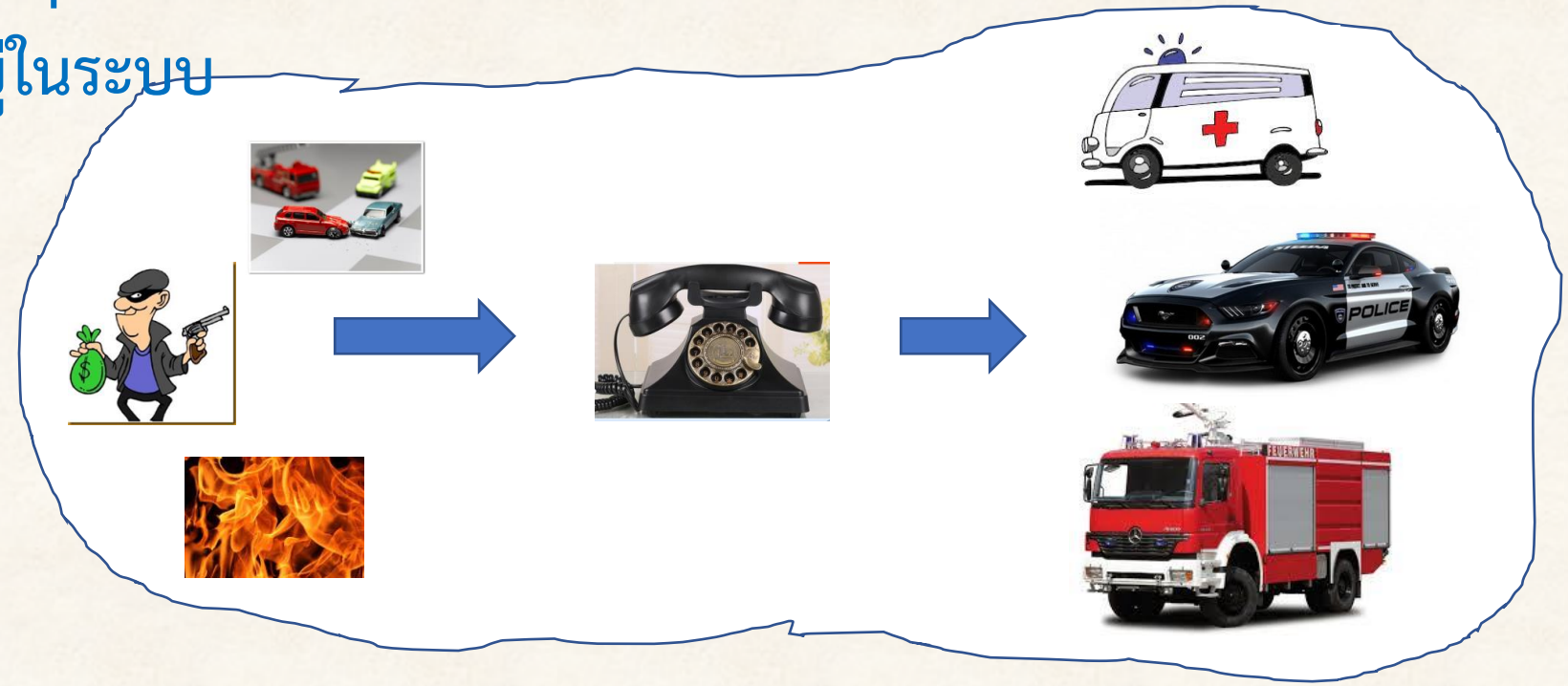
การเขียนโปรแกรม ด้วยภาษา C#

Events

Events

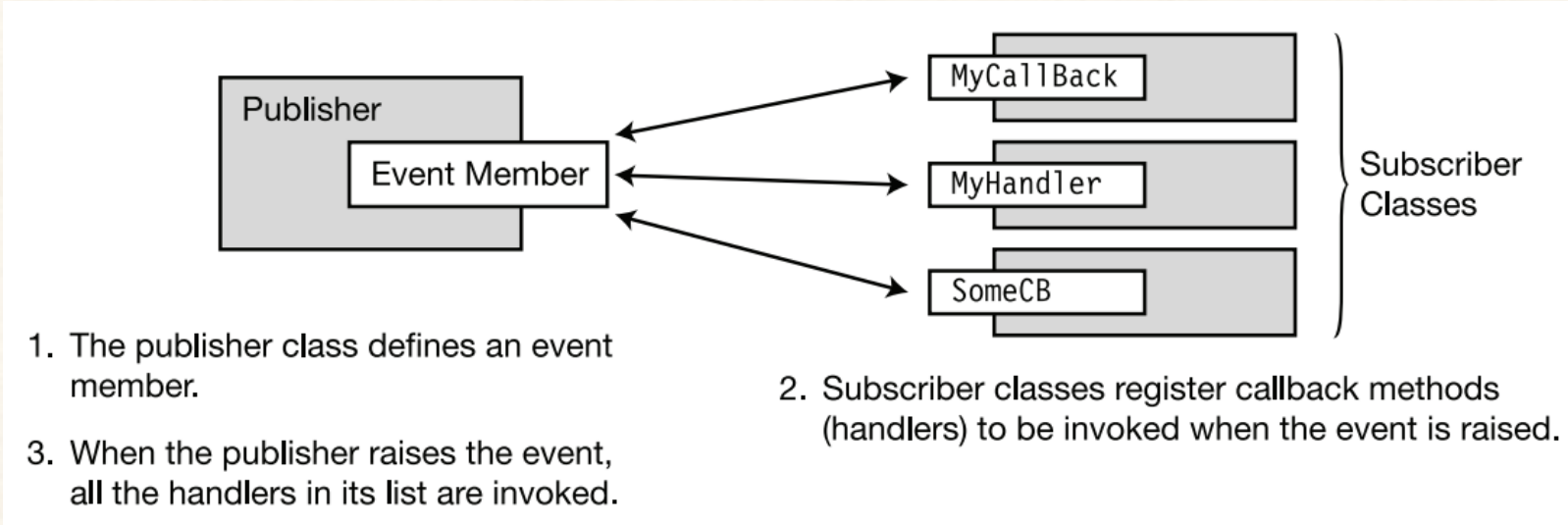
Publishers and Subscribers

- โดยปกติแล้ว ส่วนต่างๆ ของโปรแกรม ควรดำเนินการไปตามปกติ ที่ผู้ออกแบบโปรแกรมกำหนดไว้
- ในบางครั้ง ส่วนต่างๆ ของโปรแกรม ก็จะดำเนินการโดยไม่สนใจว่ามีโค้ดอะไรบ้าง อยู่ในโปรแกรมหรืออยู่ในระบบ



publisher/subscriber pattern

- คลาสที่เป็น publisher จะทำการกำหนด event ขึ้นมาชุดหนึ่ง
- คลาสอื่นๆ จะทำการลงทะเบียนเมธอดของตัวเอง ไว้กับ event ของ publisher เรียกว่า callback method
- เมื่อมีเหตุการณ์เกิดขึ้น คลาส publisher จะผลิต event (raises the event) จากนั้น ระบบก็จะเรียกเมธอดที่ลงทะเบียนไว้มาทำงาน

