События C# по-человечески

Невозможно, просто взять и вникнуть в этот глубокий смысл, изучая События (event) в просторах базового и, на первый взгляд, бесконечного C#.

Когда я изучал События (не в рамках .NET!), потратил много сил, чтобы, наконец-то, разобраться, как они устроены и должны конструироваться. Поэтому, я решил опубликовать свою методику понимания структуры **пользовательского события**, коим представляется ключевое слово **event** в C#.
Не буду цитировать и без того замученную MSDN, а постараюсь объяснить понятно и доступно.

Что требуется Вам для изучения:

- 1. Вы не должны испытывать страх перед изучением. Пожалуйста, читайте медленно и вдумчиво.
- 2. Вы должны понимать классы и методы.
- 3. Вы должны ~~знать~~ понимать, что есть делегаты. Хотя, Вы можете попытаться понять их в ходе чтения статьи.

Событие, это не что иное, как ситуация, при возникновении которой, произойдет действие или несколько действий. Говоря языком программного моделирования, *Событие* — это именованный делегат, при вызове которого, будут запущены все подписавшиеся на момент

вызова события методы заданной сигнатуры. Эта трактовка хоть и раскрывает всю суть структуры события, но не только сбивает с толку начинающих «шарп-прогеров», но и не дает возможность рационально представить в программистской голове весь смысл.

Итак, *Событие*, это ситуация, при возникновении которой, произойдут некоторые действия. Само событие имеет определенную структуру.

Предположим, что стоит такая задача: определено три класса. Первый класс будет считать до 100, используя цикл. Два других класса будут ждать, когда в первом классе счетчик досчитает, например, до 71, и после этого каждый выведет в консоль фразу «Пора действовать, ведь уже 71!». Проще говоря, при обнаружении значения 71, вызовутся по методу, соответственно для каждого класса. Разложим все по полкам.

1. Моделирование ситуации.

Подготовим эти три простейших класса, оставив точку входа в программу **main** нетронутой. Класс **ClassCounter** и его метод *Count()* в котором будет производится счет. (В коде я опускаю пространства имен *namespace*, ибо это ясно, как день).

```
    class ClassCounter  //Это класс - в котором производится счет.
    {
        public void Count()
        {
            //Здесь будет производиться счет
        }
    }
```

Два других класса (имена им **Handler_I** и **Handler_II**), которые должны реагировать на возникновение события методами public void *Message()*. У каждого по методу, как и договаривались.

```
    class Handler_I //Это класс, реагирующий на событие (счет равен 71) записью
строки в консоли.
    {
        public void Message()
        {
            //Не забудьте using System
            //для вывода в консольном приложении
            Console.WriteLine("Пора действовать, ведь уже 71!");
        }
```

```
    }
```

```
    class Handler_II
    {
        public void Message()
        {
            Console.WriteLine("Точно, уже 71!");
        }
    }
```

Напомню, когда счетчик будет считать до 100 и достигнет 71, должны сработать методы Message() для классов Handler_I и Handler_II.
Теперь вернемся к классу ClassCounter и создадим счетчик при помощи цикла **for** с переменной-счетчиком **int i**.

```
    class ClassCounter  //Это класс - в котором производится счет.
    {
        public void Count()
        {
            for (int i = 0; i < 100; i++)
            {
            }
        }
    }
```

Первый этап завершен. У нас есть класс счетчик и два класса, которые будут выводить сообщения. Условия задачи: как только **i=71**, должны сработать методы *Message()* для двух классов **Handler_I** и **Handler_II**.

2. Оформление события.

Абстрагируемся от программирования. Событие, которое мы хотим создать, будет представлять фразу "… счетчик считает. И как только он будет равен 71, должны выполниться действия". Значит, нам необходимо условие «как только он будет равен 71». Представим его при помощи **условного оператора if**.

```
    class ClassCounter  //Это класс - в котором производится счет.
    {
        public void Count()
```

```
        {
            for (int i = 0; i < 100; i++)
            {
                if (i == 71)
                {
                }
            }
        }
    }
```

Конструируем событие **event**. Определяем по методам, которые должны сработать при i=71 их **сигнатуру** (или прототип).
Сигнатура метода — это так называемая спецификация (или простыми словами «шаблон») какого-л. метода или методов. Представляет собой сочетание названия типа, который метод возвращает, плюс название типов входящих параметров (по порядку! порядок очень важен.) Например, метод *int NewMethod(int x, char y)* будет иметь сигнатуру **int (int, char)**, а метод *void NewMethod()* — **void (void)**.
Как толкует MSDN, события (event) основаны на делегатах (delegate), а делегат, говоря очень простым языком — «переменная, хранящая ссылку на метод». Как Вы уже поняли, т.к. наше событие будет ссылаться на два метода void Message(), мы должны определить сигнатуру этих методов, и составить на основе этой сигнатуры делегат. Сигнатура выглядит так: **void (void)**.

