

ЛЕКЦИЯ

ДЕЛЕГАТЫ И СОБЫТИЯ

Лектор Крамар Ю.М.

Содержание

1. Объявление и использование делегатов
2. Использование лямбда-выражений
3. Обработка событий

ОБЪЯВЛЕНИЕ И ИСПОЛЬЗОВАНИЕ ДЕЛЕГАТОВ

Зачем нужны делегаты?

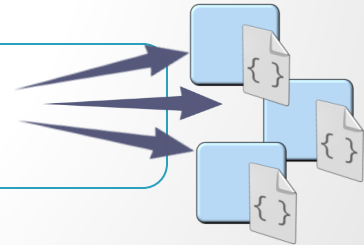
Методы определяются динамически во время выполнения



Методы обратного вызова



Групповые (multicast) операции



Назначение делегатов. 1

Динамическое определение вызываемых методов



При разработке фреймворков, в которых различные методы вызываются для выполнения определенной работы в зависимости от критериев, определяемых динамически.

Одним из способов реализации такой функциональности является использование операторов `if` или `switch`. Однако такой подход является статичным.

Более расширяемый подход заключается в использовании делегатов.

Назначение делегатов. 2

Использование методов обратного вызова (callback)

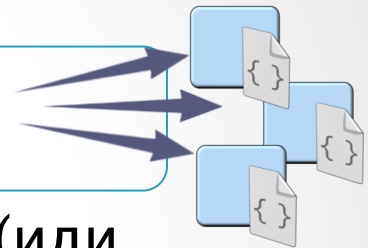


Сторонние разработчики, предоставляющие сборки с методами, которые можно вызвать асинхронно, часто позволяют указать метод в коде пользователя, при этом код не переписывается.

При таком сценарии заранее неизвестны варианты всех возможных в этом случае методов, поэтому эффективным решением является использование делегата, который можно связать с одним или более методов.

Назначение делегатов. 3

Выполнение групповых (multicast) операций



Если операция является групповой (или расширяемой новыми методами), несколько методов могут реализовывать одну и ту же операцию в различных вариантах.

Тогда можно добавить ссылки на все реализации методов делегату, а при вызове делегата среда выполнения вызовет каждый метод по очереди. При использовании групповых операций реализующие их методы вызываются последовательно в соответствии с тем, как они были добавлены в делегат.

Что такое делегат?

Делегат – это специальный тип классов, определенный в .NET, назначение объектов которого – ссылаться на определенный метод.

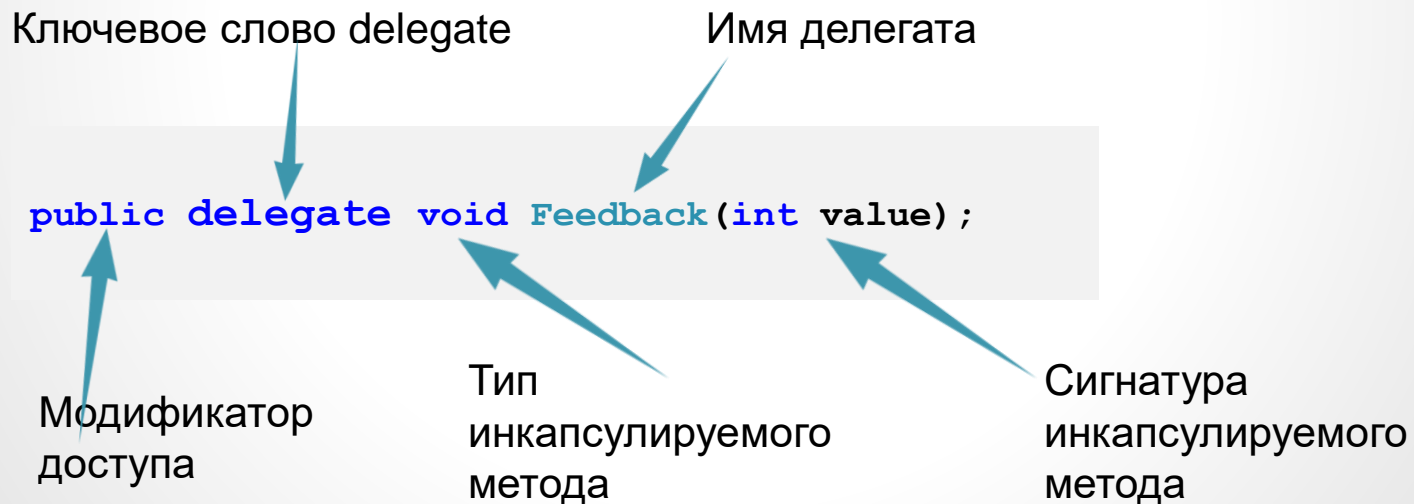
Так как назначение этого класса сводится лишь к хранению ссылки на метод, а его объекта – вызывать этот метод, то такие классы имеют одинаковую структуру и незначительно отличающееся между собой описание.

