**فصل 11**

**نوع های شمارشی**

* نوع های شمارشی
* پرچم های بیت
* اطلاعات اضافی درباره نوع های شمارشی

**نوع های شمارشی**

نوع های شمارشی[[1]](#footnote-1) نوع های تعریف شده توسط برنامه نویسی است، مانند یک کلاس و یک ساختار است.

* مانند ساختارها، نوع های شمارشی نوع ارجاعی هستند و بنابراین داده ها آن به جای اینکه به طور جداگانه ذخیره شوند به طور مستقیم با مرجع و داده ها ذخیره می شوند.
* نوع های شمارشی فقط یک نوع عضو دارند. به طور ثابت و با یک مقدار صحیح نام گذاری شده اند.

کد زیر نمونه ای از تعریف یک نوع شمارشی به نام TrafficLight را نشان می دهد، که شامل سه عضو است. توجه کنید که عضوها با کاما از یک دیگر جدا شده اند، هیچ سمی کالنی در تعریف نوع های شمارشی وجود ندارد.

|  |
| --- |
| کلمه کلیدی نام  ↓ ↓  enum TrafficLight  {  Green, ← با کاما جدا شده است – هیچ سمی کالنی وجود ندارد  Yellow, ← با کاما جدا شده است – هیچ سمی کالنی وجود ندارد  Red  } |

هر نوع شمارشی یک نوع زمینه ای صحیح دارد، که به طور پیش فرض از نوع int است.

* هر عضو شمارشی با یک مقدار ثابت از نوع زمینه ای مقداردهی شده است.
* به طور پیش فرض کامپایلر اولین عضو با مقدار صفر و هر عضو را با یک واحد بیشتر از عضو قبلی مقداردهی می کند.

به عنوان مثال، در نوع TrafficLight، کامپایلر به ترتیب مقادیر 0 و 1 و 2 را به عضو های Green و Yellow و Red اختصاص می دهد. در خروجی کد می توان مقادیر عضو زمینه ای را با مرجع گیری آنها برای نوع int مشاهده کنید. شکل 11.1 نحو قرار گیری آن را در stack نشان می دهد.

|  |
| --- |
| TrafficLight t1 = TrafficLight.Green;  TrafficLight t2 = TrafficLight.Yellow;  TrafficLight t3 = TrafficLight.Red;  Console.WriteLine("{0},\t{1}", t1, (int) t1);  Console.WriteLine("{0},\t{1}", t2, (int) t2);  Console.WriteLine("{0},\t{1}\n", t3, (int) t3);  ↑  intمرجع گیری برای |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Green, 0  Yellow, 1  Red, 2 |



شکل 11.1 عضوهای ثابت از هر نوع شمارشی با مقادیر صحیح نشان دهده می شوند.

با عضوهای نوع های شمارشی می توان متغییرهای آن نوع شمارشی را مقداردهی کرد. به عنوان مثال، کد زیر سه متغییر از نوع TrafficLight تعریف کرده است. توجه کنید که می توان لیترال های عضو را برای متغییرها اختصاص داد، یا می توان مقدار یک متغییر دیگر را در یک متغییر از همان نوع کپی کرد.

|  |
| --- |
| class Program  {  static void Main()  { نوع متغییر عضو  ↓ ↓ ↓  TrafficLight t1 = TrafficLight.Red; // مقداردهی از یک عضو  TrafficLight t2 = TrafficLight.Green; // مقداردهی از یک عضو  TrafficLight t3 = t2; // مقداردهی از یک متغییر  Console.WriteLine(t1);  Console.WriteLine(t2);  Console.WriteLine(t3);  }  } |

این کد خروجی زیر را تولید می کند. توجه کنید که نام عضوها به عنوان رشته چاپ می شوند.

|  |
| --- |
| Red  Green  Green |

تعیین نوع زمینه ای و مقادیر صریح

می توان یک نوع صحیح غیر از نوع int برای نوع زمینه ای استفاده کنید. با قرار دادن دو نقطه و نوع زمینه ای بعد از نوع شمارشی این کار انجام می شد. نوع زمینه ای می تواند هر نوع صحیح باشد. تمام عضو های نوع شمارشی از نوع زمینه ای هستند.

