Лекция № 3

ITEA

IT Education Academy

WWW.ITEA.UA

C# Base

Урок № 3

Делегаты и События

План урока

- Понятие делегата
- Объявление делегатов
- Свойства делегатов
- Комбинированные (групповые) делегаты.
- Анонимные методы
- Лямбда операторы и Лямбда-выражения
- События
- Применение событий
- Создание событий
- Свойства событий

Понятие делегата

- **Делегат(delegate)**—это разновидность объектов которые содержат в себе указатели на методы.

- По сути **делегат** представляет собой объект, который может ссылаться на метод. Следовательно, когда создается делегат, то в итоге получается объект, содержащий ссылку на метод. Более того, метод можно вызывать по этой ссылке. Иными словами, делегат позволяет вызывать метод, на который он ссылается.

- Все **делегаты**, являются производными от абстрактного класса System.MulticastDelegate, который в свою очередь наследуется от абстрактного класса Delegate.

Понятие делегата

- Тип **делегата** объявляется с помощью ключевого слова delegate.
- Общая форма объявления делегата:

delegate возвращаемый_тип имя (список_параметров);

```
public delegate void Mydelegete();
```

- **возвращаемый_тип** обозначает тип значения, возвращаемого методами, которые будут вызываться делегатом;
- **имя** — конкретное имя делегата;
- **список_параметров** — параметры, необходимые для методов, вызываемых делегатом

Как только будет создан экземпляр делегата, он может вызывать и ссылаться на те методы, возвращаемый тип и параметры которых соответствуют указанным в объявлении делегата.

Объявление делегатов

Делегат может служить для вызова любого метода с соответствующей сигнатурой и возвращаемым типом. Более того, вызываемый метод может быть методом экземпляра, связанным с отдельным объектом, или же статическим методом, связанным с конкретным классом.

```
public delegate void Mydelegete();

static void Main(string[] args)
{
   Mydelegete myDelegate = new Mydelegete(MyClass.Method);
   myDelegate.Invoke();
   myDelegate();
}

static class MyClass
{
   public static void Method(){}
}
```

создание делегата, делегат не имеет параметров и не возвращает значение

создание экземпляра делегата и присвоение этой переменной адреса метода

вызов метода

пользовательский класс

Объявление делегатов

Делегат ссылается на метод и после назначения метода ведёт себя идентично ему. Делегат можно использовать как любую функцию с параметром и возвращаемым значением».

```
public delegate int Mydelegete(int x, int y);
```

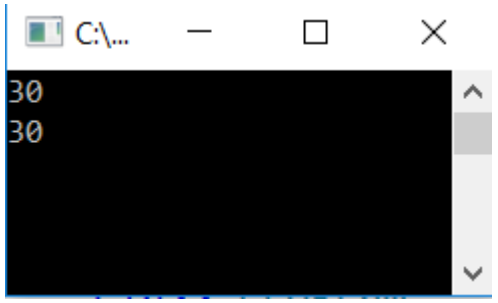
← делегат `Mydelegete` возвращает значение типа int и имеет два параметра типа int

```
static void Main(string[] args)
{
    Mydelegete myDelegate = new Mydelegete(MyClass.Add);
    myDelegate.Invoke(10, 20);
    myDelegate(10, 20);
}
```

← создание экземпляра делегата и присвоение этой переменной адреса метода

} вызов метода

```
static class MyClass
{
    public static int Add(int x, int y)
    {
        return x + y;
    }
}
```



Свойства делегатов

Делегаты C# обладают следующими свойствами:

- позволяют обрабатывать методы в качестве аргумента;
- могут быть связаны вместе;
- несколько методов могут быть вызваны по одному событию;
- тип делегата определяется его именем;
- не зависят от класса объекта, на который ссылается;
- сигнатура метода должна совпадать с сигнатурой делегата.

Комбинированные делегаты

Делегат может указывать на множество методов, которые имеют ту же сигнатуру и возвращаемые тип. Все методы в делегате попадают в специальный список - список вызова или invocation list. И при вызове делегата все методы из этого списка последовательно вызываются. И мы можем добавлять в этот список не один, а несколько методов.

Операция += используется для добавления делегатов.

В реальности будет происходить создание нового объекта делегата, который получит методы старой копии делегата и новый метод, и новый созданный объект.

Операция -= используется для удаления делегатов.

