

# Лекция 5

# Делегаты. Что это такое?

- Делегат - это тип, который представляет собой **ссылки на методы с определенным списком параметров и возвращаемым типом.**

# Делегаты. Что это такое?

- При создании экземпляра делегата этот экземпляр можно связать с **любым методом с совместимой сигнатурой и возвращаемым типом**.
- Метод можно вызвать (активировать) с помощью экземпляра делегата.

# Делегаты. Что это такое?

- Поскольку созданный экземпляр делегата является **объектом**, его можно **передавать как параметр** или назначать свойству.
- Это позволяет методу принимать делегат в качестве параметра и **вызывать** делегат в дальнейшем.

# Делегаты. Что это такое?

- Эта процедура называется **асинхронным обратным вызовом** и обычно используется для **уведомления** вызывающего объекта о завершении длительной операции.

# Как его создать?

```
<мод. доступа> delegate <возвр. знач.>  
    <имя делегата>(<параметры>);
```

Примеры:

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

```
public delegate void Del(string message);
```

# Пример

- Возьмем делегат:

```
public delegate void Del(string message);
```

- Создадим метод, где его будем использовать

```
public void Method (int param1, Del callback)
{
    callback("The number is: " +
            param1.ToString());
}
```

# Пример

- Делегат может создаваться вне класса, как НОВЫЙ ТИП

```
public delegate void Del(string message);  
0 references  
class Class6  
{  
    0 references  
    public void Method(int param1, Del callback)  
    {  
        callback("The number is: " + param1.ToString());  
    }  
}
```



# Пример

- Теперь создадим функцию, которая будет соответствовать нашему делегату

```
public static void PrintMes(string mes)
{
    Console.WriteLine(mes);
}
```

- Создадим объект от класса и вызовем тот метод

# Пример

- Мы передали функцию PrintMes как параметр в функцию

```
class Program
{
    1 reference
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();
        cl.Method(3, PrintMes);
        Console.ReadKey();
    }
}
```

# Пример. Вариант 2

- А можно и так. Создали переменную от типа нашего делегата

```
class Program
{
    1 reference
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();
        Del deleg = PrintMes;
        cl.Method(3, deleg);
        Console.ReadKey();
    }
}
```

## 2 В 1

- При вызове делегат может вызывать сразу **несколько** методов.
- Это называется **многоадресностью**.
- Чтобы добавить в список методов делегата (список вызова) дополнительный метод, необходимо просто добавить два делегата с помощью оператора сложения или назначения сложения ("+" или "+=").

# 2 В 1

- 2 метода, вызываемые через 1 делегат.
- Методов на делегат можно прикрутить сколько угодно

```
class Program
{
    1 reference
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }
    1 reference
    public static void PrintMes1(string mes)
    {
        Console.WriteLine("Вариант 2: " + mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();
        Del deleg = PrintMes;
        deleg += PrintMes1;
        cl.Method(3, deleg);
        Console.ReadKey();
    }
}
```

# Пример. Вариант 3

- Делегат  
МОЖНО СОЗДАТЬ  
через  
конструктор

```
class Program
{
    1 reference
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }
    1 reference
    public static void PrintMes1(string mes)
    {
        Console.WriteLine("Вариант 2: " + mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();
        Del deleg = new Del(PrintMes);
        deleg += PrintMes1;
        cl.Method(3, deleg);
        Console.ReadKey();
    }
}
```

# Удаление метода

- Чтобы удалить метод из списка вызова, используйте оператор decrement или назначения decrement ("-" или «-=»).

# Удаление метода

- И «отписать метод»

```
class Program
{
    2 references
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }
    1 reference
    public static void PrintMes1(string mes)
    {
        Console.WriteLine("Вариант 2: " + mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();
        Del deleg = new Del(PrintMes);
        deleg += PrintMes1;
        cl.Method(3, deleg);
        deleg -= PrintMes;
        cl.Method(3, deleg);
        Console.ReadKey();
    }
}
```



# Анонимные методы

- Что означает «анонимный метод»?
- Создание анонимных методов является, по существу, способом передачи **блока кода** в качестве **параметра делегата**.

