : دقت کنید برای آنکه از ایده شیء گرایی در برنامه‌ها حداکثر استفاده را ببریم مفاهیمی در مهندسی نرم افزار به آن اضافه شده است که ممکن است در دنیای واقعی نیازی به طرح آن‌ها نباشد. پس لطفاً تلاش نکنید با دیدن هر مفهوم تازه بلافاصله سعی در تطبیق آن با محیط اطراف کنید. هر چند بسیاری از آن‌ها به طور ضمنی در اشیاء پیرامون ما نیز وجود دارند.

زبان برنامه نویسی مورد استفاده برای بیان مفاهیم برنامه نویسی در این سری مقالات زبان سی شارپ است. اما درک برنامه‌های نوشته شده برای علاقه مندان به زبان‌های دیگری مانند وی بی دات نت نیز دشوار نیست. چراکه اکثر دستورات مشابه است و تبدیل Syntax نیز به راحتی با اندکی جستجو میسر می‌باشد. لازم به یادآوری است زبان سی شارپ به بزرگی یا کوچکی حروف حساس است.

تشخیص و تعریف کلاس‌های برنامه کار را با یک مثال شروع می‌کنیم. فرض کنید به عنوان بخشی از راه حل یک مسئله بزرگ، لازم است محیط و مساحت یک سری چهارضلعی را محاسبه کنیم و قصد داریم این وظیفه را به طور کامل بر عهده قطعه کدهای مستقلی در برنامه قرار دهیم. به عبارت دیگر قصد داریم متناظر با هر یک از چهارضلعی‌های موجود در مسئله یک شیء در برنامه داشته باشیم که قادر است محیط و مساحت خود را محاسبه و ارائه نماید. کلاس زیر که با زبان سی شارپ نوشته شده امکان ایجاد اشیاء مورد نظر را فراهم می‌کند.

```
public class Rectangle
{
   public double Width;
   public double Height;

   public double Area()
   {
      return Width*Height;
   }

   public double Perimeter()
   {
      return 2*(Width + Height);
   }
```

```
}
```

در این قطعه برنامه نکات زیر قابل توجه است:
کلاس با کلمه کلیدی class تعریف می‌شود.
همان طور که مشاهده می‌کنید تعریف کلاس با کلمه public آغاز شده است. این کلمه محدوده دسترسی به کلاس را تعیین می‌کند. در اینجا از کلمه public استفاده کردیم تا بخش‌های دیگر برنامه امکان استفاده از این کلاس را داشته باشند.
پس از کلمه کلیدی class نوبت به نام کلاس می‌رسد. اگرچه انتخاب نام مورد نظر امری اختیاری است اما در آینده حتماً اصول و قراردادهای نام‌گذاری در دات‌نت را مطالعه نمایید. در حال حاضر حداقل به خاطر داشته باشید تا انتخاب نامی مناسب که گویای کاربرد کلاس باشد بسیار مهم است.
باقیمانده کد، بدنه کلاس را تشکیل می‌دهد. جاییکه ویژگی ها، رفتارها و ... یا به طور کلی اعضای کلاس تعریف می‌شوند.
نکته : کلماتی مانند public که پیش از تعریف کلاس یا اعضای آن قرار می‌گیرند Modifier یا پیرایندە نام دارند. که البته به نظر من ترجمه این گونه واژه‌ها از کارهای شیطان است. بنابراین از این پس بهتر است همان Modifier را به خاطر داشته باشید. از آنجا که public مدیفایری است که سطح دسترسی را تعیین می‌کند به آن یک Access Modifier می‌گویند. در یک بخش از این سری مقالات تمامی مدیفایرها بررسی خواهند شد.

ایجاد شیء از یک کلاس و نحوه دسترسی به شیء ایجاد شده شیء و کلاس چیزهای متفاوتی هستند. یک کلاس نوع یک شیء را تعریف می‌کند. اما یک شیء یک موجودیت عینی و واقعی بر اساس یک کلاس است. در اصطلاح از شیء به عنوان یک نمونه (Instance) یا وهله ای از کلاس مربوطه یاد می‌کنیم. همچنین به عمل ساخت شیء نمونه سازی یا وهله سازی گوییم.
برای ایجاد شیء از کلمه کلیدی new و به دنبال آن نام کلاسی که قصد داریم بر اساس آن یک شیء بسازیم استفاده می‌کنیم. همان طور که اشاره شد کلاس یک نوع را تعریف می‌کند. پس از آن می‌توان همانند سایر انواع مانند int, string, ... برای تعریف متغیر استفاده نمود. به مثال زیر توجه کنید.

```
Rectangle rectangle = new Rectangle();
```

در این مثال rectangle که با حرف کوچک شروع شده و می‌توانست هر نام دلخواه دیگری باشد ارجاعی به شیء ساخته شده را به دست می‌دهد. وقتی نمونه ای از یک کلاس ایجاد می‌شود یک ارجاع به شیء تازه ساخته شده برای برنامه نویس برگشت داده می‌شود. در این مثال rectangle یک ارجاع به شیء تازه ساخته شده است یعنی به آن اشاره می‌کند اما خودش شامل داده‌های آن شیء نیست. تصور کنید این ارجاع تنها دستگیره ای برای شیء ساخته شده است که دسترسی به آن را برای برنامه نویس میسر می‌کند. درک این مطلب از این جهت دارای اهمیت است که بدانید می‌شود یک دستگیره یا ارجاع دیگر بسازید بدون آنکه شیء جدیدی تولید کنید.

```
Rectangle rectangle;
```

البته توصیه نمی‌کنم چنین ارجاعی را تعریف کنید چرا که به هیچ شیء خاصی اشاره نمی‌کند. و تلاش برای استفاده از آن منجر به بروز خطای معروفی در برنامه خواهد شد. به هر حال یک ارجاع می‌توان ساخت چه با ایجاد یک شیء جدید و یا با نسبت دادن یک شیء موجود به آن.

```
Rectangle rectangle1 = new Rectangle();
Rectangle rectangle2 = rectangle1;
```

در این کد دو ارجاع یا دستگیره ایجاد شده است که هر دو به یک شیء اشاره می‌کنند. بنابراین ما با استفاده از هر دو ارجاع می‌توانیم به همان شیء واحد دسترسی پیدا کنیم و اگر مثلاً با rectangle1 در شیء مورد نظر تغییری بدهیم و سپس با rectangle2 شیء را مورد بررسی قرار دهیم تغییرات داده شده قابل مشاهده خواهد بود چون هر دو ارجاع به یک شیء اشاره می‌کنند. به همین دلیل کلاس‌ها را به عنوان نوع ارجاعی می‌شناسیم در مقایسه با انواع داده دیگری که اصطلاحاً نوع مقداری هستند.
حالا می‌توان شیء ساخته شده را با استفاده از ارجاعی که به آن داریم به کار برد.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
rectangle.Height = 10;
double a = rectangle.Area();
```

ابتدا عرض و ارتفاع شیء چهارضلعی را مقدار دهی کرده و سپس مساحت را دریافت کرده ایم. از نقطه برای دسترسی به اعضای یک شیء استفاده می‌شود.

فیلدها اگر به تعریف کلاس دقت کنید مشخص است که دو متغییر Width و Height را با سطح دسترسی عمومی تعریف کرده ایم. به متغیرهایی از هر نوع که مستقیماً درون کلاس تعریف شوند (و نه مثلاً داخل یک تابع درون کلاس) فیلد می‌گوییم. فیلدها از اعضای کلاس دربردارنده آن‌ها محسوب می‌شوند.
تعریف فیلدها مستقیماً در بدنه کلاس با یک Access Modifier شروع می‌شود و به دنبال آن نوع فیلد و سپس نام دلخواه برای فیلد می‌آید.
تذکر : نامگذاری مناسب یکی از مهمترین اصولی است که یک برنامه نویس باید همواره به آن توجه کافی داشته باشد و به شدت در بالا رفتن کیفیت برنامه موثر است. به خاطر داشته باشید تنها اجرا شدن و کار کردن یک برنامه کافی نیست. رعایت بسیاری از اصول مهندسی نرم افزار که ممکن است نقش مستقیمی در کارکرد برنامه نداشته باشند موجب سهولت در نگهداری و توسعه برنامه شده و به همان اندازه کارکرد صحیح برنامه مهم هستند. بنابراین مجدداً شما را دعوت به خواندن مقاله یاد شده بالا در مورد اصول نامگذاری صحیح می‌کنم. هر مفهوم تازه ای که می‌آموزید می‌توانید به اصول نامگذاری همان مورد در مقاله پیش گفته مراجعه نمایید. همچنین افزونه هایی برای Visual Studio وجود دارد که شما را در زمینه نامگذاری صحیح و بسیاری موارد دیگر هدایت می‌کنند که یکی از مهمترین آن‌ها Resharper نام دارد.
مثال:

```
// public field (Generally not recommended.)
public double Width;
```

همان طور که در این قطعه کد به عنوان توضیح درج شده است استفاده از فیلدهایی با دسترسی عمومی توصیه نمی‌شود. علت آن واضح است. چون هیچ کنترلی برای مقداری که برای آن در نظر گرفته می‌شود نداریم. به عنوان مثال امکان دارد یک مقدار منفی برای عرض یا ارتفاع شیء درج شود حال آنکه می‌دانیم عرض یا ارتفاع منفی معنا ندارد. در قسمت بعدی این سری مقالات این مشکل را بررسی و حل خواهیم نمود.
فیلدها معمولاً با سطح دسترسی خصوصی و برای نگهداری از داده‌هایی که مورد نیاز بیش از یک متد (یا تابع) درون کلاس است و آن داده‌ها باید پس از خاتمه کار یک متد همچنان باقی بمانند استفاده می‌شود. بدیهی است در غیر اینصورت به جای تعریف فیلد می‌توان از متغیرهای محلی (متغیری که درون خود تابع تعریف می‌شود) استفاده نمود.
همان طور که پیشتر اشاره شد برای دسترسی به یک فیلد ابتدا یک نقطه پس از نام شیء درج کرده و سپس نام فیلد مورد نظر را می‌نویسیم.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
```

در هنگام تعریف یک فیلد در صورت نیاز می‌توان برای آن یک مقدار اولیه را در نظر گرفت. مانند مثال زیر:

```
public class Rectangle
{
   public double Width = 5;
   // ...
}
```

متدها متدها قطعه کدهایی شامل یک سری دستور هستند. این مجموعه دستورات با فراخوانی متد و تعیین آرگومان‌های مورد نیاز اجرا می‌شوند. در زبان سی شارپ به نوعی تمام دستورات در داخل متدها اجرا می‌شوند. در این زبان تمامی توابع در داخل کلاس‌ها تعریف می‌شوند و بنابراین همه متد هستند.

