

# C# Essential

События

# C# Essential

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)



Проверьте как Вы усвоили данный материал  
на [TestProvider.com](http://testprovider.com)

# C# Essential

Тема

События

# События

## Событийно-ориентированное программирование

**Событийно-ориентированное программирование (event-driven programming)** – парадигма программирования, в которой выполнение программы определяется событиями – действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).

**События** – это особый тип многоадресных делегатов, которые можно вызвать только из класса или структуры, в которой они объявлены (класс издателя). Если на событие подписаны другие классы или структуры, их методы обработчиков событий будут вызваны, когда класс издателя инициирует событие.



События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций.

# События

## Применение событий

**Событийно-ориентированное программирование, как правило, применяется в трех случаях:**

- При построении пользовательских интерфейсов (в том числе графических);
- При создании серверных приложений в случае, если по тем или иным причинам нежелательно порождение обслуживающих процессов;
- При программировании игр, в которых осуществляется управление множеством объектов.

# События

## Создание событий

**Чтобы класс мог породить событие, необходимо подготовить три следующих элемента:**

- Класс, предоставляющий данные для события.
- Делегат события.
- Класс, порождающий событие.

# События

## Создание событий

Для объявления события в классе издателя, используется ключевое слово **event**

```
// Метод который вызывает событие.
```

```
instance.InvokeEvent();
```

```
instance.
```

```
Console.WriteLine("g('-', 20));  
  
// Откреп  
instance.EventDelegate(Handler2);  
instance.  
  
// Delay.  
Console.ReadKey();
```

```
public event EventDelegate MyEvent = null;
```

```
// Метод который вызывает событие.
```

```
instance.
```

```
Console.WriteLine("g('-', 20));  
  
// Откреп  
instance.EventDelegate(Handler2);  
instance.  
  
// Delay.  
Console.ReadKey();
```

```
public EventDelegate MyEvent = null;
```



На событии нельзя напрямую вызвать метод `Invoke()`

# События

## Свойства событий

**События имеют следующие свойства:**

- Издатель определяет момент вызова события, подписчики определяют предпринятое ответное действие.
- У события может быть несколько подписчиков. Подписчик может обрабатывать несколько событий от нескольких издателей.
- События, не имеющие подписчиков, никогда не возникают.
- Обычно события используются для оповещения о действиях пользователя, таких как нажатия кнопок или выбор меню и их пунктов в графическом пользовательском интерфейсе.
- Если событие имеет несколько подписчиков, то при его возникновении происходит синхронный вызов обработчиков событий.



# События

## Сигнатура событий

Сигнатура обработчика событий должна соответствовать следующим соглашениям:

```
private void button_Click(object sender, EventArgs e)
{
}
}
```

- Метод обработчик события принимает ровно два параметра.
- Первый параметр называется **sender** и имеет тип **object**. Это объект, вызвавший событие.
- Второй параметр называется – **e** и имеет тип **EventArgs** или тип производного класса от **EventArgs**. Это данные, специфичные для события.
- Тип возвращаемого значения метода обработчика – **void**.

# События

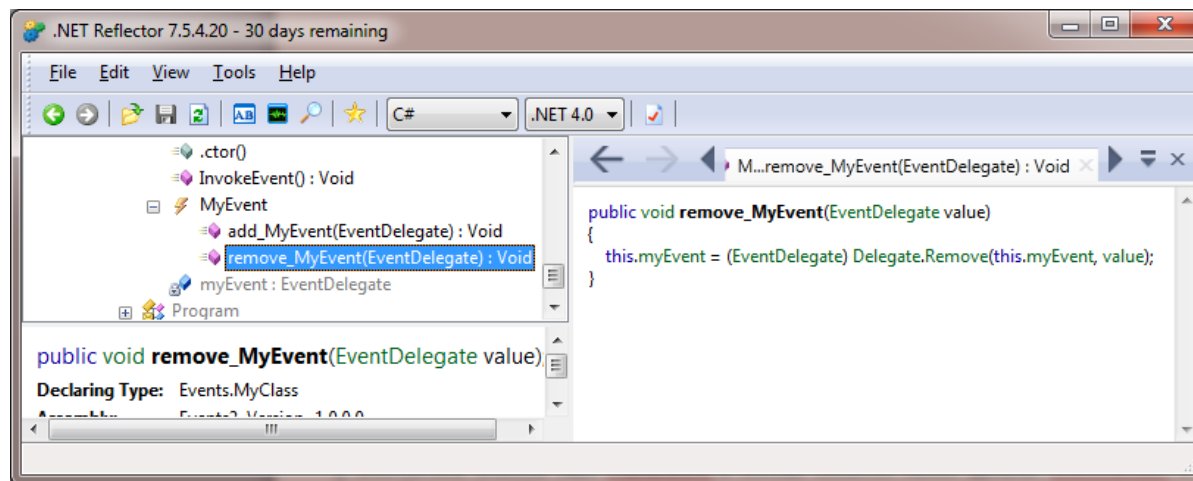
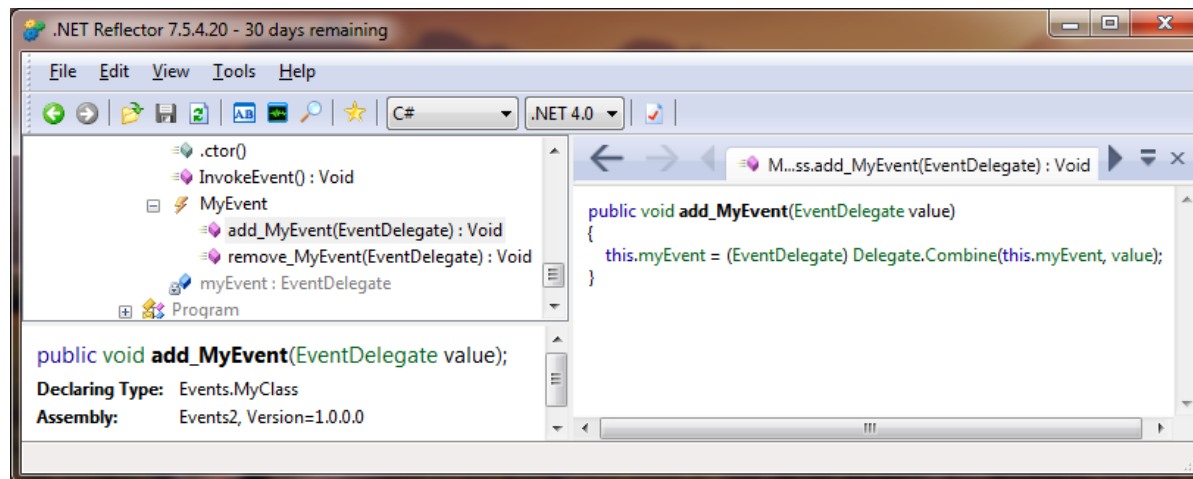
## Add-Remove