При удалении методов из делегата фактически будет создаваться новый делегат. Если делегат содержит несколько ссылок на один и тот же метод, то операция -= начинает поиск с конца списка вызова делегата и удаляет только первое найденное вхождение

Комбинированные делегаты

Пример комбинированных делегатов

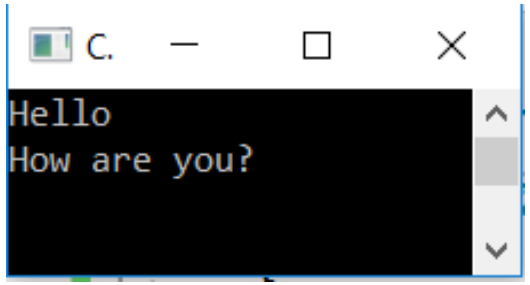
```
class Program
{
    delegate void Message();

    static void Main(string[] args)
    {
        Message message = new Message(HowAreYou);
        message += HowAreYou;
        message.Invoke();
    }
    private static void Hello()
    {
        Console.WriteLine("Hello");
    }
    private static void HowAreYou()
    {
        Console.WriteLine("How are you?");
    }
}
```

Объявление делегата

Создаем переменную делегата и присваиваем адреса метода через конструктор.

присваиваем адреса второго метода

вызов методов



Анонимные функции

Анонимная функция — это "встроенный" оператор или выражение, которое может использоваться, когда тип делегата неизвестен. Ее можно использовать для инициализации именованного делегата или передать вместо типа именованного делегата в качестве параметра метода.

Анонимная функция – это функция, которая не имеет имени а содержит только блок программного кода, который она выполняет.

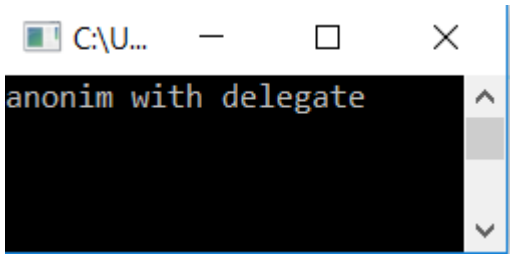
Анонимные функции удобно использовать в объединении с **делегатами**. Анонимную функцию можно вызвать только с помощью делегата. Сама функция непосредственно не вызовется никогда.

```
delegate void Anonim();
```

Объявление делегата

```
static void Main(string[] args)
{
    Anonim anonim = delegate { Console.WriteLine("anonim with delegate"); };
    anonim();
}
```

используем анонимный метод



Лямбда-операторы

Во всех лямбда-выражениях используется лямбда - оператор **=>**, который читается как "переходит в". Левая часть **лямбда – оператора** определяет параметры ввода (если таковые имеются), а правая часть содержит выражение или блок оператора. **Лямбда – выражение** x=>x*x читается как "x переходит в x,x раз".

Ламбда - операторы имеют следующий синтаксис:

(input-parameters) => expression или (список_параметров) => выражение.

```
delegate int Operation(int x, int y);

static void Main()
{
   Operation operation = (x, y) => x + y;
   int result = operation(10, 20);
}
```

Лямбда-выражение

Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы.

Ламбда - выражения имеют следующий синтаксис:

(input-parameters) => {expression} или (список_параметров) => {выражение}

```
delegate int Operation(int x, int y);

static void Main()
{
   Operation operation = (x, y) => { return x + y; };
   int result = operation(10, 20);
}
```

Анонимные методы

С делегатами тесно связаны **анонимные методы**. Анонимные методы используются для создания экземпляров делегатов.

Определение анонимных методов начинается с ключевого слова delegate, после которого идет в скобках список параметров и тело метода в фигурных скобках:

```
delegate(параметры)
{
  // инструкции
}
```

```
delegate void Message(string message);
static void Main()
{
   Message message = delegate (string mes){Console.WriteLine(mes);};
   message.Invoke("Hello world!");
}
```

Техника предположения делегатов

- Техника предположения делегата используется при написании анонимных методов и привязки их к делегату.
- Что б написать анонимный метод, понадобится указать ключевое слово delegate, затем объявить параметры, а потом - тело самого анонимного метода.
- Техника предположения делегата используется как при анонимных методах, так и при лямбда-выражениях.

```
//Mydelegate mydelegate = new Mydelegate(MyClass.Method);
Mydelegate mydelegate = MyClass.Method;
mydelegate("Hellow World!");
```

Правила использования делегатов

Следующие правила применимы к области действия переменной в лямбда-выражениях.

- Захваченная переменная не будет уничтожена сборщиком мусора до тех пор, пока делегат, который на нее ссылается, не выйдет за границы области.
- Переменная, введенная в лямбда-выражение, невидима во внешнем методе.
- Лямбда - выражение не может непосредственно захватывать параметры ref или out из включающего их метода.
- Лямбда – выражение не может содержать оператор goto, оператор break или оператор continue, для которых, метка перехода находится вне тела либо в теле содержащейся анонимной функции.

События

Событийно-ориентированное программирование

Событийно – ориентированное программирование (event – driven programming) — парадигма программирования, в которой выполнение программы определяется событиями —действиями пользователя (клавиатура, мышь), сообщениями других программи 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.

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.

