فصل 14

رویداد ها

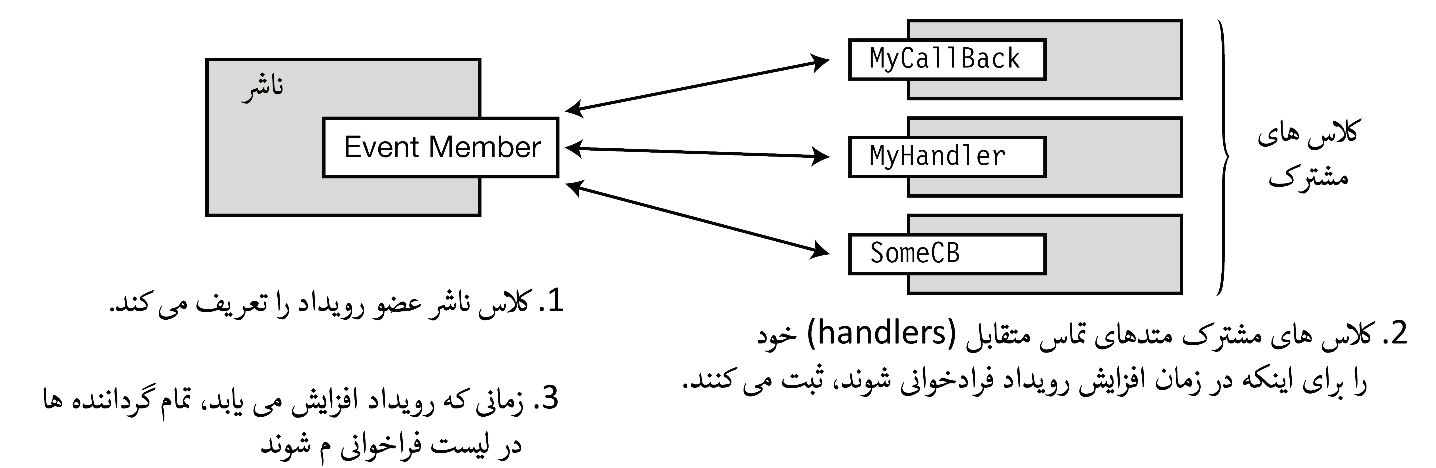
* ناشران و مشترکین
* مروری بر اجزای کد منبع
* تعریف یک رویداد
* اشتراک گذاری یک رویداد
* افزایش یک رویداد
* استفاده از رویداد استاندارد
* اکسسر های رویداد

انتشار مشترک

نیاز مشترک در بسیاری از برنامه ها این است که، زمانی که رویداد خاصی در برنامه رخ می دهد، بخش های دیگری از برنامه نیاز دارند که از رخداد این رویداد مطلع شوند.

الگویی برای برآورده این نیاز الگوی انتشار/مشترک[[1]](#footnote-1) نامیده می شود. در این الگو، یک کلاس، انتشار[[2]](#footnote-2) نامیده می شود، و مجموعه ای از رویداد ها را تعریف می کند که ممکن از در دیگر بخش های برنامه به آن علاقه مند شود. زمانی که رویدادی رخ می دهد، کلاس های دیگر می توانند برای اطلاع یافتن توسط ناشر "ثبت نام کنند". این کلاس های مشترک با اختصاص یک متد به ناشر از این رویداد آگاه می شوند. وقتی رویداد اتفاق می افتد، ناشر "این رویداد را افزایش می دهد،" و تمام متد های ارسال شده توسط مشترکان را اجرا می کند.

متد های ارائه شده توسط مشترکین، متد های تماس متقابل[[3]](#footnote-3) می نامند، زیرا ناشر با اجرا شدن متد خود متد مشترکان را فراخوانی می کند. همچین آنها گرداننده رویداد[[4]](#footnote-4) نیز نامیده می شوند. زیرا آنها کدی است که گفته می شود که مسئولیت رسیدگی به این رویداد است. شکل 1-14 این روند را به تصویر کشیده است، ناشر را با یک رویداد و سه مشترک برای این رویداد نشان می دهد.



