**فصل 10**

**ساختارها**

* ساختار چیست؟
* ساختارها نوع های مقداری هستند
* مقداردهی یک ساختار
* سازنده ها و مخرب ها
* فیلدها مجاز نیست مقدار دهی اولیه شوند
* ساختارها مهر و موم شده هستند
* Boxing و Unboxing
* ساختارها به عنوان مقادیر بازگشتی و پارامترها
* اطلاعات اضافی درباره ساختارها

**ساختار چیست؟**

ساختارها نوع های تعریف شده توسط برنامه نویس و خیلی شبیه به کلاس ها هستند. آنها داده های عضو و توابع عضو دارند. اگر چه ساختارها شبیه به کلاس ها هستند اما در آنها تفاوت های مهمی وجود دارد. مهمترین تفاوت آنها عبارتند از:

* کلاس های نوع های ارجاعی، و ساختارها نوع های مقداری هستند.
* ساختارها به طور ضمنی مهر و موم شده هستند، بدان معنی که از آنها نمی توان مشتق گرفت.

نحو تعریف یک ساختار شبیه به تعریف یک کلاس است:

|  |
| --- |
| کلمه کلیدی  ↓  struct *StructName*  {  عضوهای تعریف شده  } |

به عنوان مثال، کد زیر ساختار Point را تعریف کرده است. که دو رشته public با نام های X و Y دارد. در متد Main، سه متغییر از نوع ساختار Point تعریف شده است، و مقادیری به آنها اختصاص یافته و این مقادیر را در خروجی چاپ می کند.

|  |
| --- |
| struct Point  {  public int X;  public int Y;  }  class Program  {  static void Main()  {  Point first, second, third;  first.X = 10; first.Y = 10;  second.X = 20; second.Y = 20;  third.X = first.X + second.X;  third.Y = first.Y + second.Y;  Console.WriteLine("first: {0}, {1}", first.X, first.Y);  Console.WriteLine("second: {0}, {1}", second.X, second.Y);  Console.WriteLine("third: {0}, {1}", third.X, third.Y);  }  } |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| first: 10, 10  second: 20, 20  third: 30, 30 |

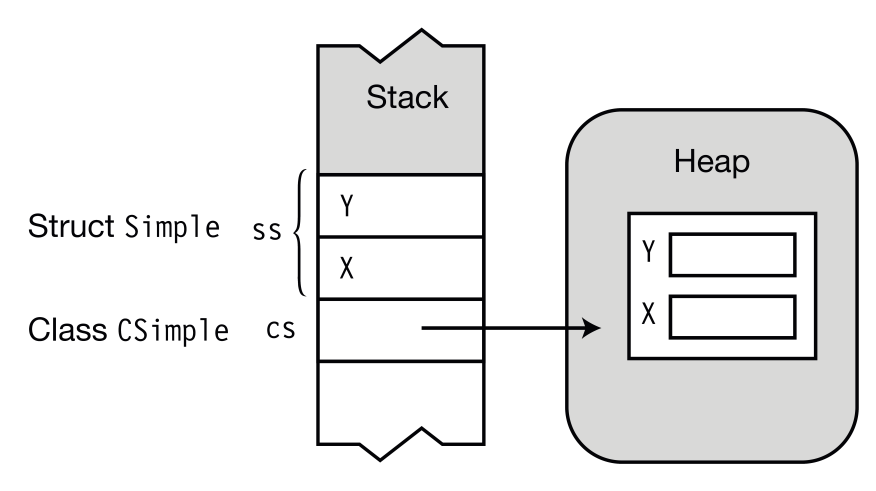
**ساختارهای نوع های مقداری هستند**

از آنجا که همه عضوها مقداری، یا یک متغییر از نوع ساختار، از داده های خاص خود شامل شده است. بنابراین:

* یک متغییر از نوع ساختار نمی تواند تهی (null) باشد.
* دو ساختار نمی توانند به یک شی اشاره کنند.

به عنوان مثال،کد زیر یک کلاس CSimple و یک ساختار Simple ایجاد کرده است و از هر یک متغییری تعریف می کند. شکل 10.1 چگونگی قرارگیری این دو را در حافظه نشان می دهد.

|  |
| --- |
| class CSimple  {  public int X;  public int Y;  }  struct Simple  {  public int X;  public int Y;  }  class Program  {  static void Main()  {  CSimple cs = new CSimple();  Simple ss = new Simple();  ... |



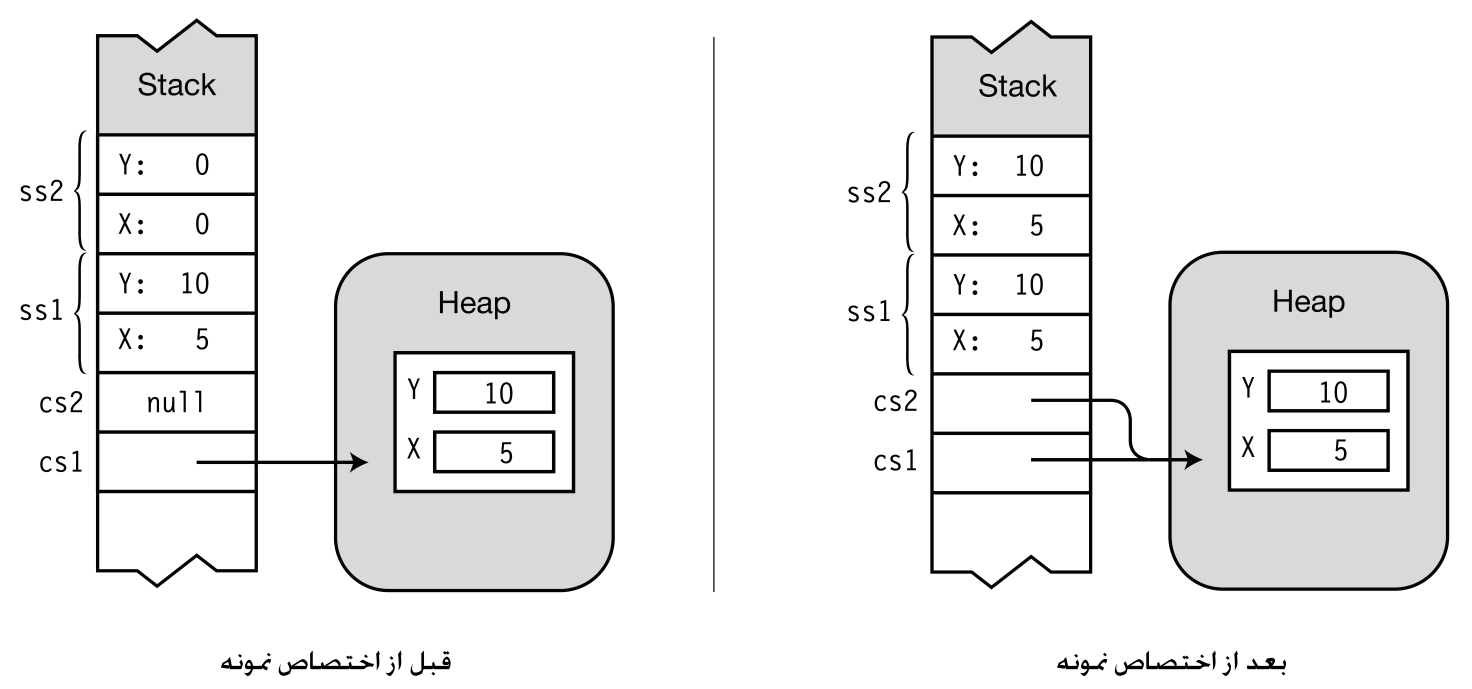
شکل 10.1 ترتیب قرار گیری کلاس و ساختار در حافظه

**مقداردهی یک ساختار**

در اختصاص دادن یک ساختار به ساختار دیگر، مقادیر از یکی به دیگری کپی می شود. این کاملا با کپی کردن یک تغییر کلاس تفاوت دارد، در متغییر کلاس تنها مرجع کلاس کپی می شود.

شکل 10.2 تفاوت بین اختصاص یک متغییر کلاس و یک متغییر ساختار را نشان می دهد. توجه داشته باشید که بعد از تخصیص دادن کلاس، مرجع cs2 به همان شی ای اشاره می کند که cs1 به آن اشاره می کند. اما بعد از تخصیص دادن ساختار، مقادیر از عضو ss1 به عضو ss2 کپی شده اند.

|  |
| --- |
| class CSimple  { public int X; public int Y; }  struct Simple  { public int X; public int Y; }  class Program  {  static void Main()  {  CSimple cs1 = new CSimple(), cs2 = null; // نمونه سازی از کلاس  Simple ss1 = new Simple(), ss2 = new Simple(); // نمونه سازی از ساختار  cs1.X = ss1.X = 5; // cs1.X و ss1.Xتخصیص 5 به  cs1.Y = ss1.Y = 10; // cs1.Y و ss1.Yتخصیص 10 به  cs2 = cs1; // اختصاص دادن کلاس نمونه سازی شده  ss2 = ss1; // اختصاص دادن ساختار نمونه سازی شده  }  } |



شکل 10.2 اختصاص یک متغییر کلاس و یک متغییر ساختار

**سازنده ها و مخرب ها**

ساختارها می تواند سازنده های نمونه و سازنده های استاتیکی داشته باشند، اما اجازه ندارند که مخرب داشته باشند.

سازنده های نمونه

این زبان به طور ضمنی یک سازنده بدون پارمتر برای هر ساختار ایجاد می کند. این سازنده هر یک از عضوهای ساختار را با مقدار پیش فرض آن نوع مقداردهی می کند. عضوهای مقداری با مقدار پیش فرض خود مقداردهی می شوند و عضوهای ارجاعی با null مقداردهی می شوند.

سازنده های پارمتری از پیش تعریف شده برای هر ساختار وجود دارد و نمی تواند آنها را حذف و یا دوباره آنها را تعریف کرد. با این حال، زمانی می توان سازنده های اضافه ایجاد کرد که آنها دارای دو پارمتر باشند. توجه کنید که این با کلاس ها تفاوت دارد. برای کلاس ها، اگر هیچ سازنده ای ایجاد نشده باشد، کامپایلر به طور ضمنی تنها یک سازنده بدون پارمتر برای آن ایجاد می کند.

برای فراخوانی یک سازنده بدون پارامتر، از عملگر new استفاده می شود. توجه داشته باشید حتی اگر حافظه از heap تخصیص داده نشود باز هم که عملگر new استفاده می شود.

به عنوان مثال، کد زیر ساختار simple را با یک سازنده دو پارامتر از نوع int تعریف کرده است. متد Main دو نمونه از این ساختار ایجاد کرده است یکی با سازنده بدون پارمتر ضمنی و دیگری با سازنده ای که دو پارامتر دارد.

|  |
| --- |
| struct Simple  {  public int X;  public int Y;  public Simple(int a, int b) // سازنده پارامتر دار  {  X = a;  Y = b;  }  }  class Program  {  static void Main()  { فراخوانی سازنده ضمنی  ↓  Simple s1 = new Simple();  Simple s2 = new Simple(5, 10);  ↑  فراخوانی سازنده  Console.WriteLine("{0},{1}", s1.X, s1.Y);  Console.WriteLine("{0},{1}", s2.X, s2.Y);  }  } |

همچنین می توان نمونه ای از ساختار را بدون استفاده از عملگر new ایجاد کرد. با این حال اگر این کار انجام شود محدودیت های متعددی ایجاد می کند که به شرح زیر است:

* تا زمانی که داده های عضو مقدار دهی نشوند نمی توان از آنها استفاده کرد.
* تا زمانی که تمام داده های عضو مقدار دهی نشوند نمی توان توابع عضو آن ساختار فراخوانی کنیم.

به عنوان مثال، کد زیر دو نمونه از ساختار Simple را بدون عملگر new ایجاد می کند. هنگامی که برای دستیابی به s1 تلاش می شود به دلیل این که داده عضو آن مقداردهی نشده است، کمپایلر یک پیغام خطا تولید می کند. اما هیچ مشکلی در خواندن s2 بعد از مقداردهی عضو های آن وجود ندارد.

|  |
| --- |
| struct Simple  {  public int X;  public int Y;  }  class Program  {  static void Main()  { سازنده فراخوانی نشده است  ↓ ↓  Simple s1, s2;  Console.WriteLine("{0},{1}", s1.X, s1.Y); // خطای کامپایلر  ↑ ↑  s2.X = 5; هنوز مقداردهی نشده است  s2.Y = 10;  Console.WriteLine("{0},{1}", s2.X, s2.Y); // صحیح است  }  } |

سازنده های استاتیک

مانند کلاس ها، در ساختارها سازنده های استاتیک ایجاد می شوند تا داده های عضو استاتیک را مقداردهی اولیه می کنند و نمی توانند عضوهای نمونه را مقداردهی کنند. همان قوانین که برای سازنده های استایک کلاس وجود دارد برای ساختارها نیز وجود دارد.

یک سازنده استاتیک قبل از هر یک از موارد زیر فراخوانی می شود:

* فراخوانی یک سازنده که به صراحت تعریف شده است.
* ارجاع به یک عضو استایک از ساختار.

**خلاصه ای از سازنده ها و مخرب ها**

جدول 10.1 استفاده از سازنده ها و مخرب ها را در ساختار خلاصه کرده است.

جدول 10.1 خلاصه ای از سازنده ها و مخرب ها

|  |  |
| --- | --- |
| **نوع** | **شرح** |
| سازنده نمونه ( بدون پارامتر) | نمی تواند در برنامه تعریف شود. یک سازنده ضمنی همیشه توسط سیستم برای تمام ساختارها ایجاد می شود. نمی تواند حذف شود یا در برنامه دوباره تعریف شود. |
| سازنده نمونه ( با پارامتر) | می تواند در برنامه تعریف شود. |
| سازنده استاتیک | می تواند در برنامه تعریف شود. |
| مخرب | نمی تواند در برنامه تعریف شود. اجازه تعریف سازنده وجود ندارد. |

**فیلد ها مجاز نیستند مقداردهی اولیه شوند**

همانطور که در کد زیر نشان داده شده است، مقداردهی اولیه فیلدها در دستورات ساختار اجازه داده نشده است:

|  |
| --- |
| struct Simple  { اجازه داده نشده  ↓  public int x = 0; // خطای کامپایلر  public int y = 10; // خطای کامپایلر  } ↑  اجازه داده نشده |

**ساختارها مهر و موم شده هستند**

ساختارها به طور ضمنی مهر و موم شده هستند، و از این رو ساختار های دیگر نمی توانند از آن مشتق شوند.

از آنجا که ساختارها ارث بری را پشتیبانی نمی کنند، استفاده از چند اصلاحگر عضوهای کلاس برای عضوهای ساختار احساس نمی شود، بنابراین نمی توان در دستورات از آنها استفاده کرد. اصلاحگرهایی که نمی توانند در ساختارها استفاده شوند به شرح زیر است:

* protected
* internal
* abstract
* virtual

ساختارها خودشان از System.ValueType مشتق شده اند، که آن نیز از شی مشتق شده است.

هنگامی عضوی با همان نام به عنوان عضو از کلاس پایه System.ValueType ایجاد می شود، که تمام ساختارها از آن مشتق شده اند، دو کلمه کلیدی (اصلاحگر) مرتبط با ارث بری می توانند با عضوهای ساختار استفاده شوند این کلمات، کلمات کلیدی new و override هستند.

**Boxing و Unboxing**

مانند دیگر داده های نوع مقداری، اگر بخواهید یک ساختار نمونه سازی شده را به عنوان شی نوع ارجاعی استفاده کنید باید یک جعبه کپی ایجاد کنید. Boxing فرایند ایجاد کپی یک نوع ارجاعی از متغییر نوع مقداری است. Boxing و Unboxing با جزئیات بیشتری در فصل 16 بررسی می شوند.

**ساختارها به عنوان مقادیر بازگشتی و پارامترها**

ساختارها می توانند به عنوان مقادیر بازگشتی و پارامترها استفاده شوند.

* مقدار بازگشتی : هنگامی که یک ساختار مقدار بازگتشتی است، در بازگشت از توابع یک کپی ایجاد می شود.
* پارامتر مقداری : هنگامی که یک ساختار به عنوان یک پارامتر مقداری استفاده می شود، یک کپی از پارامترهای واقعی ایجاد می شود، و این کپی در متد استفاده می شود.
* پارامترهای ref و out : اگر یک ساختار به عنوان پارامتر ref یا out استفاده شود، مرجع ساختار به عنوان پارامتر به متد ارسال می شود بنابراین داده های عضو می توانند تغییر کنند.

**اطلاعات اضافی درباره ساختارها**

نمونه سازی ساختارها نسبت به نمونه سازی یک کلاس سربار کمتری نیاز دارد، به طوری که گاهی اوقات می توان برای کارایی بهتر از ساختارها بجای کلاس ها استفاده کرد. اما مواظب هزینه بالای boxing و unboxing باشید.

درنهایت، برخی از مواردی که بازم است درباره ساختارها بدانید به شرح زیر است:

* اگر چه نوع های از پیش تعریف شده ساده (int، short، long و غیره)، در .NET و C# از ابتدا ایجاد شده اند اما در واقع در .NET به عنوان ساختار اجرا می شوند.
* همانطور که در فصل 6 کلاس های جزئی توضیح داده شد می توان ساختارهای جزئی را ایجاد کرد.

ساختارها، مانند کلاسها، می توانند واسط ها را پیاده سازی کنند، که در فصل 15 بررسی می شوند.