- Контекстно-зависимое ключевое слово **add** используется для определения пользовательского метода доступа к событию, вызываемому при подписке клиентского кода к событию. Если указан пользовательский метод доступа **add**, то необходимо также указать метод доступа **remove**.

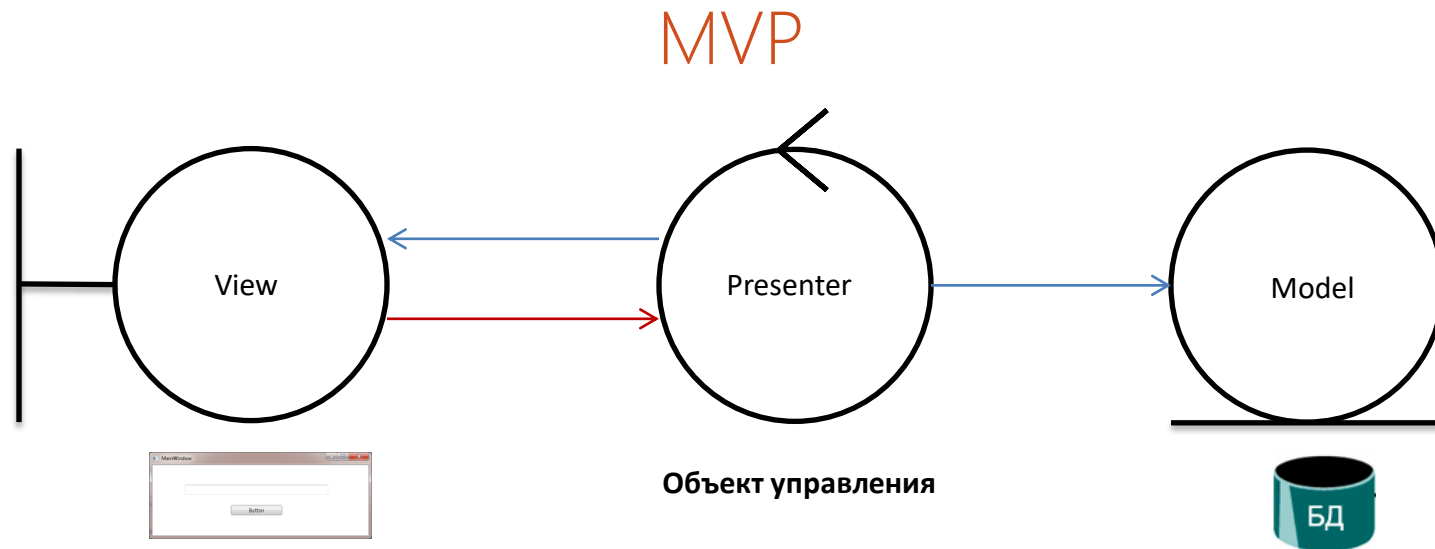
```
public event EventDelegate MyEvent
{
    add { myEvent += value; }
    remove { myEvent -= value; }
}
```

# События

## Методы доступа add и remove под .Net Reflector



# Model-View-Presenter



**MVP** — шаблон проектирования пользовательского интерфейса, который был разработан для облегчения автоматического модульного тестирования и улучшения разделения ответственности в презентационной логике

**Модель (model)** представляет собой интерфейс, определяющий данные для отображения или участвующие в пользовательском интерфейсе иным образом

**Вид (view)** — это интерфейс, который отображает данные (модель) и маршрутизирует пользовательские команды (или события) Presenter-у, чтобы тот действовал над этими данными.

**Presenter** действует над моделью и видом. Он извлекает данные из хранилища (модели), и форматирует их для отображения в Виде (view).

# C# Essential

Q&A

# Информационный видеосервис для разработчиков программного обеспечения