Определяем делегат (назовем его MethodContainer):

```
    class ClassCounter  //Это класс - в котором производится счет.
    {
        //Синтаксис по сигнатуре метода, на который мы создаем делегат:
        //delegate <выходной тип> ИмяДелегата(<тип входных параметров>);
        //Мы создаем на void Message(). Он должен запуститься, когда условие
выполнится.

        public delegate void MethodContainer();

        public void Count()
        {
            for (int i = 0; i < 100; i++)
            {
                if (i == 71)
                {
                }
            }
        }
```

```
    }
```

Далее, мы создаем событие при помощи ключевого слова **event** и связываем его с этим делегатом (**MethodContainer**), а, следовательно, с методами, имеющими сигнатуру *void (void)*. Событие должно быть public, т.к. его должны использовать разные классы, которым нужно как-то отреагировать (классы Handler_I и Handler_II).
Событие имеет синтаксис: public event <НазваниеДелегата> <НазваниеСобытия>;
Название делегата — это имя делегата, на который «ссылаются» наши методы.

```
class ClassCounter  //Это класс - в котором производится счет.
   {
       public delegate void MethodContainer();

       //Событие OnCount с типом делегата MethodContainer.
       public event MethodContainer onCount;

       public void Count()
       {
           for (int i = 0; i < 100; i++)
           {
               if (i == 71)
               {
               }
           }
       }
   }
```

Теперь запустим наше событие onCount, в условии когда i=71:

```
if (i == 71)
{
     onCount();
}
```

Все. **Событие создано**. Методы, которые вызовет это событие, определены по сигнатурам и на основе их создан делегат. Событие, в свою очередь, создано на основе делегата. Пора показать событию *onCount*, какие же все-таки методы должны сработать (мы ведь указали только их сигнатуру).

3. Подписка.

Вернемся в точку входа программы main и создадим экземпляр класса **ClassCounter**. А также создадим по экземпляру классов, которые должны запуститься. (Они должны быть *public*).

```
class Program
{
    static void Main(string[] args)
    {
        ClassCounter Counter = new ClassCounter();
        Handler_I Handler1 = new Handler_I();
        Handler_II Handler2 = new Handler_II();
    }
}
```

Теперь укажем событию *onCount*, методы, которые должны запуститься.
Происходит это следующим образом: **<КлассИлиОбъект>.<ИмяСобытия> += <КлассЧейМетодДолженЗапуститься>.<МетодПодходящийПоСигнатуре>**.
Никаких скобочек после метода! Мы же не вызываем его, а просто указываем его название.

```
class Program
{
    static void Main(string[] args)
    {
        ClassCounter Counter = new ClassCounter();
        Handler_I Handler1 = new Handler_I();
        Handler_II Handler2 = new Handler_II();

        //Подписались на событие
        Counter.onCount += Handler1.Message;
        Counter.onCount += Handler2.Message;
    }
}
```

Проверка.

Теперь осталось запустить счетчик класса *ClassCounter* и подождать, пока i станет равным 71. Как только i=71, запустится событие *onCount* по делегату *MethodContainer*, который (в свою очередь) запустит методы *Message()*, которые были *подписаны* на событие.

```
class Program
```

```
    {
        static void Main(string[] args)
        {
            ClassCounter Counter = new ClassCounter();
            Handler_I Handler1 = new Handler_I();
            Handler_II Handler2 = new Handler_II();

            Counter.onCount += Handler1.Message;
            Counter.onCount += Handler2.Message;

            //Запустили счетчик
            Counter.Count();
        }
    }
```

Результат:
Пора действовать, ведь уже 71!
Точно, уже 71!

Заключение.

Постарайтесь понять смысл и порядок создания события.

- 1. Определите условие возникновения события и методы которые должны сработать.
- 2. Определите сигнатуру этих методов и создайте делегат на основе этой сигнатуры.
- 3. Создайте общедоступное событие на основе этого делегата и вызовите, когда условие сработает.
- 4. Обязательно (где-угодно) подпишитесь на это событие теми методами, которые должны сработать и сигнатуры которых подходят к делегату.

Класс, в котором вы создаете событие (генерируете) называется классом-издателем, а классы, чьи методы подписываются на это событие при помощи **"+="** — классами-подписчиками.

Запомните! Если Вы не подписались на событие и его делегат пустой, возникнет ошибка. Чтобы избежать этого, необходимо подписаться, или не вызывать событие вообще, как показано на примере (Т.к. событие — делегат, то его отсутствие является «нулевой ссылкой» **null**).

```
            if (i == 71)
            {
                if (onCount != null)
                {
```

```
                    onCount();
                }
            }
```

Вы всегда можете отписаться, используя оператор "**-=**": <КлассИлиОбъект>.<ИмяСобытия> -= <КлассЧейМетодДолженЗапуститься>.<МетодПодходящийПоСигнатуре>.

Преимущество Событий очевидно: *классу-издателю, генерирующему событие* не нужно знать, сколько *классов-подписчиков подпишется* или отпишется. Он создал событие для определенных методов, ограничив их делегатом по определенной сигнатуре.
События широко используются для составления собственных компонентов управления (кнопок, панелей, и т.д.).

У самых маленьких может возникнуть вопрос: что делать, если методы, которые должны сработать имеют входящий параметр (а то и не один!)?
Ответ: Все дело в делегате, на котором базируется событие. А точнее сигнатура подходящих для делегата методов. Когда Вы сконструируете делегат, «принимающий» метод с параметром, то (!) при запуске событие запросит этот параметр. Естественно, параметр может быть чем угодно.

Пару слов о .NET-событиях. Microsoft упростила задачу конструирования делегатов: .NET предлагает готовый делегат **EventHandler** и т.н. «пакет» входных параметров **EventArgs**. Желаете событие? Берете готовый EventHandler, определяетесь в параметрах, «запихиваете» их в *класс*, а класс наследуете от EventArgs. А дальше — как по расписанию)

Многие разработчики утверждают (и я с ними согласен), что главная проблема «недопонимания» событий — их специфическая область применения, а вследствие — мало доступных примеров. Ну и не забывайте о практике.

P.S. Если вы не ни разу не использовали делегаты, лучше попробуйте потренироваться на делегатах, а затем попытайтесь понять эту статью.
Я надеюсь, что внес небольшое понимание в эту непростую тему. Успехов!