شکل 1-14 ناشران و مشترکان

مواردی که مهم است درباره رویداد های بدانید به شرح زیر است:

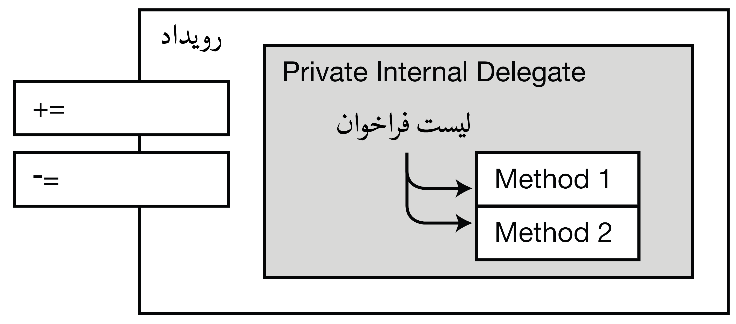
* *ناشر*: کلاس یا ساختاری است که رویداد را منتشر می کند به طوری که کلاس های می توانند از رخداد این رویداد مطلع شوند.
* *مشترک*: کلاس یا ساختاری است که برای اطلاع از وقوع رویداد ثبت نام می کنند.
* *گرداننده رویداد*: متدی است که توسط مشترک در ناشر ثبت شده است و اجرا زمانی صورت می گیرد که ناشر رویداد را افزایش می دهد. متد گرداننده رویداد را می توان در همان کلاس یا ساختار به عنوان رویداد یا در یک کلاس یا ساختار مختلف تعریف کرد.
* *افزایش رویداد*: واژه ای برای فراخوانی یک رویداد است. هنگامی که رویداد افزایش می یابد تمام متد های ثبت شده در آن رویداد فراخوانی می شوند.

در فصل قبل delegate ها بررسی شده است. در بسیاری از جنبه ها رویداد ها مانند delegate ها هستند. در واقع، رویداد مانند یک delegate ساده است که برای استفاده خاصی است. دلایلی برای شباهت رفتار delegate ها و رویداد ها وجود دارد. همانطور که در شکل 2-14 نشان داده شده است، یک رویداد شامل delegate خصوصی است.

مواردی که مهم است درباره delegate خصوصی رویداد ها بدانید به شرح زیر است:

* یک رویداد به ساختار delegate کنترل شده خصوصی آن دسترسی دارد. نمی توان به طور مستقیم به delegate دسترسی پیدا کرد.
* در مقایسه با delegate عملیات های کمی در دسترس است. با یک رویداد تنها می توان گرداننده رویداد را اضافه و حذف کرد و رویداد را فراخوانی کرد.
* هنگامی که رویداد افزایش می یابد، آن delegate را فراخوانی می کند، که به ترتیب متد های لیست فراخوان آن فراخوانی می شود.

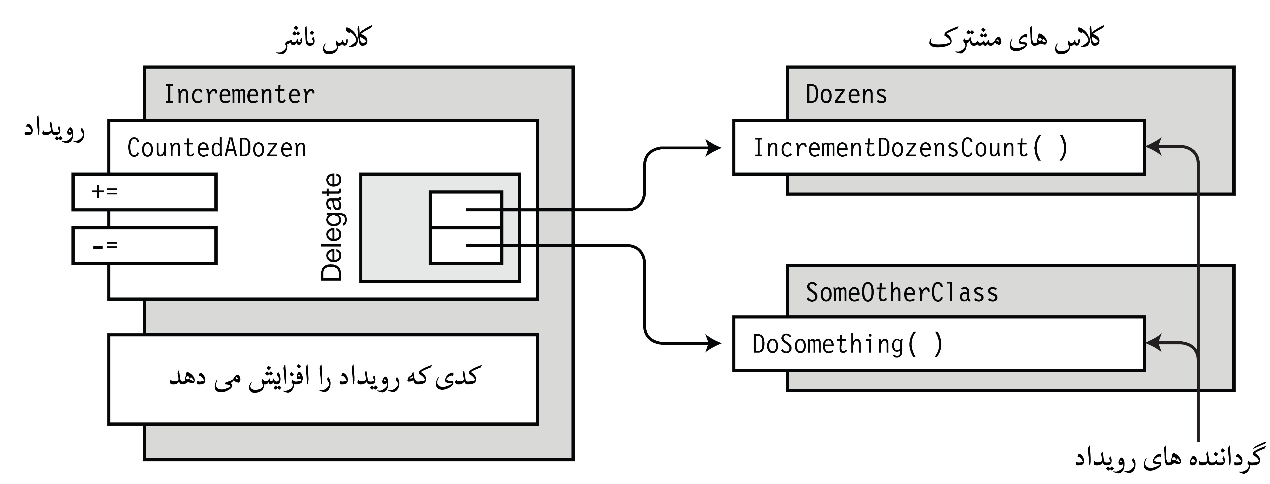
به شکل 2-14 توجه کنید که تنها عملگر های += و -= به سمت چپ جعبه رویداد چسبیده شده است. به این دلیل که تنها این عملیات ها (صرف نظر از فراخوانی خود رویداد) روی یک رویداد قابل انجام هستند.