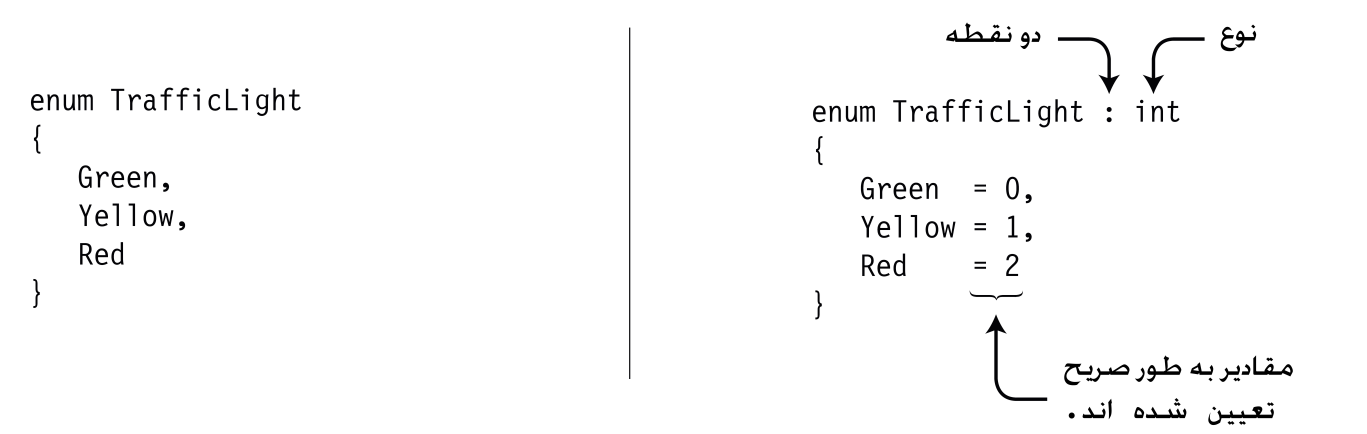
|  |
| --- |
| دو نقطه  ↓  enum TrafficLight : ulong  { ↑  ... نوع زمینه ای |

مقادیر عضو می توانند هر مقداری از نوع زمینه ای باشند. به طور صریح مقدار عضو با مقداردهی اولیه بعد از نام آن عضو تعیین می شود.همانطور که در زیر نشان داده شده است، مقادیر تکراری می تواند وجود داشته باشد، اما نام عضو نمی تواند تکراری باشد:

|  |
| --- |
| enum TrafficLight  {  Green = 10,  Yellow = 15, // مقادیر تکراری  Red = 15 // مقادیر تکراری  } |

به عنوان مثال، کد موجود در شکل 11.2 دو تعریف از نوع شمارشی TrafficLight را نشان می دهد، که معادل یکدیگر اند.

* کد سمت چپ به طور پیش فرض نوع و نام ها تعیین می شود.
* کد سمت راست به صراحت نوع زمنیه ای int و مقادیر عضو را مطابق با مقادیر پیش فرض تعیین می کند.



شکل 11.2 دستورات نوع های شمارشی معادل

شماره گذاری عضو به طور ضمنی

به صراحت می توان مقادیری برای هر یک از عضوها تعیین کرد. اگر عضوی مقداردهی اولیه نشود، کامپایلر به طور ضمنی مقداری برای آن تعیین می کند.شکل 11.3 قوانینی که کامپایلر برای مقداردهی این عضوها استفاده می کند را نشان می دهد.

* مقادیر مرتبط با نام عضوها لازم نیست به طور مجزا باشند.