متدها نیز مانند فیلدها در داخل کلاس تعریف می‌شوند. ابتدا یک Access Modifier سطح دسترسی را تعیین می‌نماید. سپس به ترتیب نوع خروجی، نام متد و لیست پارامترهای آن در صورت وجود درج می‌شود. به مجموعه بخش‌های یاد شده امضای متد می‌گویند.
پارامترهای یک متد داخل یک جفت پرانتز قرار می‌گیرند و با کاما (,) از هم جدا می‌شوند. یک جفت پرانتز خالی نشان دهنده آن است که متد نیاز به هیچ پارامتری ندارد.
بار دیگر به بخش تعریف متدهای کلاسی که ایجاد کردیم توجه نمایید.

```
public class Rectangle
{
   // ...

   public double Area()
   {
      return Width*Height;
   }

   public double Perimeter()
   {
      return 2*(Width + Height);
   }
}
```

در این کلاس دو متد به نام‌های Area و Perimeter به ترتیب برای محاسبه مساحت و محیط چهارضلعی تعریف شده است.
همانطور که پیشتر اشاره شد متدها برای پیاده سازی رفتار اشیاء یا همان کارکردهای آن‌ها استفاده می‌شوند. در این مثال شیء ما قادر است مساحت و محیط خود را محاسبه نماید. چه شیء خوش رفتاری!
همچنین توجه نمایید این شیء برای محاسبه مساحت و محیط خود نگاهی به ویژگی‌های خود یعنی عرض و ارتفاعش که در فیلدهای آن نگهداری می‌کنیم می‌اندازد.
فراخوانی متد یک شیء همانند دسترسی به فیلد آن است. ابتدا نام شیء سپس یک نقطه و به دنبال آن نام متد مورد نظر به همراه پرانتزها. آرگومان‌های مورد نیاز در صورت وجود داخل پرانتزها قرار می‌گیرند و با کاما از هم جدا می‌شوند. که البته در این مثال متد ما نیازی به آرگومان ندارد. به همین دلیل برای فراخوانی آن تنها یک جفت پرانتز خالی قرار می‌دهیم.

در این بخش به دو مفهوم پارامتر و آرگومان اشاره شد. تفاوت آن‌ها چیست؟

در هنگام تعریف یک متد نام و نوع پارامترهای مورد نیاز را تعیین و درج می‌نماییم. حال وقتی قصد فراخوانی متد را داریم باید مقادیر واقعی که آرگومان نامیده می‌شود را برای هر یک از پارامترهای تعریف شده فراهم نماییم. نوع آرگومان باید با نوع پارامتر تعریف شده تطبیق داشته باشد. اما اگر یک متغیر را به عنوان آرگومان در هنگام فراخوانی متد استفاده می‌کنیم نیازی به یکسان بودن نام آن متغیر و نام پارامتر تعریف شده نیست.
متدها می‌توانند یک مقدار را به کدی که آن متد را فراخوانی کرده است بازگشت دهند.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10.5;
rectangle.Height = 10;
double p = rectangle.Perimeter();
```

در این مثال مشاهده می‌کنید که پس از فراخوانی متد Perimeter مقدار بازگشتی آن در متغیری به نام p قرار گرفته است. اگر نوع خروجی یک متد که در هنگام تعریف آن پیش از نام متد قرار می‌گیرد void یا پوچ نباشد، متد می‌تواند مقدار مورد نظر را با استفاده از کلمه کلیدی return بازگشت دهد. کلمه return و به دنبال آن مقداری که از نظر نوع باید با نوع خروجی تعیین شده تطبیق داشته باشد، مقدار درج شده را به کد فراخوان متد بازگشت می‌دهد.
نکته : کلمه return علاوه بر بازگشت مقدار مورد نظر سبب پایان اجرای متد نیز می‌شود. حتی در صورتی که نوع خروجی یک متد void تعریف شده باشد استفاده از کلمه return بدون اینکه مقداری به دنبال آن بیاید می‌تواند برای پایان اجرای متد، در صورت نیاز و مثلاً برقراری شرطی خاص مفید باشد. بدون کلمه return متد زمانی پایان می‌یابد که به پایان قطعه کد بدنه خود برسد. توجه نمایید که در صورتی که نوع خروجی متد چیزی به جز void است استفاده از کلمه return به همراه مقدار مربوطه الزامی است.
مقدار خروجی یک متد را می‌توان هر جایی که مقداری از همان نوع مناسب است مستقیماً به کار برد. همچنین می‌توان آن را در یک متغیر قرار داد و سپس از آن استفاده نمود.
به عنوان مثال کلاس ساده زیر را در نظر بگیرید که متدی دارد برای جمع دو عدد.

```
public class SimpleMath
{
   public int AddTwoNumbers(int number1, int number2)
   {
      return number1 + number2;
   }
}
```

و حال دو روش استفاده از این متد:

```
SimpleMath obj = new SimpleMath();

Console.WriteLine(obj.AddTwoNumbers(1, 2));

int result = obj.AddTwoNumbers(1, 2);
Console.WriteLine(result);
```

در روش اول مستقیماً خروجی متد مورد استفاده قرار گرفته است و در روش دوم ابتدا مقدار خروجی در یک متغیر قرار گرفته است و سپس از مقدار درون متغیر استفاده شده است. استفاده از متغیر برای نگهداری مقدار خروجی اجباری نبوده و تنها جهت بالا بردن خوانایی برنامه یا حفظ مقدار خروجی تابع برای استفاده‌های بعدی به کار می‌رود.

در بخش‌های بعدی بحث ما در مورد سایر اعضای کلاس و برخی جزییات پیرامون اعضای پیش گفته خواهد بود.

نظرات خوانندگان

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۱/۲۴ ۱۱:۵۶

ممنون بابت مطلب آموزشی تون،
تاکیدتان بر استفاده از قرار دادهای نامگذاری، تاکید مثبتی است و واقعا مهم،
کتاب Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries در این زمینه اطلاعات کاملتر و دقیقتری در بر داره.
یکی از روش هایی که در رعایت استاندارهای کد نویسی تاثیر مفیدی داره این هستش که در قطعه کد هایی که به عنوان مثال در آموزش‌ها ارائه می‌شه تا حد امکان سعی بشه این اصول رعایت بشه تا به صورت تدریجی این روش کد نویسی جزئی از عادات برنامه نویسی ما بشود.

عنوان: **مفاهیم برنامه نویسی ـ مروری بر پروپرتی‌ها**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۷:۴۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, OOP, Programming, آموزش برنامه نویسی

در مطلب پیشین کلاسی را برای حل بخشی از یک مسئله بزرگ تهیه کردیم. اگر فراموش کردید پیشنهاد می‌کنم یک بار دیگر آن مطلب را مطالعه کنید. بد نیست بار دیگر نگاهی به آن بیاندازیم.