شکل 2-14 رویداد دارای یک delegate محصور است.

شکل 3-14 برنامه ای را با کلاس Incrementer نشان می دهد.

* کلاس incrementer رویداد CountedADozen را تعریف کرده است، که هر زمانی که آیتم های دیگری را بشمارد آن را فزایش می دهد.
* کلاس های مشترک Dozens و SomeOtherClass هر کدام دارای گرداننده رویداد ثبت شده در رویداد CountedADozen هستند.
* در هر بار افزایش رویداد، گرداننده ها فراخوانی می شوند.

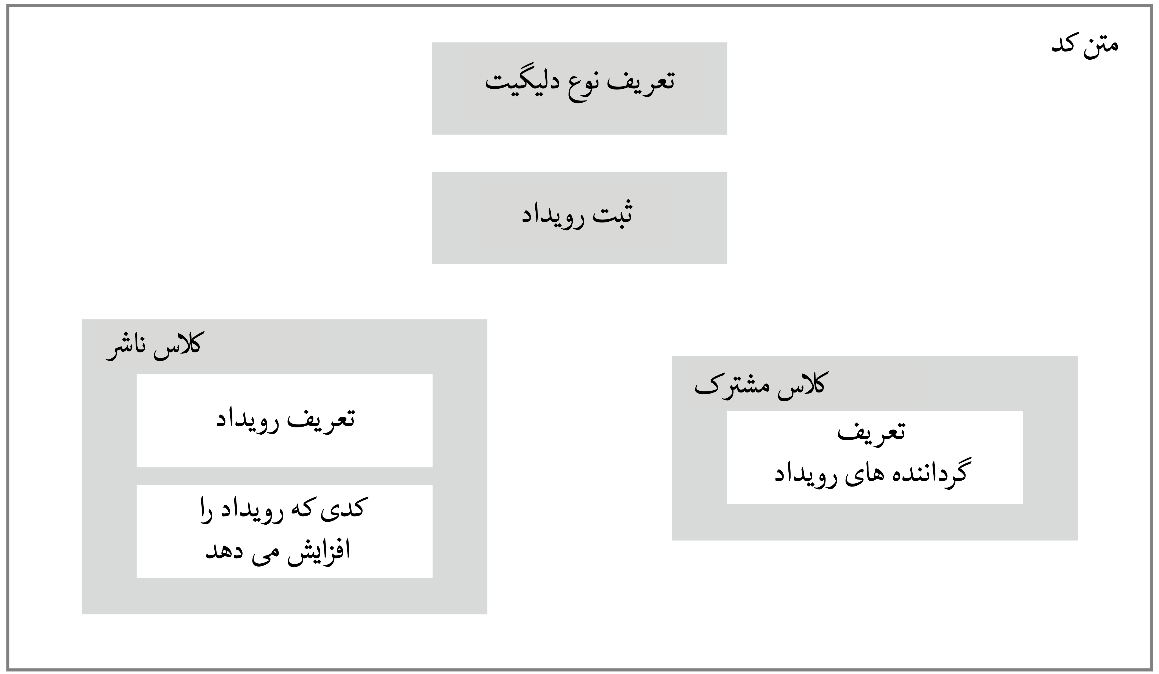


شکل 3-14 ساختار و اصطلاحات کلاس دارای رویداد

مروری بر اجزای کد منبع

پنج جزء از کد وجود دارد که نیاز است در جایی از رویداد ها استفاده کنند. این قطعات در شکل 4-14 نشان داده شده است. هر یک از آنها را در زیر بررسی می کنیم. این اجزا به شرح زیر هستند:

* تعریف نوع delegate: رویداد و گرداننده رویداد باید دارای امضا و نوع بازگشتی باشند که توسط delegate توصیف شده است.
* تعریف گرداننده رویداد: متد هایی که در کلاس های مشترک تعریف می شود تا در زمان افزایش رویداد فراخوانی شود. نیازی به متد های نامگذاری شده نیست؛ همانطور که در فصل 13 شرح داده شده است، می توان از متدهای ناشناس یا عبارات لامبدا استفاده کرد.
* تعریف رویداد: کلاس ناشر باید یک عضو رویداد را تعریف کند که کلاس های مشترک بتوانند توسط آن متد های خود را ثبت کنند. هنگامی که یک کلاس یک رویداد عمومی را تعریف می کند، به آن رویداد منتشر شده گفته می شود.
* ثبت رویداد: مشترکان باید برای اطلاع از رخداد رویداد متد خود را ثبت کنند. این کدی است که گرداننده رویداد را به رویداد متصل می کند.
* کد افزایش رویداد: این کد در ناشر است که باعث می شود تمام گرداننده های رویداد ثبت شده فراخوانی شوند.



شکل 4-14 پنج اجزای کد منبع استفاده شده در رویداد

تعریف رویداد

ناشر باید شی رویداد را ارائه کند. ایجاد یک رویداد ساده است ــ آن به یک نوع delegate و یک نام نیاز دارد. دستور تعریف یک رویداد در زیر نشان داده شده است، که رویداد CountedADozen را تعریف کرده است. به موارد زیر درباره رویداد CountedADozen توجه کنید:

* این رویداد در داخل کلاس تعریف شده است.
* این رویداد به یک delegate نیاز دارد. هر گرداننده رویداد متصل شده به رویداد (به عنوان مثال، ثبت شده در آن) باید در امضا و نوع بازگشتی با delegate مطابقت داشته باشد.
* این رویداد به طور public تعریف شده است، بنابراین کلاس ها و ساختار های دیگر می توانند گرداننده رویداد در آن ثبت کنند.
* از عبارت ایجاد کننده شی (عبارت new) برای یک رویداد استفاده نکنید.

class Incrementer

{ کلمه کلیدی نام رویداد

↓ ↓

public event EventHandler CountedADozen;

↑

Delegate نوع

می توان با استفاده از کاما بیش از یک رویداد تعریف کرد. به عنوان مثال، دستور زیر سه رویداد را تعریف می کند:

public event EventHandler MyEvent1, MyEvent2, OtherEvent;

↑

سه رویداد

همچنین همانطور که در دستور زیر نشان داده شده است، می توان با استفاده از کلمه کلیدی static رویداد های استاتیک ایجاد کرد.

public static event EventHandler CountedADozen;

↑

کلمه کلیدی

رویداد یک عضو است

اشتباه رایج این است که فکر کنیم یک رویداد همانند یک نوع است ـــ در حالی که این طور نیست. مانند یک متد یا یک خصوصیت، یک رویداد عضوی از کلاس یا ساختار است، و چندین پیامد مهم وجود دارد:

* از آنجایی که یک رویداد عضو است
  + نمی توان یک رویداد را در یک بلوک کد اجرایی تعریف کرد.
  + آن باید در داخل کلاس یا ساختار با دیگر عضو ها تعریف شود.
* یک عضو رویداد به طور ضمنی و به طور خودکار با null مقداردهی اولیه می شود.