شکل 11.3 الگریتم برای تخصیص مقادیر عضو

به عنوان مثال، کد زیر دو نوع شمارشی تعریف می کند. همانطور که در توضیحات کد نشان داده شده است، CardSuit به طور ضمنی مقداردهی می شود. مقادیر برخی از عضوهای FaceCards به صراحت تعیین شده است و عضو های دیگر به طور ضمنی تعیین می شوند.

|  |
| --- |
| enum CardSuit  {  Hearts, // 0 - زیرا اولین عضو است  Clubs, // 1 - یک واحد بیشتر از مقدار قبلی  Diamonds, // 2 - یک واحد بیشتر از مقدار قبلی  Spades, // 3 - یک واحد بیشتر از مقدار قبلی  MaxSuits // 4 - یک روش مشترک برای تعیین مقادیر عضوهای لیست شده  }  enum FaceCards  {  //عضو // مقدار اختصاص داده شده  Jack = 11, // 11 - به صراحت تعیین شده  Queen, // 12 - یک واحد بیشتر از مقدار قبلی  King, // 13 - یک واحد بیشتر از مقدار قبلی  Ace, // 14 - یک واحد بیشتر از مقدار قبلی  NumberOfFaceCards = 4, // 4 - به صراحت تعیین شده  SomeOtherValue, // 5 - یک واحد بیشتر از مقدار قبلی  HighestFaceCard = Ace // 14 - در بالا تعریف شده استAce  } |

**پرچم های بیت**

برنامه نویسان برای تعیین روشن یا خاموش بودن بیت های یک کلمه از بیت های مختلفی استفاده می کنند. در این بخش به این کلمه، کلمه پرچم[[2]](#footnote-2) می گوییم. نوع های شمارشی یک راه مناسب برای انجام این کار ارائه می دهد.

مراحل به شرح زیر است:

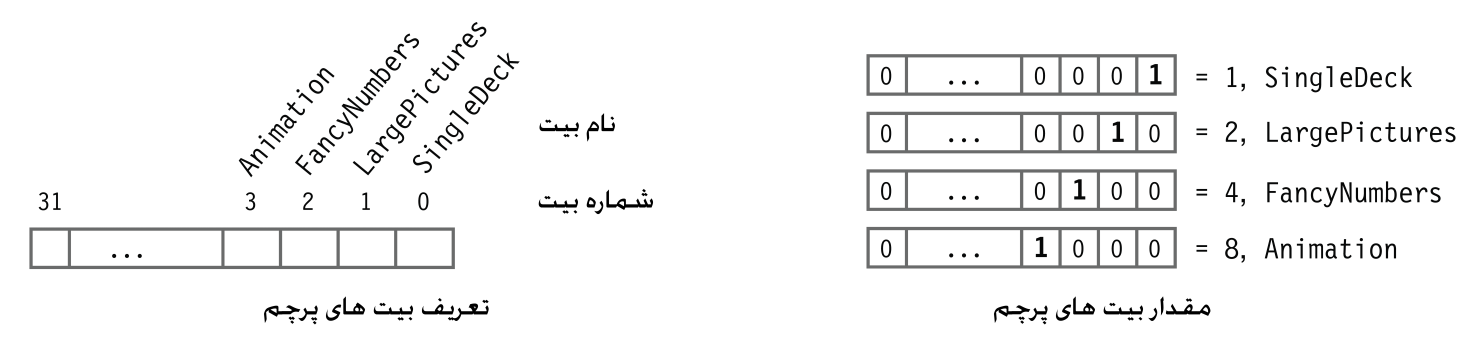
* تعداد پرچم های بیت[[3]](#footnote-3) که به آنها نیاز دارید را انتخاب و تعیین کنید و نوع زمینه را یک نوع صحیح بدون علامت با بیت های مورد نیاز انتخاب کنید.
* موقعیت هر بیت را تعیین کنید و به آن یک نام بدهید. یک نوع شمارشی با نوع زمینه انتخاب شده و با تمام عضوهای مورد نیاز تعریف کنید.
* با استفاده از عملگر بیتی OR در یک کلمه نگه دارنده پرچم های بیت، مجوعه ای از بیت های مناسب آن را نگه دارید.
* با متد HasFlag یا عملگر بیتی AND می توانید بررسی کنید که یک بیت خاص تعیین شده است.

