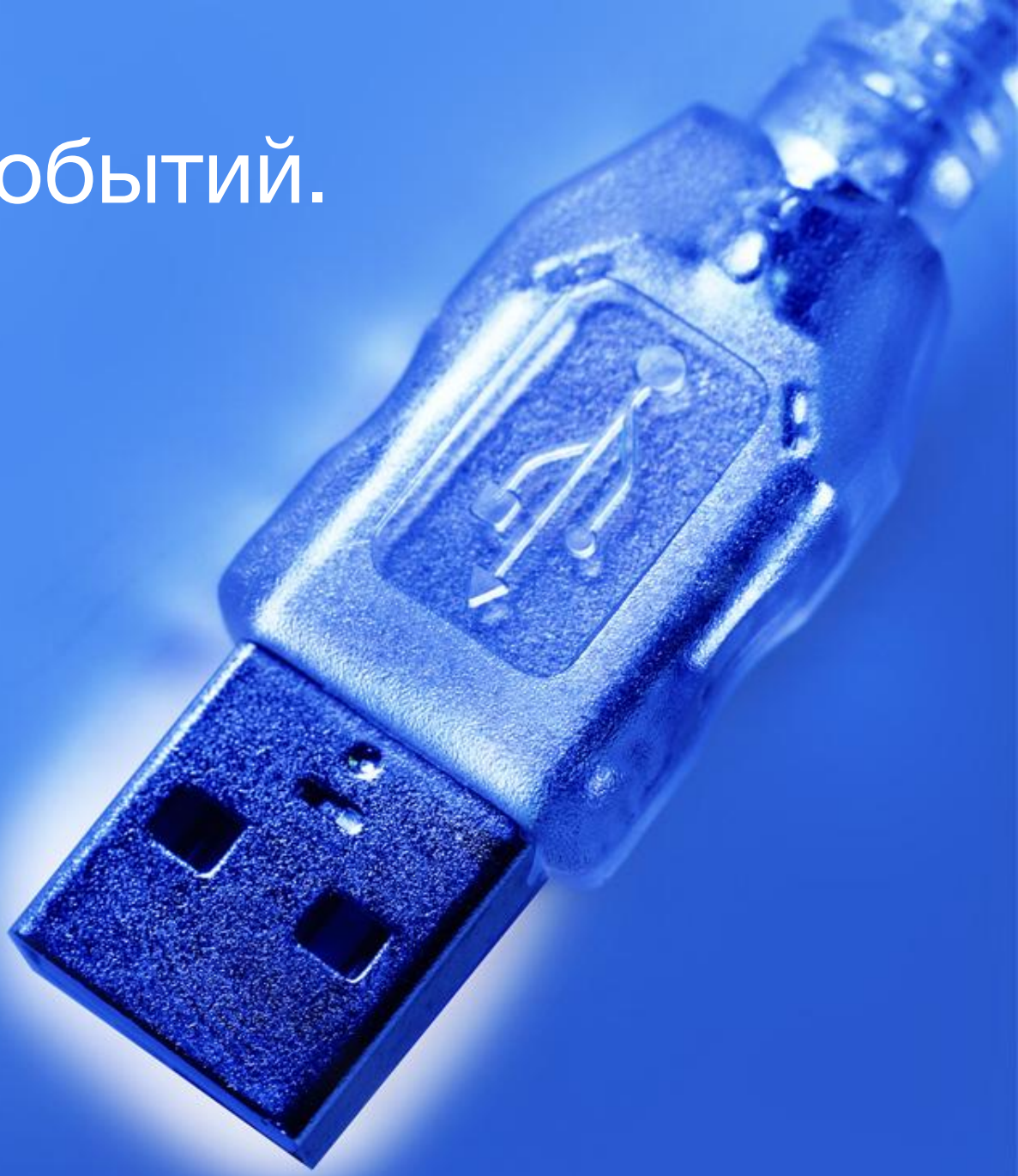
Механизм событий.

лекция №16



События

- События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций. Класс, отправляющий (или *вызывающий*) событие, называется *издателем*, а классы, принимающие (или *обрабатывающие*) событие, называются *подписчиками*.

Описание события

- **[атрибуты] [спецификаторы] event тип имя обытия**

Описание события

Keyword	Описание
static	Делает событие доступным для вызова в любое время, даже если экземпляр класса отсутствует.
virtual	Позволяет производным классам переопределять поведение события.
sealed	Указывает, что для производных классов событие более не является виртуальным.
abstract	Компилятор не создаст блоки методов доступа к событиям **add** и **remove** и, следовательно, производные классы должны предоставлять собственную реализацию.