برای تعریف یک رویداد، باید نام نوع delegate ارائه شود. هم می توان یکی ایجاد کرد یا از یکی که از قبل وجود داشته است را استفاده کنید. اگر یک نوع delegate تعریف کنید، آن باید مشخص کننده امضا و نوع بازگشتی متد هایی که در رویداد ثبت می شوند باشد.

BCL به طور خاص برای رویداد های سیستمی یک delegate با نام EventHandler تعریف می کند. دلیگیت EventHandler را بعدا در این فصل بررسی خواهیم کرد.

اشتراک گذاری یک رویداد

مشترکین، گرداننده های رویداد را به رویداد اضافه می کنند. برای اینکه یک گرداننده رویداد اضافه شود، گرداننده باید نوع بازگشتی و امضای delegate این رویداد را داشته باشد.

* همانطور که در کد زیر نشان داده شده است، از عملگر += برای اضافه کردن یک گرداننده رویداد به یک رویداد استفاده کنید. گرداننده رویداد در سمت راست عملگر قرار می گیرد.
* مشخصات گرداننده رویداد می تواند هر یک از موارد زیر باشد:
  + نام متد نمونه
  + نام متد استاتیک
  + متد ناشناس
  + عبارت لامبدا

به عنوان مثال، کد زیر سه متد به رویداد CountedADozen اضافه می کند. اولین یک متد نمونه است. دومین یک متد استاتیک است. سومین یک متد نمونه است که از فرم delegate استفاده می کند.

کلاس نمونه سازی شده متد نمونه

↓ ↓

incrementer.CountedADozen += IncrementDozensCount; // به صورت مرجع متد

incrementer.CountedADozen += ClassB.CounterHandlerB; // به صورت مرجع متد

↑ ↑

عضو رویداد متد استاتیک

mc.CountedADozen += new EventHandler(cc.CounterHandlerC); // به صورت دلیگیت

همانند delegate ها می توان برای اضافه کردن گرداننده رویداد از متد های ناشناس و عبارت لامبدا استفاده کرد. به عنوان مثال، کد زیر ابتدا از یک عبارت لامبدا و سپس از متد ناشناس استفاده می کند.

// عبارت لامبدا

incrementer.CountedADozen += () => DozensCount++;

// متد ناشناس

incrementer.CountedADozen += delegate { DozensCount++; };

افزایش یک رویداد

عضو رویداد به خودی خود تنها گرداننده های رویداد را که نیاز است فراخوانی شوند را نگه داری می کند.

Nothing happens with them unless the event is raised. You need to make sure there is code to do just that, at the appropriate times.

به عنوان مثال، کد زیر رویداد CountedADozen را افزایش می دهد. به موارد زیر درباره کد توجه کنید:

* قبل از افزایش رویداد، کد رویداد با null مقایسه می کند که آیا شامل گرداننده رویداد است. اگر رویداد null باشد، آن خالی است و نمی تواند اجرا شود.
* دستور افزایش رویداد همانند فراخوانی متد است:
  + از نام رویداد استفاده می کند، و به دنبال آن لیست پارامتر ها در داخل پرانتز قرار می گیرد.
  + لیست پارامتر باید با نوع delegate رویداد مطابقت داشته باشد.

if (CountedADozen != null) // اطمینان از داشتن متد برای اجرا کردن

CountedADozen (source, args); // افزایش رویداد

↑ ↑

نام رویداد لیست پارامتر

تعریف رویداد و کد افزایش رویداد را با هم در تعریف کلاس قرار دهید. کد شامل دو عضو است: رویداد و یک متد با نام DoCount، که رویداد را در زمان مناسب افزایش می دهد.

class Incrementer

{

public event EventHandler CountedADozen; // تعریف رویداد

void DoCount(object source, EventArgs args)

{

for( int i=1; i < 100; i++ )

if( i % 12 == 0 )

if (CountedADozen != null) // اطمینان از داشتن متد برای اجرا کردن

CountedADozen(source, args);

} ↑

افزایش متد

}

کد داخل شکل 5-14 برنامه کامل را با کلاس ناشر Incrementer و کلاس مشترک Dozens، نشان می دهد. به موارد زیر درباره کد توجه کنید:

* در سازنده کلاس مشترک Dozens، متد IncrementDozensCount را به عنوان گرداننده متد ارائه می دهد.
* در متد DoCount در کلاس Incrementer، رویداد CountedADozen را زمانی افزایش میدهد که افزایش یافته شمارنده بر 12 بخش پذیر باشد.



شکل 5-14 برنامه کامل با یک کلاس انتشار و یک کلاس مشترک، نشان دهنده پنج جزء لازم برای استفاده از یک رویداد

کدی موجود در شکل 5-14 خروجی زیر را تولید می کند:

|  |
| --- |
| Number of dozens = 8 |

استفاده از رویداد استاندارد

برنامه نویسی [[5]](#footnote-5)GUI رویداد گرا است به این معنی که در زمان اجرای برنامه می توان در هر زمان با رویداد هایی مانند کلیک کردن، فشردن کلیدی، و یا زمان سنج سیستم وقفه ای ایجاد کرد. زمانی که این اتفاق می افتد برنامه نیاز دارد که به این رویداد رسیدگی کند و سپس به کار خود ادامه دهد.

واضح است که مدیریت نا همزمان رویداد های برنامه، موقعیتی مناسب برای استفاده از رویداد های C# است. پنجره های برنامه نویسی GUI از رویداد های گسترده استفاده می کند که به صورت استاندارد وجود دارد. الگو های .NET Framework از آنها استفاده می کند. پایه و اساس الگو های استاندارد برای رویداد ها از نوع دلیگیت EventHandler استفاده می کند، که در فضای نام System تعریف شده است. کد خط زیر تعریفی از نوع دلیگیت EventHandler را نشان می هد. همه چیز درباره تعریف به شرح زیر است:

* اولین پارامتر برای نگه داری مرجع شی ای است که رویداد را بالا می برد. و از نوع Object است پس می تواند با هر نوعی مطابقت داشته باشد.
* دومین پارامتر برای نگه داری اطلاعات مناسبی از حالت رویداد (نوع رویداد) است.
* نوع بازگشتی void است. بنابراین مقداری را برنمی گرداند.

public delegate void EventHandler(object sender, EventArgs e);

پارامتر دوم در دلیگیت EventHandler شی ای از کلاس EventArgs است، که در فضای نام Syatem تعریف شده است. ممکن است اینگونه فکر کنید که چون پارامتر دوم برای ارسال اطلاعات است، شی ای از کلاس EventArgs قادر است بعضی از داده ها را ذخیره کند. که این یک تصور اشتباه است.

* کلاس EventArgs برای انتقال هیچ داده ای طراحی نشده است. آن برای گرداننده رویداد استفاده می شود که نیازی به ارسال داده وجود ندارد ــ و به طور کلی توسط آنها نادیده گرفته می شود.
* اگر بخواهید داده ای ارسال کنید، باید کلاسی با فیلد های مناسب برای نگه داری داده های ارسالی تعریف کنید که از EventArgs مشتق شده باشد.

اگر چه کلاس EventArgs داده ای را ارسال نمی کند، اما آن یک بخش مهم برای استفاده از دلیگیت EventArgs است. پارامتر های نوع های object و EventArgs، کلاس های پایه ای برای هر نوع واقعی است که در پارامتر ها استفاده می شود. این اجازه می دهد تا دلیگیت EventHandler یک امضا را ارائه دهد که برای تمام رویداد ها و گرداننده های رویداد کمترین وجه مشترک است، که اجازه می دهد بجای اینکه امضای های مختلفی برای هر رویداد تعریف شود، تمام رویداد ها دقیقا دارای دو پارامتر باشند.

اگر برنامه Incrementer را تغییر دهیم و از دلیگیت EventHandeler استفاده کنیم، برنامه به صورت کد موجود در شکل 6-14 خواهد شد. با موارد زیر درباره کد توجه کنید:

* دستور ایجاد دلیگیت Handler حذف شده است و از دلیگیت EventHandler که توسط سیستم تعریف شده است استفاده می شود.
* امضای گرداننده رویداد در کلاس مشترک باید در امضا ( و نوع بازگشت) دلیگیت رویداد مطابقت داشته باشد، که در حال حاضر از پارامتر های object و EventArgs استفاده شده است. در این مورد گرداننده رویداد IncrementerDozensCount، پارامتر های رسمی را نادیده می گیرد.
* کدی که رویداد را افزایش می دهد باید رویداد را با شی های مناسب برای نوع های پارامتر فراخوانی کند.



شکل 6-14 برنامه تغییر یافته Incrementer با استفاده از دلیگیت تعریف شده سیستم

ارسال داده با توسعه دادن EventArgs

برای ارسال داده در پارامتر دوم از خود گرداننده رویداد و با رعایت کنوانسیون های استاندارد، نیاز است که یک کلاس سفارشی مشتق شده از EventArgs تعریف کنید که بتواند داده ای مورد نیاز را ذخیره کند. نام کلاس باید با EventArgs به پایان برسد (کلاس باید از EventAgrs ارث بری کند). به عنوان مثال کد زیر یک کلاس سفارشی را تعریف می کند که می تواند عددی را در فیلد IterationCount ذخیره کند:

نام کلاس سفارشی کلاس پایه

↓ ↓

public class IncrementerEventArgs : EventArgs

{

public int IterationCount { get; set; } // ذخیره کردن یک عدد صحیح

}

حال که یک کلاس سفارشی برای ارسال داده در پارامتر دوم وجود دارد، نیاز به یک نوع دلیگیت است که از این کلاس سفارشی جدید استفاده کند. برای این منظور از نسخه جنریک دلیگیت EventHandler<> استفاده می شود. در فصل 17 جزئیات جنریک های C# بررسی شده است. بنابراین در حال حاضر تنها از آن استفاده می کنیم. برای استفاده از جنریک دلیگیت، موارد زیر را همانطور که در کدهای بعدی نشان داده شده است انجام می دهیم:

* نام کلاس سفارشی در بین علامت کوچکتر و بزرگتر قرار می گیرد.
* هر جا که از نوع دلیگیت سفارشی استفاده شده است از تمام رشته استفاده کنید. به عنوان مثال، این همان چیزی است که تعریف رویداد به نمایش می گذارد:

دلیگیت سفارشی با استفاده از کلاس سفارشی

↓

public event EventHandler<IncrementerEventArgs> CountedADozen;

↑

نام رویداد

استفاده از کلاس های سفارشی و دلیگیت های سفارشی در چهار بخش از کد با رویداد برخورد. به عنوان مثال، کد زیر کد Incrementer را بروز رسانی می کند، با استفاده از کلاس EventArgs کلاس سفارشی IncrementerEventArgs را ایجاد می کند و همچنین از جنریک دلیگیت EventHandler<IncrementerEventArgs> استفاده می کند.

public class IncrementerEventArgs : EventArgs // EventArgs کلاس سفارشی مشتق شده از

{

public int IterationCount { get; set; } // ذخیره یک عدد صحیح

}

class Incrementer دلیگیت سفارشی با استفاده از کلاس سفارشی

{ ↓

public event EventHandler<IncrementerEventArgs> CountedADozen;

public void DoCount() شی ای از کلاس سفارشی

{ ↓

IncrementerEventArgs args = new IncrementerEventArgs();

for ( int i=1; i < 100; i++ )

if ( i % 12 == 0 && CountedADozen != null )

{

args.IterationCount = i;

CountedADozen( this, args );

} ↑

} ارسال پارامتر ها هنگام افزایش رویداد

}

class Dozens

{

public int DozensCount { get; private set; }

public Dozens( Incrementer incrementer )

{

DozensCount = 0;

incrementer.CountedADozen += IncrementDozensCount;

}

void IncrementDozensCount( object source, IncrementerEventArgs e )

{

Console.WriteLine( "Incremented at iteration: {0} in {1}",

e.IterationCount, source.ToString() );

DozensCount++;

}

}

class Program

{

static void Main()

{

Incrementer incrementer = new Incrementer();

Dozens dozensCounter = new Dozens( incrementer );

incrementer.DoCount();

Console.WriteLine( "Number of dozens = {0}",

dozensCounter.DozensCount );

}

}

این برنامه خروجی زیر را تولید می کند، و تنها مرحله از نظر محاسبه هنگامی است که نام کامل کلاس از شی source فراخوانی می شود. نام کامل کلاس در فصل 21 بررسی می شود.

|  |
| --- |
| Incremented at iteration: 12 in Counter.Incrementer  Incremented at iteration: 24 in Counter.Incrementer  Incremented at iteration: 36 in Counter.Incrementer  Incremented at iteration: 48 in Counter.Incrementer  Incremented at iteration: 60 in Counter.Incrementer  Incremented at iteration: 72 in Counter.Incrementer  Incremented at iteration: 84 in Counter.Incrementer  Incremented at iteration: 96 in Counter.Incrementer  Number of dozens = 8 |

حذف گرداننده های رویداد

هنگامی که یک گرداننده رویداد انجام می شود، می توان آن را از رویداد حذف کرد. همانطور که کد زیر نشان داده شده است، یک گرداننده رویداد با استفاده از عملگر -= از رویداد حذف می شود.

p.SimpleEvent -= s.MethodB; // MethodBحذف گرداننده

به عنوان مثال، کد زیر دو گرداننده به رویداد SimpleEvent اضافه می کند، و سپس رویداد را بالا می برد. هر یک از گرداننده های رویداد فراخوانی می شوند و متنی را چاپ می کند. سپس گرداننده MethodB از رویداد حذف می شود و دوباره رویداد بالا می رود. تنها گرداننده MethodA متنی را در خروجی چاپ می کند.

class Publisher

{

public event EventHandler SimpleEvent;

public void RaiseTheEvent() { SimpleEvent( this, null ); }

}

class Subscriber

{

public void MethodA( object o, EventArgs e ) { Console.WriteLine("AAA"); }

public void MethodB( object o, EventArgs e ) { Console.WriteLine("BBB"); }

}

class Program

{

static void Main( )

{

Publisher p = new Publisher();

Subscriber s = new Subscriber();

p.SimpleEvent += s.MethodA;

p.SimpleEvent += s.MethodB;

p.RaiseTheEvent();

Console.WriteLine( "\r\nRemove MethodB" );

p.SimpleEvent -= s.MethodB;

p.RaiseTheEvent();

}

}

این کد خروجی زیر را تولید می کند:

|  |
| --- |
| AAA  BBB  Remove MethodB  AAA |

اگر یک گرداننده رویداد بیش از یک بار در رویداد ثبت شده باشد، وقتی که دستور حذف گرداننده داده می شود تنها آخرین نمونه از این گرداننده از لیست حذف خواهد شد.

اکسسر های رویداد

آخرین موضوع در این فصل اکسسر های رویداد است. قبلا عملگر های += و -= ذکر شد که تنها عملگر های مجاز برای یک رویداد است. این دو عملگر دارای رفتار هایی است که در این فصل آنها را مشاهده کردید.

با این حال، تغییر رفتار این عملگر ها وجود دارد و رویداد می تواند هر کد سفارشی که از آن استفاده کرده اید را شامل شود. این یک موضوع پیشرفته است، با این حال، پس در اینجا تنها به آن اشاره می شود.

برای تغییر عملگر های آن باید اکسسر های رویداد را برای آن تعریف کرد.

* دو اکسسر وجود دارد: add و remove.
* تعریف یک رویداد با اکسسر ها شبیه تعریف خصوصیت ها به نظر می رسد.

مثال زیر تعریفی از رویداد را با اکسسر های آن نشان می دهد. هر دو اکسسر دارای مقدار ضمنی

1. Publisher/subscriber [↑](#footnote-ref-1)
2. Publisher [↑](#footnote-ref-2)
3. Callback method [↑](#footnote-ref-3)
4. Event handler [↑](#footnote-ref-4)
5. رابط کاربری گرافیکی [↑](#footnote-ref-5)