```
public class Rectangle
{
    public double Width;
    public double Height;

    public double Area()
    {
        return Width*Height;
    }

    public double Perimeter()
    {
        return 2*(Width + Height);
    }
}
```

کلاس خوبی است اما همان طور که در بخش قبل مطرح شد این کلاس می‌تواند بهتر هم باشد. در این کلاس برای نگهداری حالت اشیائی که از روی آن ایجاد خواهند شد از متغیرهایی با سطح دسترسی عمومی استفاده شده است. این متغیرهای عمومی فیلد نامیده می‌شوند. مشکل این است که با این تعریف، اشیاء نمی‌توانند هیچ اعتراضی به مقادیر غیر معتبری که ممکن است به آن‌ها اختصاص داده شود، داشته باشند. به عبارت دیگر هیچ کنترلی بر روی مقادیر فیلدها وجود ندارد. مثلاً ممکن است یک مقدار منفی به فیلد طول اختصاص یابد. حال آنکه طول منفی معنایی ندارد.

تذکر : ممکن است این سوال پیش بیاید که خوب ما کلاس را نوشته ایم و خودمان می‌دانیم چه مقادیری برای فیلدهای آن مناسب است. اما مسئله اینجاست که اولاً ممکن است کلاس تهیه شده توسط برنامه نویس دیگری مورد استفاده قرار گیرد. یا حتی پس از مدتی فراموش کنیم چه مقادیری برای کلاسی که مدتی قبل تهیه کردیم مناسب است. و از همه مهمتر این است که کلاس‌ها و اشیاء به عنوان ابزاری برای حل مسائل هستند و ممکن است مقادیری که به فیلدها اختصاص می‌یابد در زمان نوشتن برنامه مشخص نباشد و در زمان اجرای برنامه توسط کاربر یا کدهای بخش‌های دیگر برنامه تعیین گردد.
به طور کلی هر چه کنترل و نظارت بیشتری بر روی مقادیر انتسابی به اشیاء داشته باشیم برنامه بهتر کار می‌کند و کمتر دچار خطاهای مهلک و بدتر از آن خطاهای منطقی می‌گردد. بنابراین باید ساز و کار این نظارت را در کلاس تعریف نماییم.
برای کلاس‌ها یک نوع عضو دیگر هم می‌توان تعریف کرد که دارای این ساز و کار نظارتی است. این عضو **Property** نام دارد و یک مکانیسم انعطاف پذیر برای خواندن، نوشتن یا حتی محاسبه مقدار یک فیلد خصوصی فراهم می‌نماید.
تا اینجا باید به این نتیجه رسیده باشید که تعریف یک متغیر با سطح دسترسی عمومی در کلاس روش پسندیده و قابل توصیه ای نیست. بنابراین متغیرها را در سطح کلاس به صورت خصوصی تعریف می‌کنیم و از طریق تعریف Property امکان استفاده از آن‌ها در بیرون کلاس را ایجاد می‌کنیم.
حال به چگونگی تعریف Propertyها دقت کنید.

```
public class Rectangle
{
    private double _width = 0;
    private double _height = 0;

    public double Width
    {
        get { return _width; }
        set { if (value > 0) { _width = value; } }
    }

    public double Height
    {
        get { return _height; }
        set { if (value > 0) { _height = value; } }
    }
```

```
    public double Area()
    {
        return _width * _height;
    }

    public double Perimeter()
    {
        return 2*(_width + _height);
    }
}
```

به تغییرات ایجاد شده در تعریف کلاس دقت کنید. ابتدا سطح دسترسی دو متغیر خصوصی شده است یعنی فقط اعضای داخل کلاس به آن دسترسی دارند و از بیرون قابل استفاده نیست. نام متغیرهای پیش گفته بر اساس اصول صحیح نامگذاری فیلدهای خصوصی تغییر داده شده است. ببینید اگر اصول نامگذاری را رعایت کنید چقدر زیباست. هر جای برنامه که چشمتان به _width بخورد فوراً متوجه می‌شوید یک فیلد خصوصی است و نیازی نیست به محل تعریف آن مراجعه کنید. از طرفی یک مقدار پیش فرض برای این دو فیلد در نظر گرفته شده است که در صورتی که مقدار مناسبی برای آن‌ها تعیین نشد مورد استفاده قرار خواهند گرفت.

دو قسمت اضافه شده دیگر تعریف دو Property مورد نظر است. یکی عرض و دیگری ارتفاع. خط اول تعریف پروپرتی تفاوتی با تعریف فیلد عمومی ندارد. اما همان طور که می‌بینید هر فیلد دارای یک بدنه است که با {} مشخص می‌شود. در این بدنه ساز و کار نظارتی تعریف می‌شود.

نحوه دسترسی به پروپرتی‌ها مشابه فیلدهای عمومی است. اما پروپرتی‌ها در حقیقت متدهای ویژه ای به نام اکسسور (Accessor) هستند که از طرفی سادگی استفاده از متغیرها را به ارمغان می‌آورند و از طرف دیگر دربردارنده امنیت و انعطاف پذیری متدها هستند. یعنی در عین حال که روشی عمومی برای داد و ستد مقادیر ارایه می‌دهند، کد پیاده سازی یا وارسی اطلاعات را مخفی نموده و استفاده کننده کلاس را با آن درگیر نمی‌کنند. قطعه کد زیر چگونگی استفاده از پروپرتی را نشان می‌دهد.

```
Rectangle rectangle = new Rectangle();
rectangle.Width = 10;
Console.WriteLine(rectangle.Width);
```

به خوبی مشخص است برای کد استفاده کننده از شیء که آن‌را کد مشتری می‌نامیم نحوه دسترسی به پروپرتی یا فیلد تفاوتی ندارد. در اینجا خط دوم که مقداری را به یک پروپرتی منتسب کرده سبب فراخوانی اکسسور set می‌گردد. همچنین مقدار منتسب شده یعنی ۱۰ در داخل بدنه اکسسور از طریق کلمه کلیدی value قابل دسترسی و ارزیابی است. در خط سوم که لازم است مقدار پروپرتی برای چاپ بازیابی یا خوانده شود منجر به فراخوانی اکسسور get می‌گردد.

تذکر : به دو اکسسور get و set مانند دو متد معمولی نگاه کنید از این نظر که می‌توانید در بدنه آن‌ها اعمال دلخواه دیگری بجز ذخیره و بازیابی اطلاعات پروپرتی را نیز انجام دهید.

چند نکته :

اکسسور get هنگام بازگشت یا خواندن مقدار پروپرتی اجرا می‌شود و اکسسور set زمان انتساب یک مقدار جدید به پروپرتی فراخوانی می‌شود. جالب آنکه در صورت لزوم این دو اکسسور می‌توانند دارای سطوح دسترسی متفاوتی باشند.

داخل اکسسور set کلمه کلیدی value مقدار منتسب شده را در اختیار قرار می‌دهد تا در صورت لزوم بتوان بر روی آن پردازش لازم را انجام داد.

یک پروپرتی می‌تواند فاقد اکسسور set باشد که در این صورت یک پروپرتی فقط خواندنی ایجاد می‌گردد. همچنین می‌تواند فقط شامل اکسسور set باشد که در این صورت فقط امکان انتساب مقادر به آن وجود دارد و امکان دریافت یا خواندن مقدار آن میسر نیست. چنین پروپرتی ای فقط نوشتنی خواهد بود.

در بدنه اکسسور set الزامی به انتساب مقدار منتسب توسط کد مشتری نیست. در صورت صلاحدید می‌توانید به جای آن هر مقدار دیگری را در نظر بگیرید یا عملیات مورد نظر خود را انجام دهید.

در بدنه اکسسور get هم هر مقداری را می‌توانید بازگشت دهید. یعنی الزامی وجود ندارد حتماً مقدار فیلد خصوصی متناظر با پروپرتی را بازگشت دهید. حتی الزامی به تعریف فیلد خصوصی برای هر پروپرتی ندارید. به طور مثال ممکن است مقدار بازگشتی اکسسور get حاصل محاسبه و ... باشد.

اکنون مثال دیگری را در نظر بگیرید. فرض کنید در یک پروژه فروشگاهی در حال تهیه کلاسی برای مدیریت محصولات هستید. قصد داریم یک پروپرتی ایجاد کنیم تا نام محصول را نگهداری کند و در حال حاضر هیچ محدودیتی برای نام یک محصول در نظر نداریم. کد زیر را ببینید.

```
public class Product
{
    private string _name;
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}
```

همانطور که می‌بینید در بدنه اکسسورهای پروپرتی Name هیچ عملیات نظارتی ای در نظر گرفته نشده است. آیا بهتر نبود بیهوده پروپرتی تعریف نکنیم و خیلی ساده از یک فیلد عمومی که همین کار را انجام می‌دهد استفاده کنیم؟ خیر. بهتر نبود. مهمترین دلیلی که فعلاً کافی است تا شما را قانع کند تعریف پروپرتی روش پسندیده‌تری از فیلد عمومی است را بررسی می‌کنیم.
فرض کنید پس از مدتی متوجه شدید اگر نام بسیار طولانی ای برای محصول درج شود ظاهر برنامه شما دچار مشکل می‌شود. پس باید بر روی این مورد نظارت داشته باشید. دیدیم که برای رسیدن به این هدف باید فیلد عمومی را فراموش و به جای آن پروپرتی تعریف کنیم. اما مسئله اینست که تبدیل یک فیلد عمومی به پروپرتی میتواند سبب بروز ناسازگاری‌هایی در بخش‌های دیگر برنامه که از این کلاس و آن فیلد استفاده می‌کنند شود. پس بهتر آن است که از ابتدا پروپرتی تعریف کنیم هر چند نیازی به عملیات نظارتی خاصی نداریم. در این حالت اگر نیاز به پردازش بیشتر پیدا شد به راحتی می‌توانیم کد مورد نظر را در اکسسورهای موجود اضافه کنیم بدون آنکه نیازی به تغییر بخش‌های دیگر باشد.
و یک خبر خوب! از سی شارپ ۳ به بعد ویژگی جدیدی در اختیار ما قرار گرفته است که می‌توان پروپرتی‌هایی مانند مثال بالا را که نیازی به عملیات نظارتی ندارند، ساده‌تر و خواناتر تعریف نمود. این ویژگی جدید پروپرتی اتوماتیک یا Auto-Implemented Property نام دارد. مانند نمونه زیر.

```
public class Product
{
    public string Name { get; set; }
}
```

این کد مشابه کد پیشین است با این تفاوت که خود کامپایلر یک متغیر خصوصی و بی نام را ایجاد می‌نماید که فقط داخل اکسسورهای پروپرتی قابل دسترسی است.
البته استفاده از پروپرتی برتری دیگری هم دارد. و آن کنترل سطح دسترسی اکسسورها است. مثال زیر را ببینید.