Создание событий

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

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

События являются членами класса и объявляются с помощью ключевого слова event. Чаще всего для этой цели используется следующая форма:

event модификатор_доступа делегат_события имя_события;

```
public event Mydelegate MyEvent;
```

Создание событий

```
delegate void Mydelegate ();

static void Main()
{
   EventClass eventClass = new EventClass();
   eventClass.MyEvent += EventClassData.Method;

   eventClass.CallMethod();
}

class EventClass
{
   public event Mydelegate MyEvent;

   public void CallMethod()
   {
      MyEvent.Invoke();
   }
}
```

Делегат события.

Подписка на обработчик события

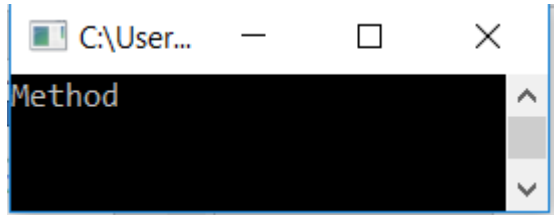
Класс, предоставляющий данные для события.

Определение события

Вызов события

Класс, порождающий событие.

```
class EventClassData
{
   public static void Method()
   {
      Console.WriteLine("Method");
   }
}
```



Свойства событий

- 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.

Добавление и удаление обработчиков

- Для добавления обработчика события применяется операция +=:
- Для одного события можно установить несколько обработчиков;
- Обработчики событий можно удалить в любой момент. Для удаления обработчиков применяется операция -=;
- В качестве обработчиков могут использоваться не только обычные методы, но также делегаты, анонимные методы и лямбда-выражения;

Анонимный метод:

```
eventClass.MyEvent += delegate () { Console.WriteLine("Work"); };
```

Анонимный метод:

```
eventClass.MyEvent += () => { Console.WriteLine("Work"); };
```

Управление обработчиками

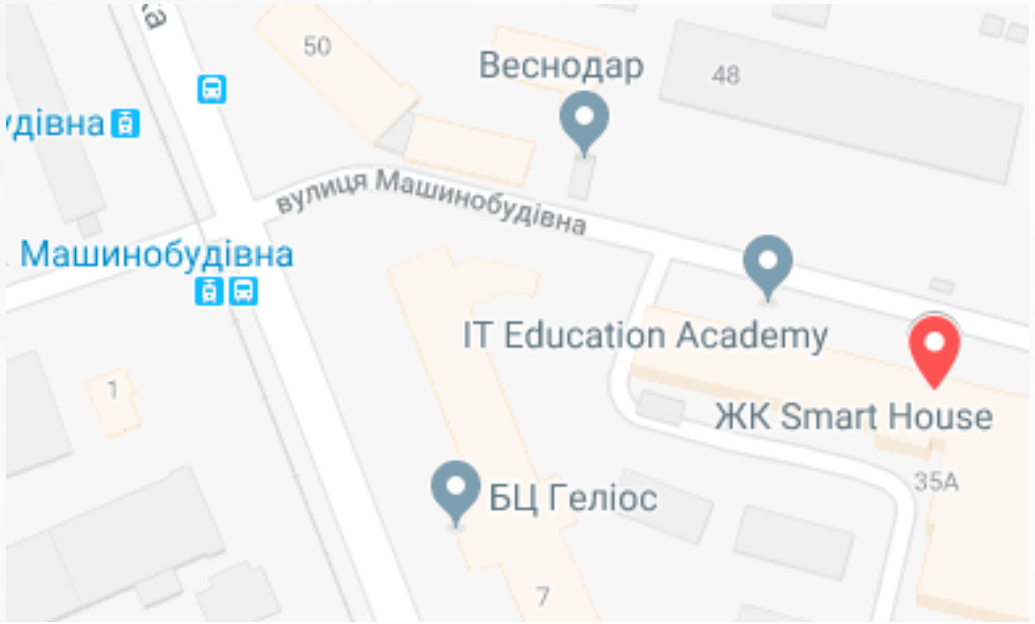
С помощью специальных акссесоров **add/remove** мы можем управлять добавлением и удалением обработчиков. Как правило, подобная функциональность редко требуется, но тем не менее мы ее можем использовать. Если указан пользовательский метод доступа add, то необходимо также указать метод доступа remove.

```
private event Mydelegate myEvent;

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

→ аксессор add вызывается при добавлении обработчика

→ блок remove вызывается при удалении обработчика


Презентация окончена.
Спасибо за внимание!



I T E A

КОНТАКТНЫЕ ДАННЫЕ

ITEA

ЖК “Smart House”, ул. Машиностроительная, 41 (м.Берестейская)

ЖК «Корона» улица Срибнокильская,1 м. Позняки

+38 (044) 599-01-79
facebook.com/Itea
info@itea.ua
itea.ua