# Пример

- Пример создания анонимного метода.
- Сам метод:  
delegate(int k) {Console.WriteLine("Число: " +  
k); };

```
public delegate void Del2(int x);  
2 references  
class Class6  
{  
    0 references  
    public Del2 Method2()  
    {  
        Del2 del = delegate(int k) {Console.WriteLine("Число: " + k); };  
        return del;  
    }  
}
```

# Использование метода

- Пример получения такого анонимного метода и вызов его

```
class Program
{
    0 references
    public static void PrintMes(string mes)
    {
        Console.WriteLine(mes);
    }
    0 references
    public static void PrintMes1(string mes)
    {
        Console.WriteLine("Вариант 2: " + mes);
    }

    0 references
    static void Main(string[] args)
    {
        Class6 cl = new Class6();

        var del = cl.Method2();
        del(6);

        Console.ReadKey();
    }
}
```

# Зачем они нужны

- Использование анонимных методов позволяет сократить издержки на кодирование при создании делегатов, поскольку **не требуется создавать отдельный метод.**
- Например, указание блока кода вместо делегата может быть целесообразно в ситуации, когда создание метода может показаться ненужным действием.

# Немного истории

- **Именованные методы** были единственным способом объявления делегата в версиях C#, предшествующих версии 2.0.
- **Анонимные методы** появились в C# 2.0 (Visual Studio 2005)
- А в версии C# 3.0 (Visual Studio 2008) их заменили **лямбда-выражения**.

# Лямбда-выражения

- Лямбда-выражение — это анонимная функция, с помощью которой можно создавать типы делегатов или деревьев выражений.
- Лямбда-выражения особенно полезны при написании выражений запросов LINQ.

# Лямбда-выражения

- Чтобы создать лямбда-выражение, необходимо указать входные параметры (если они есть) с левой стороны лямбда-оператора  $\Rightarrow$ , и поместить блок выражений или операторов с другой стороны.

# Пример

- Например, лямбда-выражение  $x \Rightarrow x * x$  задает параметр с именем  $x$  и возвращает квадрат значения  $x$ .

```
delegate int del(int i);
static void Main(string[] args)
{
    del myDelegate =  $x \Rightarrow x * x$ ;
    int j = myDelegate(5); //j = 25
}
```



# Еще пример

- Возьмем наш делегат и сделаем от него лямбда-выражение

```
public delegate void Del(string message);
```

```
static void Main(string[] args)
{
    Del delegat = (x) => Console.WriteLine("Лямбда-выражение: " + x);
    delegat("6");
}
```

# Шаблон

- Лямбда-выражение с выражением с правой стороны оператора => называется **выражением-лямбдой**.
- Выражения-лямбды возвращают результат выражения и принимают следующую основную форму.

(input parameters) => expression

# Скобки в выражении

- Если лямбда имеет только **один** входной параметр, **скобки можно не ставить**, во всех остальных случаях они **обязательны**. Два и более входных параметра разделяются запятыми и заключаются в скобки:

$$(x, y) \Rightarrow x == y$$

# Явное указание типов

- Иногда компилятору бывает трудно или даже невозможно вывести типы входных параметров. В этом случае типы можно указать в явном виде, как показано в следующем примере.

```
(int x, string s) => s.Length > x
```

# Пустые параметры

- Отсутствие входных параметров задаётся пустыми скобками.

`() => SomeMethod()`

- Тело выражения-лямбды может состоять из вызова метода.

# Выражение

- **Лямбда операторов** (или операторная лямбда) напоминает выражение-лямбду, за исключением того, что оператор (или операторы) заключается в **фигурные скобки**:

(input parameters) => {statement;}

# Пример

- Изменим наш пример, добавим еще операцию в наше выражение

```
static void Main(string[] args)
{
    Del delegat = (x) =>
    {
        Console.WriteLine("Лямбда-выражение: " + x);
        Console.WriteLine("Лямбда-выражение2: " + x);
    };

    delegat("6");
}
```

# Особенности

- Тело лямбды оператора может состоять из **любого** количества операторов;
- Однако на практике обычно используется не более **двух-трех**.



# Событийное программирование

- Ситуация: вы получили права и хотите, чтобы все об этом знали. Что бы делали лет 5-10 тому назад?



# Событийное программирование

- Что делают сейчас?



# Как это можно описать

- 1 Вариант: вам нужно иметь записную книжку с номерами **всех** тех, кого вы хотите оповестить о каком-то событии. И **каждому** нужно еще **позвонить** и сказать об этом.

# Как это можно описать

- 2 Вариант: вы просто **«постите»** новость в социальной сети и все, кто на вас **«подписан»** видят вашу новость.

# Переведем на «программистский» ЯЗЫК

- Ранее: мы должны были иметь объекты **всех** классов, кто должен знать об изменениях внутри класса и вызывать методы этих объектов.
- А представьте, что появилось еще 2 класса, кто должны знать об оповещениях. Сколько кода прописать придется?

# Переведем на «программистский» ЯЗЫК

- Теперь: мы создаем поле-«событие» в нашем классе и метод, с помощью которого любой класс может «подписаться» на это событие. Далее, если в классе происходят изменения, просто вызывается это событие и все классы, кто на него подписаны реагируют на это.