```
public class Student
{
    public DateTime Birthdate { get; set; }

    public double Age { get; private set; }
}
```

کلاس دانشجو یک پروپرتی به نام تاریخ تولد دارد که قابل خواندن و نوشتن توسط کد مشتری (کد استفاده کننده از کلاس یا اشیاء آن) است. و یک پروپرتی دیگر به نام سن دارد که توسط کد مشتری تنها قابل خواندن است. و تنها توسط سایر اعضای داخل همین کلاس قابل نوشتن است. چون اکسسور set آن به صورت خصوصی تعریف شده است. به این ترتیب بخش دیگری از کلاس سن دانشجو را بر اساس تاریخ تولد او محسابه می‌کند و در پروپرتی Age قرار می‌دهد و کد مشتری می‌تواند آن‌را مورد استفاده قرار دهد اما حق دستکاری آن‌را ندارد. به همین ترتیب در صورت نیاز اکسسور get را می‌توان خصوصی کرد تا پروپرتی از دید کد مشتری فقط نوشتنی باشد. اما حتماً می‌توانید حدس بزنید که نمی‌توان هر دو اکسسور را خصوصی کرد. چرا؟
تذکر: در هنگام تعریف یک **فیلد** می‌توان از کلمه کلیدی readonly استفاده کرد تا یک **فیلد فقط خواندنی** ایجاد گردد. اما در اینصورت فیلد تعریف شده حتی داخل کلاس هم فقط خواندنی است و فقط در هنگام تعریف یا در متد سازنده کلاس امکان مقدار دهی به آن وجود دارد. در بخش‌های بعدی مفهوم سازنده کلاس مورد بررسی خواهد گرفت.

نظرات خوانندگان

نویسنده: Kingtak
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۸:۱۶

مثل همیشه عالی بود....
واقعا با آموزش‌های شما حال میکنم.تا حالاچندین مقاله در مورد پروپرتی‌ها خونده بودم ولی بطور کامل متوجه کاربردش و فوایدش نشده بودم.
منتظر آموزش‌های بعدی هستیم

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۱۹:۱۲

تشکر. شما لطف دارید. امیدوارم مطالب بعدی هم براتون مفید باشه.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۱:۲۲

سلام،
خیلی برام لذت بخشه دانسته هام رو به شیوه ای زیباتر مرور کنم و در ضمن با نکاتی ظریفتر آشنا بشم.
چند تا پیشنهاد:
-1 خیلی خوب میشه اگه تا حد امکان معادل انگلیسی کلمات تخصصی مربوطه رو هم بزارید، مثلا از اکسسور استفاده میکنید خوبه ولی داخل پرانتز هم اشاره ای به معادل انگلیسی Accessor بکنید تا بدونیم در سینتکسش چه جوریه، و تا حد امکان هم معادل تخصصی واژه مربوطه ذکر بشه مثلا فیلدهای خصوصی داخل کلاس معمولا با عنوان backing field یا فیدلهای پشتی خطاب میشه که اگه به این موارد هم اشاره ای بشه خوبه.
-2 جاهایی که به استانداردها و کلا هر چیزی در حوزه مهندسی نرم افزار اشاره می‌کنید لطفا تا حد امکان اشاره ای هم به آن بکنید مثلا فرمودید اصول نامگذاری استاندارد فیلد خصوصی، اینجا به نظر من بهتره که اشاره ای هم بکنید مصلا در استاندارد مایکروسافت فیلدهای خصوصی از قاعده نامگذاری Camel Case استفاده میکنند و یا مثلا در متدها و کلاس‌های از روش Pascal Case استفاده میشه.
-3 در مورد اینکه چه مواقعی باید از فلان موضوع، قاعده و مبحث و ... استفاده بشه هم در صورت امکان و تا حد توان اشاره بکنید مثلا فرمودید فیلدهای فقط خواندنی Readonly، مناسبه که اشاره ای هم بکنید معمولا چه زمانی و در چه شرایطی بهتره از این نوع فیلد استفاده کنیم و چرا؟ (البته هنوز که توضیحی ندادید ولی پیشتر اشاره کردم تا انشاء الله در مقاله بعدی تان در صورت صلاحدید لحاظ بفرمایید).

پیشنهاداتی که شد صرفا برای هر چه بهتر شدن مقالات بعدی شماست و اینکه باعث بشه، خواننده وقتی با موضوعاتی در این زمینه در متون تخصصی برخورد داشت احساس بیگانگی نکنه و سریعتر در جریان کار قرار بگیره.
سلسله مباحثی که ارائه کرده اید تا کنون زیبا بود و خواندنی و بنده هم مخاطب همیشگی این سلسله مباحث و البته امیدوارم به کمتر شدن فاصله بین پست‌های ارسالی :).
متشکرم/.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۲:۵۰

ضمن تشکر از پیگیری و پیشنهادهای حضرتعالی و پوزش به جهت طولانی شدن فاصله زمانی ارائه مطالب در مورد پیشنهادهای ارزشمندی که فرمودید باید چند نکته را عرض کنم.
تا حد زیادی معمولاً سعی کردم این موارد محقق بشه. مثلا در مورد همان اکسسور و بیشتر مفاهیم و اصطلاحات مهم، معادل انگلیسی آورده شده است. اصولاً ترجمه برخی مفاهیم را مناسب نمی‌دانم و از طرفی آوردن تعداد زیادی واژه انگلیسی در بین واژگان فارسی سبب کاهش زیبایی متن می‌گردد. بنابراین معمولاً کلمات مهم را یک یا چند بار به صورت انگلیسی بیان می‌کنم و

سپس با حروف فارسی می‌نویسم مانند اکسسور تا به صورت روان‌تری در متن قابل خواندن باشد.
همچنین در امر آموزش ابتدا سعی می‌کنم یک دید کلی و از بالا به دانشجو یا خواننده منتقل کنم. در این مرحله تنها جزییات مهم که برای درک موضوع و شروع کار عملی مانند انجام یک پروژه کاربردی لازم است بیان می‌شود. چراکه اگر از ابتدا ذهن را با تعداد زیادی جزییات درگیر کنیم ممکن است در موقع خواندن هر بخش خواننده مفاهیم را درک کند اما پس از پایان مطالب نمی‌داند از کجا باید شروع کند و قدرت استفاده از آموخته‌ها را ندارد. به همین جهت سعی می‌شود بر روی مفاهیم غیر کلیدی کمتر در مراحل اولیه بحث شود.
از طرفی سعی می‌کنم مطالب دارای حجم مناسب و مفاهیم پیوسته ای باشند تا قابل درک بوده و خسته کننده نباشند. مثلاً از آنجاییکه در بخش‌های پیشین مقاله‌ای که به زحمت یکی از دوستان در سایت قرار گرفته بود برای نامگذاری معرفی شد، از تکرار قوانین یاد شده در این مطالب به جهت جلوگیری از طولانی‌تر شدن خودداری کردم.
با توجه به کارگاه‌های عملی ای که برای تثبیت مطالب در نظر گرفته خواهد شد، تا حد زیادی روش‌های بهینه برای پیاده سازی مفاهیم گوناگون معرفی خواهد شد.

نویسنده: عبداللهی
تاریخ: ۱۳۹۲/۰۲/۰۴ ۲۳:۲۶

سلام.
واقعا عالی بود.مرسی.فقط یه سوالی داشتم. در کلاس student سطح دسترسی بصورت پیش فرض private خواهد بود؟چون اگر درست یادم مونده باشه موقع تعریف متغیر سطح دسترسی بصورت پیشفرض private بود.درسته؟
بازم ممنون.3 تا مقالتون رو خودنم.واقعا مفید بود.باتشکر

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۵ ۱۱:۲۲

سلام و تشکر.
سطح دسترسی پیش فرض برای اعضای کلاس (حتی کلاس داخلی‌تر) به صورت پیش فرض private است. اما اکسسورها از سطح دسترسی پروپرتی تبعیت می‌کنند مگر آنکه صراحتاً سطح دسترسی آن‌ها تعیین گردد. بدیهی است در صورتی تعیین صریح سطح دسترسی برای اکسسورها پذیرفته است که نسبت به سطح دسترسی پروپرتی محدودتر باشد. یعنی نمی‌توانید مثلاً پروپرتی ای را private و اکسسور آن را public تعیین کنید.

نویسنده: علی صداقت
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۰:۲۸

با سپاس فراوان از سلسه مطالب مفید شما. فکر میکنم در کلاس Rectangle و پروپرتی Height در قسمت set، مقدار value باید در شرط مورد ارزیابی قرار گیرد.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۲۴