Процесс создания события

Создание события в классе состоит из следующих **частей:**

- **описание делегата, задающего сигнатуру обработчиков событий;**
- **описание события;**
- **описание метода (методов), инициирующих событие.**

Пример события

- **public delegate void Del(object o); // объявление делегата**
- **class A**
- **{**
- **public event Del Oops; // объявление события**
- **}**

Важно помнить

- **Событие — это удобная абстракция для программиста. На самом деле оно состоит из закрытого статического класса, в котором создается экземпляр делегата, и двух методов, предназначенных для добавления и удаления обработчика из списка этого делегата.**

Подписка на событие

- **Внешний код может работать с событиями единственным образом: добавлять обработчики(+=) в список или удалять их(-=).**

Паттерн наблюдатель

Через события

```
public event Del OOps;
    ссылок 0
    public void OOPS()
    {// что-то произошло
        Console.WriteLine( "OOPS!" );
        if (OOps != null) OOps(this); //
    }
```

Через делегаты

```
Del dels; // объявление экземпляра дел
ссылок 3
public void Register( Del d ) // регис
{dels += d;}
ссылка 1
public void OOPS()
{// что-то произошло
    Console.WriteLine( "OOPS!" );
    if ( dels != null ) dels( this );
}
```

Паттерн наблюдатель

Через события

```
Subj s = new Subj();
ObsA o1 = new ObsA();
ObsA o2 = new ObsA();

s.OOps+=o1.Do;
s.
```

- Equals
- GetHashCode
- GetType
- OOps — Del Subj.OOps
- OOPS
- ToString

Через делегаты

```
Subj s = new Subj();
ObsA o1 = new ObsA()
ObsA o2 = new ObsA()

s.Register(o1.Do);
s.Register(o2.Do);
s.Register(ObsB.See
```

Полная форма события

Имя делегата заканчивается суффиксом EventHandler;

делегат получает два параметра:

- **О первый параметр задает источник события и имеет тип object;**
- **О второй параметр задает аргументы события и имеет тип EventArgs или производный от него.**

Полная форма события

```
class Subj
{// класс-источник
    public event EventHandler OOps;
        ссылка 1
        public void OOPS()
        {// что-то произошло
            Console.WriteLine( "OOPS!" );
            if (OOps != null) OOps(this,null);
        }
}
```

```
class ObsA
{// класс-наблюдатель
    ссылок 2
    public void Do( object sender, EventArgs e ) // pea
    {       Console.WriteLine( "Вижу, что OOPS!" );}
}
ссылка 1
static class ObsB
{// класс-наблюдатель
    ссылка 1
    public static void See(object sender, EventArgs e)
    {
        Console.WriteLine( "Я тоже вижу, что OOPS!" );
    }
```

```
Subj s = new Subj();
ObsA o1 = new ObsA();
ObsA o2 = new ObsA();

s.OOps += new EventHandler(o1.Do);
s.OOps += new EventHandler(o2.Do);
s.OOps += new EventHandler(ObsB.See);

s.OOPS();
```

```
OOPS!
Вижу, что OOPS!
Вижу, что OOPS!
Я тоже вижу, что OOPS!
Для продолжения нажмите любую клавишу . . .
```

Анонимный обработчик

```
s.OOps += new EventHandler(o1.Do);
s.OOps += new EventHandler(o2.Do);
s.OOps += new EventHandler(ObsB.See);
s.OOps += delegate(object sender, EventArgs e)
{ Console.WriteLine("Я с вами!"); };
s.OOPS();
```

```
OOPS!
Вижу, что OOPS!
Вижу, что OOPS!
Я тоже вижу, что OOPS!
Я с вами!
Для продолжения нажмите любую клавишу . . .
```

Аргументы события

- EventArgs служит в качестве базового класса, от которого получается производный класс, содержащий все необходимые поля для обработчика событий.

Аргументы события

- **class MyEventArgs : EventArgs**
- { **public char** ch; }

Аргументы события

```
class KeyEvent {
// Создадим событие, используя обобщенный
    делегат
public
    event EventHandler<MyEventArgs> KeyDown;
public void OnKeyDown(char ch) {
MyEventArgs c = new MyEventArgs();
if (KeyDown != null)
{ c.ch = ch; KeyDown(this, c); } } }
```

Аксессоры событий

- *event делегат_события имя_события {*
- *add { // Код добавления события в цепочку событий }*
- *remove { // Код удаления события из цепочки событий } }*

Аксессоры событий

- Длинная нотация для определения событий удобна, если необходимо сделать нечто большее, чем просто добавлять и удалять обработчики событий, например, добавить синхронизацию для многопоточного доступа.

Домашнее задание

- Учебник Павловской
- Повторить 1-10 главу