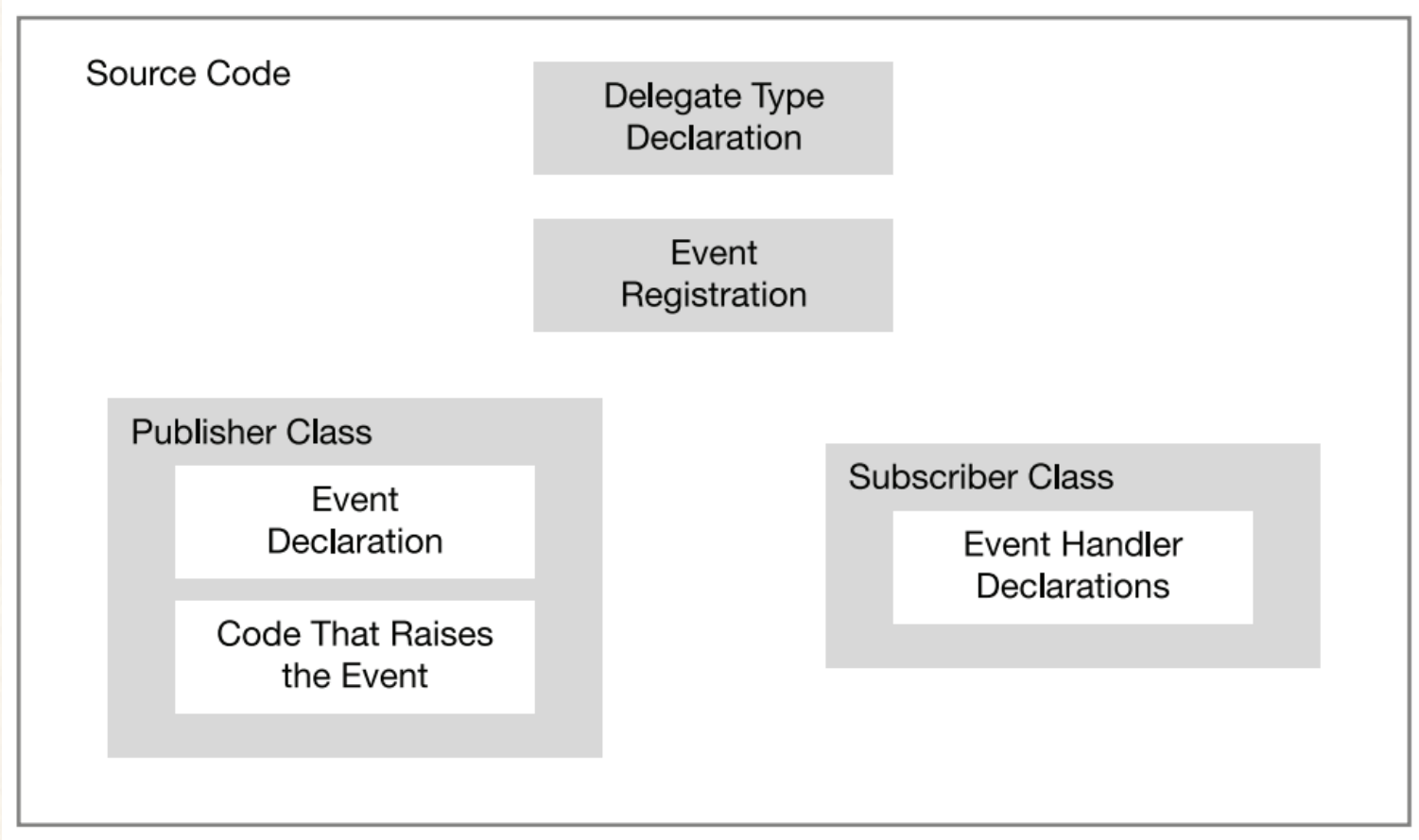


ศัพท์สำคัญ

- Publisher: class หรือ struct ที่เผยแพร่ event ทำให้คลาสอื่นๆ ได้รับการแจ้งเตือนเมื่อมี event เกิดขึ้น
- Subscriber: class หรือ struct ที่ลงทะเบียนเพื่อรับการแจ้งเตือนเมื่อมีเหตุการณ์
- Event handler: method ที่ subscriber ทำการลงทะเบียนไว้กับ publisher และถูกเรียกมาทำงานเมื่อ publisher แจ้งเตือนเหตุ โดยที่ event ต่างๆ สามารถประกาศไว้ในคลาส (หรือ struct) เดียวกันหรือไม่ก็ได้
- Raising an event: เป็นศัพท์สำหรับเรียกการ invoking หรือ firing เหตุการณ์ เมื่อมีเหตุการณ์เกิดขึ้น method ที่ลงทะเบียนไว้ จะถูกเรียกมาทำงาน

การใช้งาน Events

○ การใช้งาน Events เราต้องเขียน source code เป็นจำนวน 5 แห่ง ด้วยกัน



การใช้งาน Events

1. การประกาศ delegate (Delegate type declaration)