```
set { if (value > 0) { _width = value; } }
```

این نوع طراحی API به نظر من ایراد داره. از این جهت که مصرف کننده نمی‌دونه چرا مقداری که وارد کرده تاثیری نداشته. بهتره در این نوع موارد یک استثناء صادر شود.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۳۱

با تشکر. اصلاح شد.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۰۷ ۱۱:۴۰

تشکر فراوان از نظر حضرتعالی.
بله صحیح می‌فرمایید. اما کار با این کلاس تمام نشده است و صرفاً مثالی ساده برای بیان مفاهیم پایه ای مورد نظر در مقاله است. در چنین مثالی نباید ذهن خواننده را درگیر مسائلی نمود که مورد هدف بحث نیست. ضمناً هر مقاله دارای یک جامعه هدف است. اگرچه می‌تواند برای افراد دیگر هم مفید واقع شود اما باید دانسته‌های جامعه هدف خود را مد نظر داشته باشد. برای خواننده ای که در حال آشنایی با مفهوم پروپرتی است، صدور یک استثنا و مفاهیم مربوطه نیاز به بحثی جدا دارد.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۱۰ ۲۲:۱۷

البته امیدوارم هستم این موارد را در مقالات بعدی خود شرح دهید.

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۲/۱۷ ۳:۵۲

خیلی ممنون بابت مطلبتون
یه سوالی که مدت‌ها تو ذهنم مونده اینه که فرق این کار (استفاده از property برای مقدار دهی فیلدهای private) با مدل مشابهی که در کتاب‌های جاوا دیده می‌شه (استفاده از دو متد به طور مثال setWidth و getWidth برای مقدار دهی فیلدهای private در اینجا width) در چیه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۷ ۸:۹

این روش در سی‌شارپ منسوخ شده در نظر گرفته میشه و یکی از مواردی هست که حین تبدیل کدهای جاوا به سی شارپ تبدیل به یک خاصیت خواهد شد بجای دو متد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۸:۱۹

با اجازه آقای آرمان فرقانی
دوست گلم دقیقا وقتی برنامه کامپایل میشه خصیصه‌ها به متدهایی مانند جاوا (که با set_ یا get_ شروع شده و به نام خصیصه ختم می‌شود) تبدیل میشوند
مثلا شما توی کلاست اگر خصیصه ای با نام Width داشته باشید و یک متد مانند get_Width تعریف کنی زمان کامپایل خطا زیر دریافت می‌کنی، به منزله اینکه این متد وجود داره:

```
Error 1 Type 'Rectangle' already reserves a member called 'get_Width' with the same parameter types
E:\Test\Rectangle.cs5940
```

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۲:۴۱

تشکر از شما و توضیحات ارزشمند دوستان گرامی.
پروپرتی و پروپرتی اتوماتیک امکانی است که در زبان سی شارپ و ... قرار داده شده است. پروپرتی‌ها نیز در حقیقت متدهای مشابهی دارند که همان اکسسورها هستند. تفاوت میزان بیشتر کپسوله سازی و مخفی کردن منطق پیاده سازی، و مهم‌تر سازگاری بیشتر با مفهوم ویژگی است. که البته در هنگام استفاده از پروپرتی سهولت بیشتری را نیز فراهم می‌کند.
همان که دوست عزیزم اشاره فرمودند به دلیل عدم سازگاری ذات زبان‌های مبتنی بر دات فریمورک از اکسسور، به صورت

داخلی به متد تبدیل خواهند شد.
همچنین در مورد جاوا هم پروژه هایی وجود دارند که سعی کرده اند این امکان را به کمک یک سری Annotaion به آن بیافزایند.
در مورد سی شارپ استفاده از پروپرتی روش توصیه شده است.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۳:۲۱

دات نت مشکلی با متدهای get و set دار نداره. فقط چون خیلی verbose بوده، جمع و جور شده به auto implemented properties برای زیبایی کار و سهولت تایپ. نکته‌ای هم که آقای فتح الهی عنوان کردند، در مورد ترکیب متد و خاصیت هم نام با هم بود، در یکجا البته اون هم حالتی که بعد از متد get شروع شده با حرف کوچک، یک _ باشد مثلا و نه حالت دیگری.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۴:۲۹

متوجه نکته مورد نظر شما نشدم. بیان شد در زبان سی شارپ و ... ساختار کپسوله‌تر پروپرتی در مقایسه با متدهای صریح تنظیم و بازیابی مقدار فیلدها در جاوا معرفی شده اند ولی پیاده سازی داخلی آن به همان صورت متد است. نکته دوست گرامی آقای فتح الهی هم گمان می‌کنم بیشتر به منظور اشاره به چگونگی پیاده سازی داخلی است و نه اینکه مراقب باشید تداخل نام پیش نیاید.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۲/۱۷ ۱۷:۱۰

بله من در پاسخ به سوال دوستمون گفتم **پیاده سازی داخلی** خصیصه به این شکل هست که خصیصه به شکل متد پیاده سازی است، و جهت آزمایش گفتم یک متد ... ایجاد کنید.

عنوان: **مفاهیم برنامه نویسی ـ آشنایی با سازنده‌ها**
نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۳:۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, Programming, آموزش برنامه نویسی

در مطلب پیشین برای نگهداری حالت شیء یا همان ویژگی‌های آن Propertyها را در کلاس معرفی کردیم و پس از ایجاد شیء مقدار مناسبی را به پروپرتی‌ها اختصاص دادیم.
اگرچه ایجاد شیء و مقداردهی به ویژگی‌های آن ما را به هدفمان می‌رساند، اما بهترین روش نیست چرا که ممکن است مقداردهی به یک ویژگی فراموش شده و سبب شود شیء در وضعیت نادرستی قرار گیرد. این مشکل با استفاده از سازنده‌ها (Constructors) حل می‌شود.
سازنده (Constructor) عضو ویژه ای از کلاس است بسیار شبیه به یک متد که در هنگام وهله سازی (ایجاد یک شیء از کلاس) به صورت خودکار فراخوانی و اجرا می‌شود. وظیفه سازنده مقداردهی به ویژگی‌های عمومی و خصوصی شیء تازه ساخته شده و به طور کلی انجام هر کاری است که برنامه‌نویس در نظر دارد پیش از استفاده از شیء به انجام برساند.
مثالی که در این بخش بررسی می‌کنیم کلاس مثلث است. برای پیاده سازی این کلاس سه ویژگی در نظر گرفته ایم. قاعده، ارتفاع و مساحت. بله مساحت را این بار به جای متد به صورت یک پروپرتی پیاده سازی می‌کنیم. اگرچه در آینده بیشتر راجع به چگونگی انتخاب برای پیاده سازی یک عضو کلاس به صورت پروپرتی یا متد بحث خواهیم کرد اما به عنوان یک قانون کلی در نظر داشته باشید عضوی که به صورت منطقی به عنوان داده مطرح است را به صورت پروپرتی پیاده سازی کنید. مانند نام دانشجو. از طرفی اعضایی که دلالت بر انجام عملی دارند را به صورت متد پیاده سازی می‌کنیم. مانند متد تبدیل به نوع داده دیگر. (مثلاً Object.ToString())

```
public class Triangle
{
    private int _height;
    private int _baseLength;

    public int Height
    {
        get { return _height; }

        set
        {
            if (value < 1 || value > 100)
            {
                // تولید خطا
            }

            _height = value;
        }
    }

    public int BaseLength
    {
        get { return _baseLength; }

        set
        {
            if (value < 1 || value > 100)
            {
                // تولید خطا
            }

            _baseLength = value;
        }
    }

    public double Area
    {
        get { return _height * _baseLength * 0.5; }
    }
}
```

چون در بخشی از یک پروژه نیاز پیدا کردیم با یک سری مثلث کار کنیم، کلاس بالا را طراحی کرده ایم. به نکات زیر توجه نمایید.
• در اکسسور set دو ویژگی قاعده و ارتفاع، محدوده مجاز مقادیر قابل انتساب را بررسی نموده ایم. در صورتی که مقداری خارج از محدوده یاد شده برای این ویژگی‌ها تنظیم شود خطایی را ایجاد خواهیم کرد. شاید برای برنامه نویسانی که تجربه کمتری دارند

زیاد روش مناسبی به نظر نرسد. اما این یک روش قابل توصیه است. مواجه شدن کد مشتری (کد استفاده کننده از کلاس) با یک خطای مهلک که علت رخ دادن خطا را نیز می‌توان به همراه آن ارائه کرد بسیار بهتر از بروز خطاهای منطقی در برنامه است. چون رفع خطاهای منطقی بسیار دشوارتر است. در مطالب آینده راجع به تولید خطا و موارد مرتبط با آن بیشتر صحبت می‌کنیم.
• در مورد ویژگی مساحت، اکسسور set را پیاده سازی نکرده ایم تا این ویژگی را به صورت فقط خواندنی ایجاد کنیم.

وقتی شیء ای از یک کلاس ایجاد می‌شود، بلافاصله سازنده آن فراخوانی می‌گردد. سازنده‌ها هم نام کلاسی هستند که در آن تعریف می‌شوند و معمولاً اعضای داده ای شیء جدید را مقداردهی می‌کند. همانطور که می‌دانید وهله سازی از یک کلاس با عملگر new انجام می‌شود. سازنده کلاس بلافاصله پس از آنکه حافظه برای شیء در حال تولید اختصاص داده شد، توسط عملگر new فراخوانی می‌شود.

سازنده پیش فرض سازنده‌ها مانند متدهای دیگر می‌توانند پارامتر دریافت کنند. سازنده ای که هیچ پارامتری دریافت نمی‌کند سازنده پیش فرض (Default constructor) نامیده می‌شود. سازنده پیش فرض زمانی اجرا می‌شود که با استفاده از عملگر new شیء ای ایجاد می‌کنید اما هیچ آرگومانی را برای این عملگر در نظر نگرفته اید.
اگر برای کلاسی که طراحی می‌کنید سازنده ای تعریف نکرده باشید کامپایلر سی شارپ یک سازنده پیش فرض (بدون پارامتر) خواهد ساخت. این سازنده هنگام ایجاد اشیاء فراخوانی شده و مقدار پیش فرض متغیرها و پروپرتی‌ها را با توجه به نوع آن‌ها تنظیم می‌نماید. مثلاً مقدار صفر برای متغیری از نوع int یا false برای نوع bool و null برای انواع ارجاعی که در آینده در این مورد بیشتر خواهید آموخت.
اگر مقادیر پیش فرض برای متغیرها و پروپرتی‌ها مناسب نباشد، مانند مثال ما، سازنده پیش فرض ساخته شده توسط کامپایلر همواره شیء ای می‌سازد که وضعیت صحیحی ندارد و نمی‌تواند وظیفه خود را انجام دهد. در این گونه موارد باید این سازنده را جایگزین نمود.

جایگزینی سازنده پیش فرض ساخته شده توسط کامپایلر افزودن یک سازنده صریح به کلاس بسیار شبیه به تعریف یک متد در کلاس است. با این تفاوت که:
سازنده هم نام کلاس است.
برای سازنده نوع خروجی در نظر گرفته نمی‌شود.

در مثال ما محدوده مجاز برای قاعده و ارتفاع مثلث بین ۱ تا ۱۰۰ است در حالی که سازنده پیش فرض مقدار صفر را برای آنها تنظیم خواهد نمود. پس برای اینکه مطمئن شویم اشیاء مثلث ساخته شده از این کلاس در همان بدو تولید دارای قاعده و ارتفاع معتبری هستند سازنده زیر را به صورت صریح در کلاس تعریف می‌کنیم تا جایگزین سازنده پیش فرضی شود که کامپایلر خواهد ساخت و به جای آن فراخوانی گردد.

