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

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 تفاورتی 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.