   Event และ event handler จะต้องมี signature และ return type เหมือนกัน

2. การประกาศ event handler (Event handler declarations)

   เป็นการประกาศต้นแบบ ของ event handler ใน subscriber class ซึ่งอาจจะมีชื่อหรือไม่มีชื่อของ method ก็ได้ (เช่น anonymous method หรือ lambda expressions)

3. การนิยาม event (Event declaration)

   Publisher class จะต้องประกาศต้นแบบของ event handler ที่สามารถ subscribe เข้ากับ event ได้ ในกรณีที่คลาสใดๆ ประกาศ event ที่เป็น public มันจะต้องเผยแพร่ event ด้วยเสมอ

การใช้งาน Events

4. การลงทะเบียน event (Event registration)

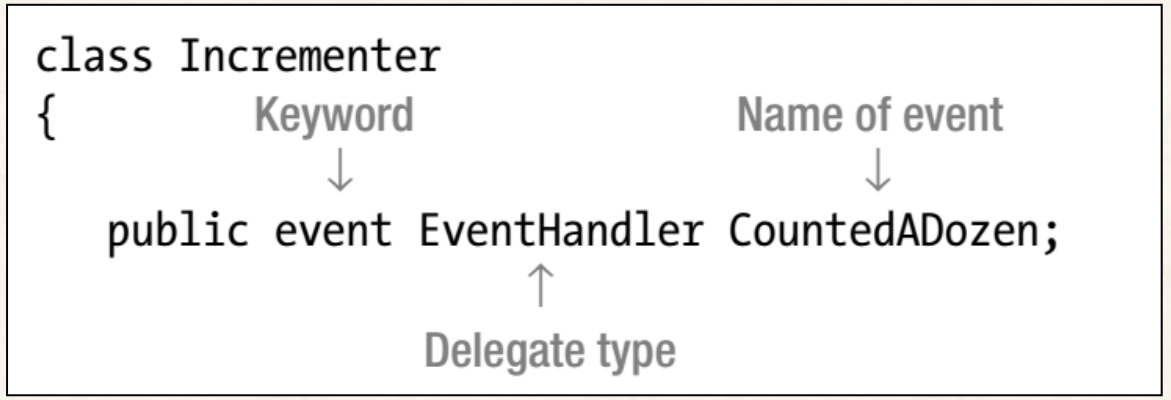
   คลาสที่เป็น subscriber จะต้องทำการลงทะเบียน method ที่จะถูกเรียกเพื่อตอบสนองต่อเหตุการณ์ โค้ดในส่วนนี้จะเป็นตัวเชื่อม event handler เข้ากับ event

5. การเขียนโค้ดที่สร้าง event (Code that raises the event)

   โค้ดในส่วนนี้ อยู่ใน publisher และจะสร้างเหตุการณ์เพื่อทำการ invoke เมธอดที่เป็น event handler ใน subscriber

Declaring an Event

- ในฐานะ publisher จะต้องมีการเตรียม event object
- การสร้าง event ทำได้ง่ายๆ โดยการประกาศชนิดและชื่อของ delegate



ข้อแนะนำ

- event จะต้องประกาศไว้ภายใน class
- จะต้องเป็น delegate type ที่มีชื่อเสมอ
- event handlers ใดๆ ที่เชื่อม (หรือลงทะเบียน) กับ event
  จะต้องมี signature และ return type ที่ตรงกับ delegate นี้
- จะต้องประกาศเป็น public เพื่อให้ classes และ structs อื่นๆ สามารถลงทะเบียน
  event handlers ได้
- ในการสร้าง event นั้น ไม่ต้องสร้างด้วยคีย์เวิร์ด new

การประกาศหลาย event ในครั้งเดียว

- ในการประกาศ event จำนวนหลายๆ ตัวนั้น เราสามารถทำได้โดยการประกาศใน statement เดียว โดยคั่นแต่ละ event ด้วยเครื่องหมาย comma

```
public event EventHandler MyEvent1, MyEvent2, OtherEvent;
                                   ↑
                              Three events
```