```
public Triangle()
{
    _height = _baseLength = 1;
}
```

در این سازنده مقدار ۱ را برای متغیر خصوصی پشت (backing field یا backing store) هر یک از دو ویژگی قاعده و ارتفاع تنظیم نموده ایم.

اجرای سازنده همانطور که گفته شد سازنده اضافه شده به کلاس جایگزین سازنده پیش فرض کامپایلر شده و در هنگام ایجاد یک شیء جدید از کلاس مثلث توسط عملگر new اجرا می‌شود. برای بررسی اجرا شدن سازنده به سادگی می‌توان کدی مشابه مثال زیر را نوشت.

```
Triangle triangle = new Triangle();

Console.WriteLine(triangle.Height);
Console.WriteLine(triangle.BaseLength);
Console.WriteLine(triangle.Area);
```

کد بالا مقدار ۱ را برای قاعده و ارتفاع و مقدار ۰.۵ را برای مساحت چاپ می‌نماید. بنابراین مشخص است که سازنده اجرا شده و مقادیر مناسب را برای شیء تنظیم نموده به طوری که شیء از بدو تولید در وضعیت مناسبی است.

سازنده‌های پارامتر دار در مثال قبل یک سازنده بدون پارامتر را به کلاس اضافه کردیم. این سازنده تنها مقادیر پیش فرض مناسبی را تنظیم می‌کند. بدیهی است پس از ایجاد شیء در صورت نیاز می‌توان مقادیر مورد نظر دیگر را برای قاعده و ارتفاع تنظیم نمود. اما برای اینکه سازنده بهتر بتواند فرآیند وهله سازی را کنترل نماید می‌توان پارامترهایی را به آن افزود. افزودن پارامتر به سازنده مانند افزودن پارامتر به متدهای دیگر صورت می‌گیرد. در مثال زیر سازنده دیگری تعریف می‌کنیم که دارای دو پارامتر است. یکی قاعده و دیگری ارتفاع. به این ترتیب در حین فرآیند وهله سازی می‌توان مقادیر مورد نظر را منتسب نمود.

```
public Triangle(int height, int baseLength)
{
    Height = height;
    BaseLength = baseLength;
}
```

با توجه به اینکه مقادیر ارسالی به این سازنده توسط کد مشتری در نظر گرفته می‌شود و ممکن است در محدوده مجاز نباشد، به جای انتساب مستقیم این مقادیر به فیلد خصوصی پشت ویژگی قاعده و ارتفاع یعنی _baseLength و _height آنها را به پروپرتی‌ها منتسب کردیم تا قانون اعتبارسنجی موجود در اکسسور set پروپرتی‌ها از انتساب مقادیر غیر مجاز جلوگیری کند.
سازنده اخیر را می‌توان به صورت زیر با استفاده از عملگر new و فراهم کردن آرگومان‌های مورد نظر مورد استفاده قرار داد.

```
Triangle triangle = new Triangle(5, 8);

Console.WriteLine(triangle.Height);
Console.WriteLine(triangle.BaseLength);
Console.WriteLine(triangle.Area);
```

مقادیر چاپ شده برابر ۵ برای ارتفاع، ۸ برای قاعده و ۲۰ برای مساحت خواهد بود که نشان از اجرای صحیح سازنده دارد.
در مطالب بالا چندین بار از سازنده ها صحبت کردیم و گفتیم سازنده دیگری به کلاس اضافه می‌کنیم. این دو نکته را به خاطر داشته باشید:
یک کلاس می‌تواند دارای چندین سازنده باشد که بر اساس آرگومان‌های فراهم شده هنگام وهله سازی، سازنده مورد نظر انتخاب و اجرا می‌شود.
الزامی به تعریف یک سازنده پیش فرض (به معنای بدون پارامتر) نیست. یعنی یک کلاس می‌تواند هیچ سازنده بی پارامتری نداشته باشد.

در بخش‌های بعدی مطالب بیشتری در مورد سازنده‌ها و سایر اعضای کلاس خواهید آموخت.

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۴ ۸:۴۵

ممنون از شما. بنابراین مقدار دهی خواص در سازنده کلاس به معنای نسبت دادن مقدار پیش فرض به آن‌ها است. چون سازنده کلاس پیش از هر کد دیگری در کلاس فراخوانی می‌شود.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۰:۳۱

تشکر. همانطور که گفته شد بلافاصله پس از تخصیص حافظه به شیء سازنده اجرا می‌شود. در صورت استفاده از سازنده پارامتر دار کد مشتری از ابتدا شیء را با مقادیر عملیاتی خود ایجاد می‌کند.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۲:۵۹

سلام،
آقای فرقانی بسیار عالی بود، واقعا لذت بردم،
فقط خواهشی از شما دارم این هستش که : دقت و حساسیت مبحث شی گرایی رو احیانا فدای چیزهای دیگر نکنید!، منظورم این هستش که تا حد توان اصول مهندسی نرم افزار رو به صورت کامل رعایت بفرمایید و نگران کاربر مبتدی نباشید، کاربر مبتدی ، ناخودآگاه خودش مطالب غیر قابل فهم رو فیلتر میکنه و در بدترین حالت، در آن مورد، از شما سوال خواهند کرد و شما هم آنها را به موضوع مناسب ارجاع خواهید داد هر چند اگر موضوع ارائه شده بعدا قراره باز بشه، خیلی راحت می‌تونید در متن نوید توضیحات بیشتر رو به خواننده بدهید و در غیر این صورت ارجاع به توضیحات مناسب.
به هر حال باز هم تاکید می‌کنم و سفارش میکنم که دقت بیان و رعایت اصول رو فدای هیچ چیزی نکنید.
ممنونم.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۴ ۱۳:۲۳

تشکر از نظر شما.
اگر مورد خاصی مد نظر دارید بفرمایید تا توضیح بدم یا در صورت لزوم در متن اصلاح کنم. در نظر داشته باشید این سری مطالب با مطالب دیگر موجود در سایت متفاوت است و هدف خاصی را دنبال می‌کند که در بخش اول توضیح داده شد. بنابراین نمی‌توان در این سری مطالب نگران کاربر مبتدی‌تر نبود چراکه جامعه هدف این بحث‌ها هستند. به دلیل همین جامعه هدف مورد نظر، نوشتن این گونه مطالب دشوارتر از بیان مفاهیم پیچیده‌تر برای کاربران حرفه‌ای‌تر است. و همین امر به علاوه وقت محدود بنده سبب تأخیر در ارسال مطالب است. ضمناً صرف بیان انبوهی از اطلاعات تأثیر لازم را در خواننده نمی‌گذارد. در بسیاری مواقع بیان برخی مفاهیم مهندسی نرم افزار یا ویژگی‌های جدید زبان (مانند var) به صورت مقایسه ای با روش پیشین سبب تثبیت بهتر مطالب در ذهن خواننده می‌شود. اما در شیوه کد نویسی تا حد ممکن سعی شده اصول رعایت شود و بیهوده خواننده با روش ناصحیح آشنا نشود. اگر نکته خاصی یافتید بفرمایید اصلاح کنیم.

نویسنده: عبداللهی
تاریخ: ۱۳۹۲/۰۲/۱۶ ۰:۵۱

باسلام.تشکر ازینکه مطالب رو ساده گویا و دقیق بیان میکنید.بسیار عالی بود.سپاسگذارم.
یک کلاس همیشه 1 سازنده پیش فرض بدون پارامتر دارد که هنگام وهله سازی فراخوانی شده و اجرا میشود و در صورت نیاز میتوان 1 سازنده بر اساس نیازهای پروژه تعریف کرد که هنگام وهله سازی از یک کلاس سازنده تعریف شده ما بر سازنده پیش فرض اولویت دارد.درسته؟
بازم متشکرم.مرسی

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۱۶ ۱۱:۱۲

محدودیتی برای تعداد متدهای سازنده وجود نداره (مبحث overloading است که نیاز به بحثی جداگانه دارند). در زمان وهله سازی کلاس میشه مشخص کرد کدام متد مورد استفاده قرار بگیره. این متد بر سایرین مقدم خواهد بود. همچنین سازنده استاتیک هم قابل تعریف است که نکته خاص خودش رو داره.

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۱۶ ۱۱:۱۴

سلام و تشکر. به نکات زیر در متن توجه فرمایید.
۱. سازنده پیش فرض منظور همان سازنده بی پارامتر است خواه توسط کامپایلر ایجاد شده باشد خواه توسط برنامه نویس به صورت صریح در کلاس تعریف شده باشد.
۲. اگر توسط برنامه نویس هیچ سازنده ای تعریف نگردد، کامپایلر یک سازنده بدون پارامتر خواهد ساخت که وظیفه تنظیم مقادیر پیش فرض اعضای داده ای کلاس را برعهده بگیرد.
۳. اگر برنامه نویس سازنده ای تعریف کند خواه با پارامتر یا بی پارامتر کامپایلر سازنده ای نخواهد ساخت. پس اگر مثلاً برنامه نویس تنها یک سازنده با پارامتر تعریف کند، کلاس فاقد سازنده بی پارامتر یا پیش فرض خواهد بود.
۴. در یک کلاس می‌توان چندین سازنده تعریف نمود.
موفق باشید.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۴:۴۴

سلام/با تشکر از مطالب پایه ای و مفیدتون
کاربرد دو قطعه کد زیر در چه زمانی بهتر است؟

1.گرفتن مقدار مساحت به صورت یک Property