به عنوان مثال، کد زیر نوع شمارشی را نشان میدهد که نماینده ای برای گذینه های ورق در بازی ورق است. نوع زمینه، uint است که برای نگه داری چهار پرچم کافی است. به موارد زیر در کد توجه کنید:

* عضوها نام هایی دارند که گذینه های باینری را نشان می دهد.
  + هر گذینه موقعیت یک کلمه را نشان می دهد. پوزیشن بیت ها با 0 یا 1 نگه داشته می شود.
  + از آنجا که پرچم های بیت نشان دهنده مجموعه ای از بیت ها است که هر کدام یا روشن و یا خاموش هستند، پس نمی خواهیم از 0 به عنوان مقدار یک عضو استفاده کنیم. این کار به این معنی است که تمام پرچم های بیت خاموش هستند.
* با مبنای شانزده، هر رقم مبنای شانزده دقیقا چهار بیت را نشان دهد. در هنگام کار با الگوهای بیتی، اغلب اعداد مبنای شانزده بجای اعداد دهدهی استفاده می شود زیرا بین الگوهای بیتی و نماینده مبنای شانزده آنها ارتباط مستقیمی وجود دارد.
* تعریف نوع شمارشی با ویژگی Flags ضروری نیست، اما راحتی بیشتری می دهد، که به زودی آن را بررسی می کنیم. یک ویژگی با رشته ای بین براکت نشان داده می شود. در این مورد، ویژگی قبل از نوع شمارشی است. ویژگی ها را در فصل 24 بررسی خواهیم کرد.

|  |
| --- |
| [Flags]  enum CardDeckSettings : uint  {  SingleDeck = 0x01, // Bit 0  LargePictures = 0x02, // Bit 1  FancyNumbers = 0x04, // Bit 2  Animation = 0x08 // Bit 3  } |

شکل 11.4 این نوع شمارشی را نشان می دهد.



شکل 11.4 تعریف بیت های پرچم (سمت چپ)، با نمایش هر یک از آنها ( سمت راست)

برای ایجاد یک کلمه با بیت های مناسب، یک متغییر از نوع شمارشی ایجاد کرده و با استفاده از عملگر OR، بیت های مورد نیاز را برای آن تعیین می کنیم. به عنوان مثال، کد زیر از چهار بیت موجود در نوع شمارشی سه بیت را برای کلمه پرچم تعیین می کند.

|  |
| --- |
| نوع شمارشی کلمه پرچم بیتی قرار گرفته اند ORپرچم های بیت با یکدیگر مورد ارزیابی  ↓ ↓ ↓  CardDeckSettings ops = CardDeckSettings.SingleDeck  | CardDeckSettings.FancyNumbers  | CardDeckSettings.Animation ; |

برای اطمینان از اینکه آیا برای کلمه پرچم یک بیت خاص تعیین شده است، می توان از متد HasFlag که یک نوع بویلن برمی گرداند استفاده کرد. همانطور که در کد زیر نشان داده شده است، متد HasFlag روی کلمه پرچم فراخوانی می شود، و پرچم بیت چک شونده به متد ارسال می شود.

|  |
| --- |
| bool useFancyNumbers = ops.HasFlag(CardDeckSettings.FancyNumbers);  ↑ ↑  کلمه پرچم پرچم بیت |

همچنین متد HasFlag می تواند برای چندین پرچم های بیت بررسی شود. به عنوان مثال، کد زیر بررسی می کند که آیا بیت های Animation و FancyNumbers برای کلمه پرچم ops تعیین شده اند یا خیر. کد موارد زیر انجام می دهد:

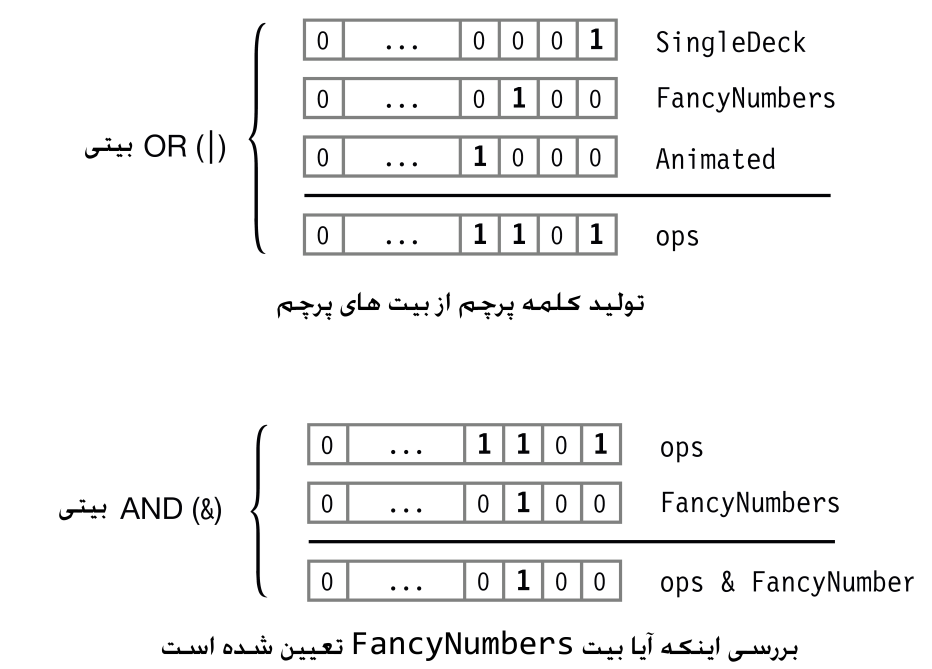
* در اولین دستور یک کلمه برای تست با نام testFlags ایجاد می کند که این کلمه با بیت های Animation و FancyNumbers تعیین شده است.
* سپس کلمه testFlags را به عنوان پارامتر به متد HasFlag ارسال می کند.
* متد HasFlag بررسی می کند آیا تمام پرچم های این کلمه (testFlags) در کلمه پرچم ops تعیین شده اند یا خیر. اگر بیت های کلمه testFlags در ops باشد متد HasFlag مقدار true و در غیر این صورت مقدار false برمی گرداند.

|  |
| --- |
| CardDeckSettings testFlags =  CardDeckSettings.Animation | CardDeckSettings.FancyNumbers;  bool useAnimationAndFancyNumbers = ops.HasFlag( testFlags );  ↑ ↑  کلمه پرچم کلمه ارزیابی |

روش دیگری برای تشخیص اینکه آیا یک یا چند پرچم بیتی خاص تعیین شده اند یا خیر، استفاده از عملگر بیتی AND است. به عنوان مثال، کد زیر مانند کد بالا، مشخص می کند آیا پرچم بیت FancyNumbers در کلمه پرچم تعیین شده است یا خیر. برای کار بین کلمه پرچم و پرچم بیت (های) AND بیتی قرار می دهیم و نتیجه را با خود پرچم بیت (ها) مقایسه می کنیم. اگر پرچم بیت (ها) در کلمه پرچم وجود داشته باشد نتیجه عملگر AND با خود پرچم بیت (ها) یکسان خواهد شد.

|  |
| --- |
| bool useFancyNumbers =  (ops & CardDeckSettings.FancyNumbers) == CardDeckSettings.FancyNumbers;  ↑ ↑  کلمه پرچم پرچم بیت |

شکل 11.5 روند ایجاد کلمه پرچم و استفاده آن با عملگر بیتی AND برای تعیین وجود بیت خاص را نشان می دهد.



شکل 11.5 ایجاد یک کلمه پرچم و بررسی آن برای پرچم بیت خاص

ویژگی Flags

در کد قبل از ویژگی Flags قبل از تعریف نوع شمارشی استفاده شده است مانند کد زیر:

|  |
| --- |
| [Flags]  enum CardDeckSettings : uint  {  ...  } |

ویژگی Flags در تمام محاسبات تغییر ایجاد نمی کند. اما چندین ویژگی مناست ارائه می دهد. اول اینکه، آن را به کامپایلر، مرورگر شی و دیگر ابزار در کد اطلاع می دهد که عضو های نوع شمارشی برای ترکیب کردن پرچم های بیت هستند، آنها به عنوان مقادیر جداگانه استفاده می شوند. این اجازه می دهد تا متغییرهای نوع شمارشی مناسب تر تفسیر شوند.

دوم اینکه، این اجازه می دهد تا متد ToString نوع شمارشی برای مقادیر پرچم های بیت مناسب تر قالب بندی شود. متد ToString یک مقدار نوع شمارشی را می گیرد و آن با مقادیر عضوهای نوع شمارشی مقایسه می کند. اگر با یکی از عضوها هماهنگ بود متد ToString نام آن عضو را به عنوان رشته برمی گرداند.