- นอกจากนี้ยังสามารถประกาศ event แบบ static

```
public static event EventHandler CountedADozen;
              ↑
          Keyword
```

Event ไม่ใช่ Type แต่เป็น member ของ class

- ดูจากวิธีการประกาศแล้ว อาจจะมีการเข้าใจผิดว่า event มีลักษณะเป็น type เช่นเดียวกับ delegate
- แต่จริงๆ แล้ว event มีฐานะเป็นสมาชิกของ class (หรือ struct)
  - เราไม่สามารถประกาศ event ไว้ในบล็อกของโค้ดที่รันได้
  - จะต้องประกาศไว้ใน class หรือ struct เช่นเดียวกับสมาชิกอื่นๆ
- เมื่อประกาศ event มันจะถูก initial ค่าเป็น null เอาไว้ก่อนเสมอ
- การประกาศ event อาจจะประกาศเชื่อมโยงไว้กับ delegate ที่มีอยู่แล้วก็ได้ แต่ถ้าสร้างใหม่จะต้องตั้งชื่อเป็น delegate type ที่มี signature และ return type ตรงกับ method ที่จะมาลงทะเบียนกับ event นั้นๆ

การลงทะเบียน events

- การลงทะเบียน เป็นการ subscribe ตัว event handler และ event เข้าด้วยกัน
- ทำได้โดยการใช้เครื่องหมาย += ในการเพิ่ม event handler (คล้ายๆ กับการเพิ่ม invocation list ใน delegate)
- Event handler สามารถเป็นอย่างใดอย่างหนึ่งต่อไปนี้
  - ชื่อของ Instance method
  - ชื่อของ Static method
  - Anonymous method
  - Lambda Expression

การลงทะเบียน events

Class instance ↓ Instance method ↓

```
incrementer.CountedADozen += IncrementDozensCount;        // Method reference form
incrementer.CountedADozen += ClassB.CounterHandlerB;      // Method reference form
```

↑ Event member ↑ Static method

```
mc.CountedADozen += new EventHandler(cc.CounterHandlerC);  // Delegate form
```

```
// Lambda expression
incrementer.CountedADozen += () => DozensCount++;

// Anonymous method
incrementer.CountedADozen += delegate { DozensCount++; };
```

การกระตุ้น event (Raising an Event)

- เมื่อสร้าง event และเชื่อมโยงไปยัง event handler แล้ว โปรแกรมของเราก็จะทำงานตามปกติ จะยังไม่มีการทำงานของ event handler แต่อย่างใด
- ขั้นตอนการกระตุ้นให้ event handler ทำงาน
  - ตรวจสอบว่า event นั้น ได้เชื่อมโยงไปยัง event handler ใดๆ หรือไม่ ถ้ามีค่าเป็น null ก็แสดงว่ายังไม่มีปลายทาง ไม่สามารถใช้งานได้
  - ในการเรียกใช้ event ก็ทำเหมือนกับการใช้งานเมธอดธรรมดา นั่นคือใช้ชื่อ event ตามด้วยวงเล็บ
    - ภายในวงเล็บ มี parameter แบบเดียวกับที่ประกาศไว้ใน delegate

การกระตุ้น event (Raising an Event)

```
if (CountedADozen != null)              // Make sure there are methods to execute.
   CountedADozen (source, args);        // Raise the event.
```

Event name — Parameter list

```
class Incrementer
{
   public event EventHandler CountedADozen;   // Declare the event.

   void DoCount(object source, EventArgs args)
   {
      for( int i=1; i < 100; i++ )
         if( i % 12 == 0 )
            if (CountedADozen != null)       // Make sure there are methods to execute.
               CountedADozen(source, args);
   }
}
```

Raise the event.

ตัวอย่างโปรแกรม

```
delegate void Handler();
```
Declare the delegate

Publisher

```
class Incrementer
{
   public event Handler CountedADozen;

   public void DoCount()
   {
      for ( int i=1; i < 100; i++ )
         if ( i % 12 == 0 && CountedADozen != null )
            CountedADozen();
   }
}
```
Create and publish an event.