```
public double Area
{
       get { return _height * _baseLength * 0.5; }
}
```

2.محاسبه مساحت با استفاده از یک Method

```
public double Area()
{
     return _height * _baseLength * 0.5;
}
```

نویسنده: احمد
تاریخ: ۱۳۹۲/۰۲/۲۸ ۱۸:۳۳

Properties vs Methods

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۹ ۸:۳۸

احمد عزیز ممنون بابت لینک تون.
سوالم اینه که چرا آقای فرقانی تو این مثال برای Area هم یک Property تعریف کردن (چون صرفا فقط یک مثال | براساس تعریف Method و Property ، متد بهتر نبود؟)

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۲۹ ۸:۴۸

خلاصه لینک احمد: اگر محاسبات پیچیده و طولانی است، یا تاثیرات جانبی روی عملکرد سایر قسمت‌های کلاس دارند، بهتره از متد استفاده بشه. اگر کوتاه، سریع و یکی دو سطری است و ترتیب فراخوانی آن اهمیتی ندارد، فرقی نمی‌کنه و بهتره که خاصیت باشه و اگر این شرایط حاصل شد، عموم کاربران تازه کار استفاده از خواص را نسبت به متدها ساده‌تر می‌یابند و به نظر آن‌ها Syntax تمیزتری دارد (هدف این سری مقدماتی).

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۲/۲۹ ۹:۱۳

باز هم به نظر من استفاده از method صحیح‌تر بود
درسته که در اینجا محاسبات پیچیده (computationally complex) نداریم اما چون یک action تلقی می‌شه پس ... .
(البته شاید این حساسیت بیجاست و به قول دوست عزیز 'محسن خان' چون این سری مقدماتی است به این صورت نوشته شده است)

نویسنده: آرمان فرقانی
تاریخ: ۱۳۹۲/۰۲/۳۰ ۱۹:۵

ضمن تشکر از دوستانی که در بحث شرکت کردند و پوزش به دلیل اینکه چند هفته ای در سفر هستم و تهیه مطالب با تأخیر انجام خواهد شد.
برای پاسخ به پرسش دوست گرامی آقای نجف زاده ابتدا بخشی از این مطلب را یادآوری می‌کنم.
"... مساحت را این بار به جای متد به صورت یک پروپرتی پیاده سازی می‌کنیم. اگرچه در آینده بیشتر راجع به چگونگی انتخاب برای پیاده سازی یک عضو کلاس به صورت پروپرتی یا متد بحث خواهیم کرد اما به عنوان یک قانون کلی در نظر داشته باشید عضوی که به صورت منطقی به عنوان داده مطرح است را به صورت پروپرتی پیاده سازی کنید. مانند نام دانشجو. از طرفی اعضایی که دلالت بر انجام عملی دارند را به صورت متد پیاده سازی می‌کنیم. مانند متد تبدیل به نوع داده دیگر. (مثلاً (Object.ToString)
..."

بنابراین به نکات زیر توجه فرمایید.
۱. در این مطالب سعی شده است امکان پیاده سازی یک مفهوم به دو صورت متد و پروپرتی نشان داده شود تا در ذهن خواننده زمینه ای برای بررسی بیشتر مفهوم متد و پروپرتی و تفاوت آن‌ها فراهم گردد. این زمینه برای کنجکاوی بیشتر معمولاً با انجام یک جستجوی ساده سبب توسعه و تثبیت علم شخص می‌گردد.
۲. در متن بالاً به صورت کلی اشاره شده است هر یک از دو مفهوم متد و پروپرتی در کجا باید استفاده شوند و نیز خاطرنشان شده است در مطالب بعدی در مورد این موضوع بیشتر صحبت خواهد شد.
۳. نکته مهم در طراحی کلاس، پایگاه داده و ... خرد جهان واقع یا محیط عملیاتی مورد نظر طراح است. به عبارت دیگر گسی نمی‌تواند به یک طراح بگوید به طور مثال مساحت باید متد باشد یا باید پروپرتی باشد. طراح با توجه به مفهوم و کارکردی که برای هر مورد در ذهن دارد بر اساس اصول و قواعد، متد یا پروپرتی را بر می‌گزیند. مثلاً در خرد جهان واقع موجود در ذهن یک طراح مساحت به عنوان یک عمل یا اکشنی که شیء انجام می‌دهد است و بنابراین متد را انتخاب می‌کند. طراح دیگری در خرد جهان واقع دیگری در حال طراحی است و مثلاً متراژ یک شیء خانه را به عنوان یک ویژگی ذاتی و داده ای می‌نگرد و گمان می‌کند خانه نیازی به انجام عملی برای بدست آوردن مساحت خود ندارد بلکه یکی از ویژگی‌های خود را می‌تواند به اطلاع استفاده کننده برساند. پس شما به طراح دیگر نگویید اکشن تلقی میشه پس باید متد استفاده شود. اگر خود در پروژه ای چیزی را اکشن تلقی نمودید بله باید متد به کار ببرید. تلقی‌ها بر اساس خرد جهان واقع معنا دارند.
۴. پروپرتی و متد از نظر شیوه استفاده و ... با هم تفاوت دارند. اما یک تفاوت مهم بین آنها بیان نوع مفاهیم موجود در ذهن طراح به کد مشتری است. فراموش نکنید خود پروپرتی دارای اکسسور است که چیزی مانند متد است. در خیلی از موارد صحیح‌تر بودن پیاده سازی با متد یا با پروپرتی معنا ندارد. انتخاب ما بین متد یا پروپرتی بر اساس نحوه استفاده مطلوب در کد مشتری و نیز اطلاع به مشتری که مثلاً فلان مفهوم از دید ما یک اکشن است و فلان چیز داده صورت می‌گیرد.

نویسنده: محسن نجف زاده
تاریخ: ۱۳۹۲/۰۳/۲۹ ۸:۹

با تشکر از آرمان فرقانی عزیز / مطلب و بحث مفید بود.

عنوان: **آموزش F#**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۳/۲۴ ۲۳:۴۵
آدرس: www.dotnettips.info
گروه‌ها: F#, Programming

در نظر سنجی که قبلا توسط دوستان درباره میزان آشنایی و استفاده از زبان‌های مختلف برنامه نویسی در تولید پروژه‌های نرم افزاری انجام شده بود (^) تعداد رای زبان F# سه رای بود(یعنی کمتر از یک درصد). یکی از دلایلی که F# کمتر از سایر زبان‌ها مورد توجه است (البته تا این زمان) نبود منبع یا کتاب فارسی در زمینه یادگیری و هم چنین عدم شناخت از امکانات و قدرت این زبان است. در نتیجه تصمیم گرفتم در طی دو یا چند دوره به آموزش برنامه نویسی این زبان بپردازم. دوره اول که از قسمت دوره‌ها (^)در این سایت در دسترس عموم قرار دارد سطوح مقدماتی و متوسط را پوشش می‌دهد (سرفصل‌های این دوره در قسمت آموزش F# ذکر شده است). به دلیل حجم گسترده مطالب امکان ارایه تمام مفاهیم و روش‌ها در طی یک دوره امکان پذیر نبود در نتیجه تصمیم بر آن شد که با توجه به اولویت‌های آموزشی این مطالب طبقه بندی شوند و طی دو یا چند دوره به دوستان عزیز ارائه شوند.
دوره ای که هم اکنون در دسترس است صرفا جهت آشنایی دوستان با نوع کدنویسی و مفاهیم برنامه نویسی این زبان تهیه شده است اما دوره پیشرفته این زبان که بعدا در طی چند فصل، آموزش داده خواهد شد دارای سرفصل‌های زیر خواهد بود:
استفاده از F# در پروژه‌های تولید شده با زبان C# و در محیط Visual Studio.Net

استفاده از EntityFramework در زبان F#
تولید و توسعه پروژهای Windows Application با زبان F#

تولید و توسعه پروژهای WPF با زبان F#
تولید و توسعه پروژه‌های تحت Silverlight با زبان F#
و...

موفق باشید.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۳/۲۵ ۱۰:۴۰

- عالی است. مطلبی رو اینجا دیدم در مورد « ۱۰ رفتار که باید هنگام استخدام یک برنامه نویس دنبالشان بگردید ». خصوصا این چند مورد ویژه که در مورد شما بیشتر صادق است؛ برخلاف جماعت صرفا کارمند :

۱- کنجکاوی

۵- یادگیری سریع

۶- مهارت‌های خودآموزی

۷- علاقه

نویسنده: مسعود م.پاکدل
تاریخ: ۱۳۹۲/۰۳/۲۵ ۱۱:۱

ممنونم جناب نصیری.

تشکر و قدردانی اصلی برای شماست به خاطر فراهم نمودن محیطی عالی برای ارائه مطالب.

نویسنده: اردلان شاه قلی
تاریخ: ۱۳۹۲/۰۹/۲۴ ۹:۳۲

سلام . لطفا آموزش #F را ادامه دهید. با تشکر

عنوان: **استفاده از F# در پروژه های WPF**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۱۷ ۸:۳۵
آدرس: www.dotnettips.info
برچسب‌ها: WPF, Programming, F#, FSharpx

در دوره F# این سایت (^) با نحوه کد نویسی و مفاهیم و مزایای این زبان آشنا شده اید. اما دانستن syntax یک زبان برای پیاده سازی یک پروژه کافی نیست و باید با تکنیک‌های مهم دیگر از این زبان آشنا شویم. همان طور که قبلا (فصل اول دوره F#) بیان شد Visual Studio به صورت Visual از پروژه‌های F# پشتیبانی نمی‌کند. یعنی امکان ایجاد یک پروژه WPF یا Windows Application یا حتی پروژه‌های تحت وب برای این زبان همانند زبان C# به صورت Visual در VS.Net تعبیه نشده است. حال چه باید کرد؟ آیا باید در این مواقع این گونه پروژه‌ها را با یک زبان دیگر نظیر C# ایجاد کنیم و از زبان F# در حل برخی مسائل محاسباتی و الگوریتمی استفاده کنیم. این اولین راه حلی است که به نظر می‌رسد. اما در حال حاضر افزونه هایی، توسط سایر تیم‌های برنامه نویسی تهیه شده اند که پیاده سازی و اجرای یک پروژه تحت ویندوز یا وب را به صورت کامل با زبان F# امکان پذیر می‌کنند. در این پست به بررسی یک مثال از پروژه WPF به کمک این افزونه‌ها می‌پردازیم.

نکته : آشنایی با کد نویسی و مفاهیم F# برای درک بهتر مطالب توصیه می‌شود.

معرفی پروژه FSharpX

پروژه FSharpx یک پروژه متن باز است که توسط یک تیم بسیار قوی از برنامه نویسان F# در حال توسعه می‌باشد. این پروژه شامل چندین زیر پروژه و بخش است که هر بخش آن برای یکی از مباحث دات نت در F# تهیه و توسعه داده می‌شود.

این قسمت‌ها عبارتند از :

FSharpx.Core : شامل مجموعه ای کامل از توابع عمومی، پرکاربرد و ساختاری است که برای این زبان توسعه داده شده اند و با تمام زبان‌های دات نت سازگاری دارند؛

FSharpx.Http : استفاده از F# در برنامه نویسی مدل Http؛

FSharpx.TypeProvider : این پروژه خود شامل چندین بخش است که در این جا چند مورد از آن‌ها را عنوان می‌کنم:

FSharpx.TypeProviders.AppSetting : متد خواندن و نوشتن (getter و setter) را برای فایل‌های تنظیمان پروژه (Application Setting File) فراهم می‌کند.

FSharpx.TypeProviders.Vector : برای محاسبات با ساختارهای برداری استفاده می‌شود.

FSharpx.TypeProviders.Machine : برای دسترسی و اعمال تغییرات در رجیستری و فایل‌های سیستمی استفاده می‌شود.

FSharpx.TypeProviders.Xaml : با استفاده از این افزونه می‌توانیم از فایل‌های Xaml، در پروژه‌های F# استفاده کنیم و WPF Designer نرم افزار VS.Net هم برای این زبان قابل استفاده خواهد شد.

FSharpx.TypeProviders.Regex : امکان استفاده از عبارات با قاعده را در این پروژه فراهم می‌کند.

یک مثال از عبارات با قاعده:

```
type PhoneRegex = Regex< @"(?<AreaCode>^\d{3})-(?<PhoneNumber>\d{3}-\d{4}$)">

PhoneRegex.IsMatch "425-123-2345"
|> should equal true

PhoneRegex().Match("425-123-2345").CompleteMatch.Value
|> should equal "425-123-2345"

PhoneRegex().Match("425-123-2345").PhoneNumber.Value
|> should equal "123-2345"
```

شروع پروژه

ایتدا یک پروژه از نوع F# Console Application ایجاد کنید. از قسمت Project Properties (بر روی پروژه کلیک راست کنید و گزینه Properties را انتخاب کنید) نوع پروژه را به Windows Application تغییر دهید(قسمت Out Put Type). اسمبلی‌های زیر را به پروژه ارجاع دهید:

PresentationCore

PresentationFramework

WindowBase

System.Xaml

با استفاده از پنجره Package Manager Console دستور نصب زیر را اجرا کنید(آخرین نسخه این پکیج 1.8.31 و حجم آن کمتر از یک مگابایت است):

```
PM> Install-Package FSharpx.TypeProviders.Xaml
```

حال یک فایل Xaml به پروژه اضافه کنید و کدهای زیر را در آن کپی کنید:

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="WPF F# Sample By Masoud Pakdel" Height="350" Width="525">
    <Grid Name="MainGrid">
        <StackPanel Name="StackPanel1" Margin="50">
            <Button Name="Button1">Who are you?</Button>
        </StackPanel>
    </Grid>
</Window>
```

کدهای بالا کاملا واضح است و نیاز به توضیح دیده نمی‌شود. اما اگر دقت کنید می‌بینید که این فایل، فایل Code Behind ندارد. برای این کار باید یک فایل جدید از نوع F# Source File ایجاد کنید. بهتر است که فایل جدید شما همنام با همین فایل باشد. پسوند این فایل fs است. حال کدهای زیر را در آن کپی کنید:

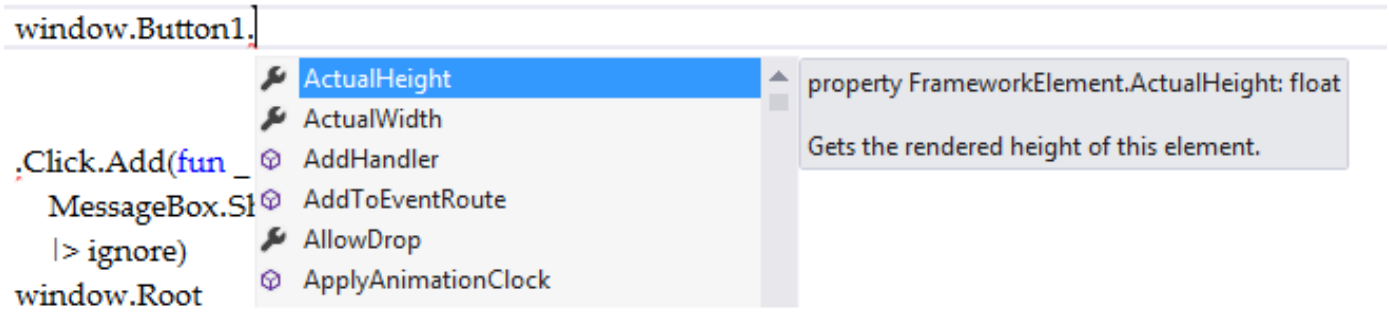
```
open System
open System.Windows
open System.Windows.Controls
open FSharpx

type MainWindow = XAML<"MainWindow.xaml">

let loadWindow() =
    let window = MainWindow()
    window.Button1.Click.Add(fun _ ->
        MessageBox.Show("Masoud Pakdel")
        |> ignore)
    window.Root

[<STAThread>]
(new Application()).Run(loadWindow())
|> ignore
```

نوع XAML استفاده شده که به صورت generic است در فضای نام FSharpx تعبیه شده است و این اجازه را می‌دهد که یک فایل F# بتواند برای مدیریت یک فایل Xaml استفاده شود.برای مثال می‌توانید به اشیا و خواص موجود در فایل Xaml دسترسی داشته باشید. در اینجا دیگر خبری از متد InitializeComponent موجود در سازنده کلاس CodeBehind پروژه‌های #C نیست. این تعاریف و آماده سازی کامپوننت‌ها به صورت توکار در نوع XAML موجود در FSharpx انجام می‌شود.



در تابع loadWindow یک نمونه از کلاس MainWindow ساخته می‌شود و برای button1 آن رویداد کلیک تعریف می‌کنیم. دستورات زیر معادل دستورات شروع برنامه در فایل program پروژه‌های C# است.

```
[<STAThread>]
(new Application()).Run(loadWindow())
|> ignore
```

پروژه را اجرا کنید و بر روی تنهای Button موجود در صفحه، کلیک کنید و پیغام مورد نظر را مشاهده خواهید کرد. به صورت زیر: