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 پرانترها. 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.