به عنوان مثال، کد زیر که نوع شمارشی با ویژگی Flags آغاز نشده است را بررسی می کند.

|  |
| --- |
| enum CardDeckSettings : uint  {  SingleDeck = 0x01, // bit 0  LargePictures = 0x02, // bit 1  FancyNumbers = 0x04, // bit 2  Animation = 0x08 // bit 3  }  class Program  {  static void Main( )  {  CardDeckSettings ops;  ops = CardDeckSettings.FancyNumbers; // تعیین یک پرچم بیت  Console.WriteLine( ops.ToString() );  // تعیین دو پرچم بیت  ops = CardDeckSettings.FancyNumbers | CardDeckSettings.Animation;  Console.WriteLine( ops.ToString() ); // چه چیزی چاپ می کند؟  }  } |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| FancyNumbers  12 |

در این کد، متد Main موارد زیر را انجام می دهد:

* متغییری از نوع شمارشی با نام CardDeckSettings ایجاد می کند، یک از پرچم های بیت خود را تعیین می کند و مقدار متغییر را چاپ می کند که مقدار FancyNumbers است.
* مقدار جدیدی به متغییر اختصاص می دهد که با دو پرچم های بیتی تعیین می شود و مقدار آن را چاپ می کند که مقدار 12 است.

مقدار 12 به عنوان نتیجه دومین انتساب نمایش داده شده است زیرا مقدار 4 برای FancyNambers و مقدار 8 برای Animation تعیین شده است و مقدار 12 را می دهد. در متد WriteLine بعد از انتساب متد ToString تلاش می کند که نام عضو شمارشی با مقدار 12 را پیدا کند که هیچ عضوی با این مقدار نمی یابد بنابراین فقط آن مقدار را به عنوان خروجی چاپ می کند.

با این حال، اگر ویژگی Flags را قبل از تعریف نوع شمارشی اضافه کنیم، این به متد ToString خواهد گفت که بیت ها می توانند به طور جداگانه در نظر گرفته شوند. متد ToString بعد از جستجوی مقدار 12، به دنبال عضوهای پرچم های بیت جداگانه (FancyNambers و Animation ) می گردد و نام آنها را به عنوان رشته ای که توسط کاما از یک دیگر جدا شده اند، برمی گرداند. در زیر نتیجه ای که با ویژگی Flags تولید می شود را نشان می دهد:

|  |
| --- |
| FancyNumbers  FancyNumbers, Animation |

مثال از پرچم های بیت

کد زیر تمام قطعه کد هایی که در پرچم های بیتی استفاده کردیم را در کنار یکدیگر قرار می دهد:

|  |
| --- |
| [Flags]  enum CardDeckSettings : uint  {  SingleDeck = 0x01, // bit 0  LargePictures = 0x02, // bit 1  FancyNumbers = 0x04, // bit 2  Animation = 0x08 // bit 3  }  class MyClass  {  bool UseSingleDeck = false,  UseBigPics = false,  UseFancyNumbers = false,  UseAnimation = false,  UseAnimationAndFancyNumbers = false;  public void SetOptions( CardDeckSettings ops )  {  UseSingleDeck = ops.HasFlag( CardDeckSettings.SingleDeck );  UseBigPics = ops.HasFlag( CardDeckSettings.LargePictures );  UseFancyNumbers = ops.HasFlag( CardDeckSettings.FancyNumbers );  UseAnimation = ops.HasFlag( CardDeckSettings.Animation );  CardDeckSettings testFlags =  CardDeckSettings.Animation | CardDeckSettings.FancyNumbers;  UseAnimationAndFancyNumbers = ops.HasFlag( testFlags );  }  public void PrintOptions( )  {  Console.WriteLine( "Option settings:" );  Console.WriteLine( " Use Single Deck - {0}", UseSingleDeck );  Console.WriteLine( " Use Large Pictures - {0}", UseBigPics );  Console.WriteLine( " Use Fancy Numbers - {0}", UseFancyNumbers );  Console.WriteLine( " Show Animation - {0}", UseAnimation );  Console.WriteLine( " Show Animation and FancyNumbers - {0}",  UseAnimationAndFancyNumbers );  }  } |

|  |
| --- |
| class Program  {  static void Main( )  {  MyClass mc = new MyClass( );  CardDeckSettings ops = CardDeckSettings.SingleDeck  | CardDeckSettings.FancyNumbers  | CardDeckSettings.Animation;  mc.SetOptions( ops );  mc.PrintOptions( );  }  } |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Option settings:  Use Single Deck - True  Use Large Pictures - False  Use Fancy Numbers - True  Show Animation - True  Show Animation and FancyNumbers - True |

**اطلاعات اضافی درباره نوع های شمارشی**

نوع های شمارشی تنها یک نوع عضو دارد: عضو های تعریف شده عضو های ثابت هستند.

* نمی توان با عضو ها اصلاحگر استفاده کرد. تمام آنها به طور ضمنی به عنوان نوع های شمارشی قابل دسترس هستند.
* عضو ها به صورت static هستند. همانطور که به یاد دارید آنها در دسترس هستند حتی اگر هیچ متغییری از نوع شمارشی وجود نداشته باشد. همانند تمام نوع های استاتیک، آنها با نام نوع و علامت نقطه و نام عضو در دسترس هستند.

به عنوان مثال، کد زیر هیچ متغییری از نوع شمارشی TrafficLight ایجاد نمی کند اما عضوها به صورت استاتیک هستند، آنها قابل دسترس هستند و می توانند توسط متد WriteLine چاپ شوند.

|  |
| --- |
| static void Main()  {  Console.WriteLine("{0}", TrafficLight.Green);  Console.WriteLine("{0}", TrafficLight.Yellow);  Console.WriteLine("{0}", TrafficLight.Red);  } ↑ ↑  نام نوع شمارشی نام عضو |

یک نوع شمارشی، یک نوع ممجزا است. مقایسه عضوهای شمارشی از نوع های شمارشی مختلف، در زمان کامپایل باعث ایجاد یک خطا می شود. به عنوان مثال، کد زیر دو نوع شمارشی مختلف تعریف می کند که ساختار و عضوهای آنها یکسان هستند.

* اولین دستور if خوب است زیرا عضو های مختلفی از نوع شمارشی یکسان مورد مقایسه قرار گرفته اند.
* دومین دستور if یک خطا تولید می کند زیرا تلاش می کند که عضوهای نوع های شمارشی مختلف را با یکدیگر مقایسه کند. حتی اگر ساختارها و نام عضوها دقیقا یکسان باشد بازهم این خطا رخ می دهد.

|  |
| --- |
| enum FirstEnum // اولین نوع شمارشی  {  Mem1,  Mem2  }  enum SecondEnum // دومین نوع شمارشی  {  Mem1,  Mem2  }  class Program  {  static void Main()  {  if (FirstEnum.Mem1 < FirstEnum.Mem2) //خوب است -- عضوهای نوع های شمارشی یکسان  Console.WriteLine("True");  if (FirstEnum.Mem1 < SecondEnum.Mem1) // خطا -- نوع هاش شمارشی مختلف  Console.WriteLine("True");  }  } |

همچنین در نوع .NET Enum چندین متد مفید به صورت استاتیک وجود دارد.

* متد GetName یک شی از نوع شمارشی و یک عدد صحیح گرفته و نام نام عضو شمارشی مربوطه را برمی گرداند.
* متد GetNames یک شی از نوع شمارشی گرفته و نام تمام عضوهای آن نوع شمارشی را برمی گرداند.

کد زیر نمونه ای از استفاده هر یک از متدها را نشان می دهد. توجه کنید که برای گرفتن نوع شی شمارشی باید از عملگر typeof استفاده کنید.

|  |
| --- |
| enum TrafficLight  {  Green,  Yellow,  Red  }  class Program  {  static void Main()  {  Console.WriteLine( "Second member of TrafficLight is {0}\n",  Enum.GetName( typeof( TrafficLight ), 1 ) );  foreach ( var name in Enum.GetNames( typeof( TrafficLight ) ) )  Console.WriteLine( name );  }  } |

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| Second member of TrafficLight is Yellow  Green  Yellow  Red |

1. enumeration or enum [↑](#footnote-ref-1)
2. flag word [↑](#footnote-ref-2)
3. bit falgs [↑](#footnote-ref-3)