Raise the event every 12 counts.

ตัวอย่างโปรแกรม (ต่อ)

Subscriber

```
class Dozens
{
   public int DozensCount { get; private set; }

   public Dozens( Incrementer incrementer )
   {
      DozensCount = 0;
      incrementer.CountedADozen += IncrementDozensCount;
   }

   void IncrementDozensCount()
   {
      DozensCount++;
   }
}
```

Subscribe to the event.

Declare the event handler.

ตัวอย่างโปรแกรม (ต่อ)

```
class Program
{
   static void Main( )
   {
      Incrementer incrementer = new Incrementer();
      Dozens dozensCounter    = new Dozens( incrementer );

      incrementer.DoCount();
      Console.WriteLine( "Number of dozens = {0}",
                         dozensCounter.DozensCount );
   }
}
```

Standard Event Usage

- การเขียนโปรแกรมบน GUI จะมีการใช้งาน event อย่างหนัก
- ในขณะที่โปรแกรมกำลังทำงาน จะมีเหตุการณ์ต่างๆ เกิดขึ้นมากมาย
  - จาก GUI เช่น Button, Edit Box, Listbox, ฯลฯ
  - จากอุปกรณ์ต่างๆ เช่น คีย์บอร์ด, เมาส์
  - จากระบบเช่น system timer, Network เป็นต้น
- เราไม่สามารถวนลูปเพื่อตรวจสอบเหตุการณ์ต่างๆ เหล่านั้นได้เลย
- ใน C# ซึ่งรันบน .NET Framework จะมี namespace ที่ชื่อ System ได้บรรจุเอา publisher ของเหตุการณ์ต่างๆ เอาไว้ โดยสร้างเป็น delegate type ที่ชื่อ EventHandler

EventHandler

- Event Handler ใน .NET Framework มี prototype ดังนี้

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

- พารามิเตอร์ตัวแรก เก็บ reference ของ object ที่เป็นผู้กระตุ้น event
- พารามิเตอร์ตัวที่สอง เก็บข่าวสารต่างๆ ที่เหมาะสม
  - โดยปกติ พารามิเตอร์ตัวนี้ มีชนิดเป็น EventArgs และไม่ได้ถูกออกแบบมาให้ส่งผ่านข้อมูลใดๆ
  - หากต้องการส่งผ่านข้อมูลผ่านพารามิเตอร์นี้ ให้ทำการสืบทอดคลาสจาก EventArgs แล้วเพิ่ม fields สำหรับเก็บข้อมูลเอาเอง
- ชนิดของ return type เป็น void

EventHandler

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

- พารามิเตอร์ตัวที่สอง เป็นตัวสำคัญ ที่จะส่งผ่านข้อมูลจาก object ที่เป็นผู้สร้างเหตุการณ์
- ผู้ออกแบบ framework เห็นว่า EventHandler ควรมีรูปแบบมาตรฐาน จึงกำหนดให้มีพารามิเตอร์แค่เพียงสองตัว และทิ้งให้เป็นภาระของนักเขียนโปรแกรม ที่จะหาวิธีส่งผ่านข้อมูลเอาเอง โดยการสืบทอดคลาสจาก EventArgs.
- ดังนั้น หากเขียนโปรแกรมโดยใช้ EventHandler มาตรฐาน เราก็ไม่ต้องประกาศ delegate ของ event handler อีก

ตัวอย่างโปรแกรมที่ใช้ EventHandler

Publisher

```
class Incrementer
{
   public event EventHandler CountedADozen;

   public void DoCount()
   {
      for ( int i=1; i < 100; i++ )
         if ( i % 12 == 0 && CountedADozen != null )
            CountedADozen(this, null);
   }
}
```

Use the system-defined `EventHandler` delegate.

Use `EventHandler`'s parameters when raising the event.

ตัวอย่างโปรแกรมที่ใช้ EventHandler (ต่อ)

Subscriber

```
class Dozens
{
   public int DozensCount { get; private set; }

   public Dozens( Incrementer incrementer )
   {
      DozensCount = 0;
      incrementer.CountedADozen += IncrementDozensCount;
   }

   void IncrementDozensCount(object source, EventArgs e)
   {
      DozensCount++;
   }
}
```

The signature of the event handler must match that of the delegate.