# Переведем на «программистский» ЯЗЫК

- «Изменением в классе» может быть любой вызываемый метод класса или событие.
- «Реакция другого класса» это метод того класса, который будет вызываться при совершении события.
- «Подписать на событие» это передать в поле-событие класса метод другого класса, который будет вызываться при наступлении события.

# Как создавать

- Чтобы создать событие нам понадобится делегат  
public delegate void Del(string message);
- Теперь в классе можем создать поле типа **событие**, используя ключевое слово **event**  
private event Del event1;



# Что такое событие

- 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**.

# Что такое событие

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

# Что такое событие

- В C# в стандартном приложении Windows Forms или веб-приложении вы подписываетесь на **события**, вызываемые **элементами управления**, такими как кнопки и поля со списками.

# Как создавать

- Создать метод, с помощью которого на это событие можно будет подписаться.

```
public void AddToEvent(Del method)    {  
    if(event1 == null)  
        event1 = new Del(method);  
    else  
        event1 += method;    }
```

# Как создавать

- И метод, в котором событие будет вызываться.

```
public void SomeMethod()    {  
    //что-то происходит  
    if(event1 != null)  
    {  
        event1("Что-то произошло");  
    }  
}
```

# Пример

- Как это может выглядеть наглядно

```
public void SomeMethod()
{
    //что-то происходит
    if(event1 != null)
    {
        event1("Что-то произошло");
    }
}
```

```
namespace ConsoleApplication1
{
    public delegate void Del(string message);

    public delegate void Del2(int x);
    2 references
    class Class6
    {
        private event Del event1;

        0 references
        public void AddToEvent(Del method)
        {
            if(event1 == null)
            {
                event1 = new Del(method);
            }
            else
            {
                event1 += method;
            }
        }
    }
}
```

# Как создавать

- Все, что остается, это привязать метод одного класса к событию в этом классе через объекты классов.
- Создадим другой класс.

```
class ClassOther
{
    0 references
    public void Method(string s)
    {
        Console.WriteLine("ClassOther: " + s);
    }
}
```

# Как создавать

- Теперь связываем наши классы

```
class Program
{
    0 references
    static void Main(string[] args)
    {
        Class6 c1 = new Class6();

        ClassOther otherC1 = new ClassOther();
        //связываем метод и событие
        c1.AddToEvent(otherC1.Method);

        //где-то метод вызывается
        c1.SomeMethod();
    }
}
```



# События в С#

- В С# в стандартном приложении Windows Forms или веб-приложении вы **подписываетесь на события**, вызываемые элементами управления, такими как **кнопки и поля со списками** и т.п.

# Пример из лабораторной

- Если посмотреть файл Form1.Designer.cs

```
//  
// buttonSetSportCabriolet  
//  
this.buttonSetSportCabriolet.Location = new System.Drawing.Point(471, 523);  
this.buttonSetSportCabriolet.Name = "buttonSetSportCabriolet";  
this.buttonSetSportCabriolet.Size = new System.Drawing.Size(153, 23);  
this.buttonSetSportCabriolet.TabIndex = 5;  
this.buttonSetSportCabriolet.Text = "Задать кабриолет-спорт";  
this.buttonSetSportCabriolet.UseVisualStyleBackColor = true;  
this.buttonSetSportCabriolet.Click += new System.EventHandler(this.buttonSetSportCabriolet_Click);  
//  
// buttonSetSportSedan  
//  
this.buttonSetSportSedan.Location = new System.Drawing.Point(317, 523);  
this.buttonSetSportSedan.Name = "buttonSetSportSedan";  
this.buttonSetSportSedan.Size = new System.Drawing.Size(129, 23);  
this.buttonSetSportSedan.TabIndex = 4;  
this.buttonSetSportSedan.Text = "Задать седан-спорт";  
this.buttonSetSportSedan.UseVisualStyleBackColor = true;  
this.buttonSetSportSedan.Click += new System.EventHandler(this.buttonSetSportSedan_Click);
```

# Пример из лабораторной

- Мы можем прямо в коде добавлять или удалять методы обработки от событий элементов формы.

2 references

```
private void buttonSetSedan_Click(object sender, EventArgs e)
{
    buttonSetCabriolet.Click -= buttonSetCabriolet_Click;
    buttonSetCabriolet.Click += buttonSetSedan_Click;
    inter = new Car(100, 4, Color.Wheat, 0);
    Bitmap bmp = new Bitmap(pictureBox1.Width, pictureBox1.Height);
    Graphics gr = Graphics.FromImage(bmp);
    inter.bodyType(gr);
    pictureBox1.Image = bmp;
}
```