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.