ตัวอย่างโปรแกรมที่ใช้ EventHandler (ต่อ)

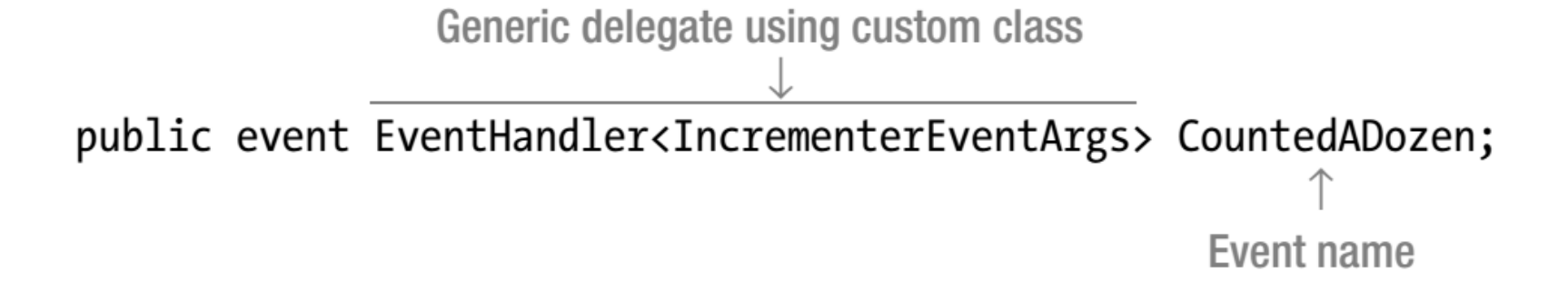
```
class Program
{
   static void Main( )
   {
      Incrementer incrementer = new Incrementer();
      Dozens dozensCounter    = new Dozens( incrementer );

      incrementer.DoCount();
      Console.WriteLine( "Number of dozens = {0}",
                         dozensCounter.DozensCount );
   }
}
```

การส่งผ่านข้อมูลโดยการขยายความสามารถของ EventArgs

- เพื่อให้สามารถส่งข้อมูลผ่าน event ได้ เราต้องส่งผ่านพารามิเตอร์ตัวที่สอง แต่ต้องสร้างคลาสเอง ซึ่งคลาสนั้นต้องสืบทอดมาจาก EventArgs
- ชื่อคลาส ควรที่จะจบด้วย EventArgs เพื่อให้เข้าใจตรงกันว่าสืบทอดมาจาก EventArgs โดยชื่อด้านหน้า ก็คือความสามารถที่เพิ่มเข้ามา

```
                    Custom class name          Base class
                            ↓                      ↓
public class IncrementerEventArgs : EventArgs
{
   public int IterationCount { get; set; }  // Stores an integer
}
```

การใช้งาน ต้องทำผ่านกระบวนการ generic



ตัวอย่างโปรแกรม

```
public class IncrementerEventArgs : EventArgs    // Custom class derived from EventArgs
{
   public int IterationCount { get; set; }       // Stores an integer
}
```

```
class Incrementer
{
   public event EventHandler<IncrementerEventArgs> CountedADozen;

   public void DoCount()
   {
      IncrementerEventArgs args = new IncrementerEventArgs();
      for ( int i=1; i < 100; i++ )
         if ( i % 12 == 0 && CountedADozen != null )
         {
            args.IterationCount = i;

            CountedADozen( this, args );
         }
   }
}
```

Generic delegate using custom class

Object of custom class

Pass parameters when raising the event

ตัวอย่างโปรแกรม

```
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++;
   }
}
```

ตัวอย่างโปรแกรม

```
class Program
{
   static void Main()
   {
      Incrementer incrementer = new Incrementer();
      Dozens dozensCounter    = new Dozens( incrementer );

      incrementer.DoCount();
      Console.WriteLine( "Number of dozens = {0}",
                         dozensCounter.DozensCount );
   }
}
```