Определение делегата

Делегат – это пользовательский тип, который инкапсулирует метод

Объект делегата поддерживает три важных фрагмента информации:

- адрес метода, на котором он вызывается;
- аргументы метода (если они есть);
- возвращаемое значение метода (если оно есть).



Определение делегата

```
internal delegate void Feedback(int value);
```

Visual C#
компилятор

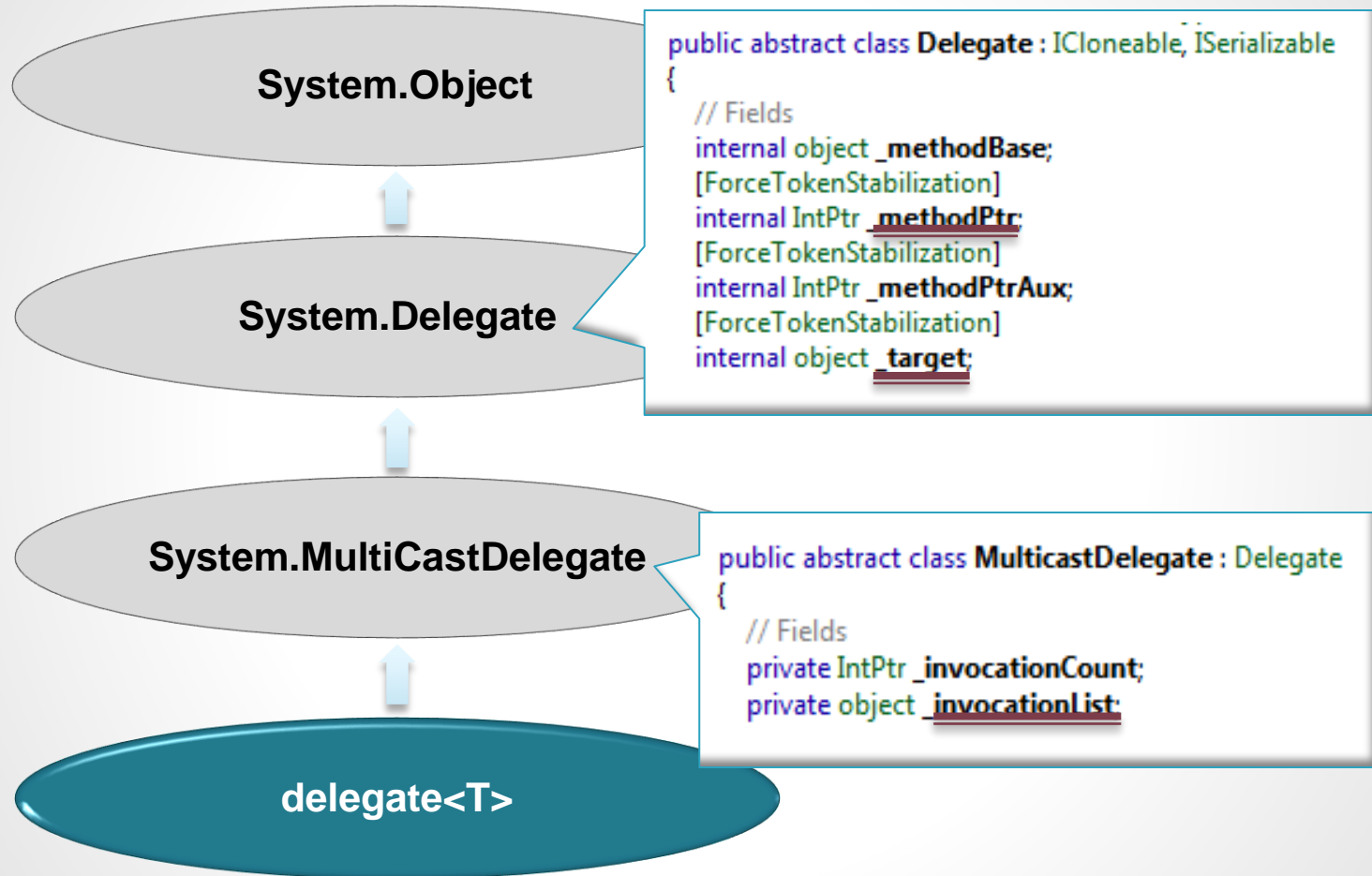


```
public class Feedback: MulticastDelegate
{
    public Feedback(Object object, IntPtr method);
    public virtual void Invoke(Int32 value);
    // Методы, обеспечивающие асинхронный обратный вызов,
    public virtual IAsyncResult BeginInvoke(Int32 value,
        AsyncCallback callback, Object object);
    public virtual void EndInvoke(IAsyncResult result);
}
```

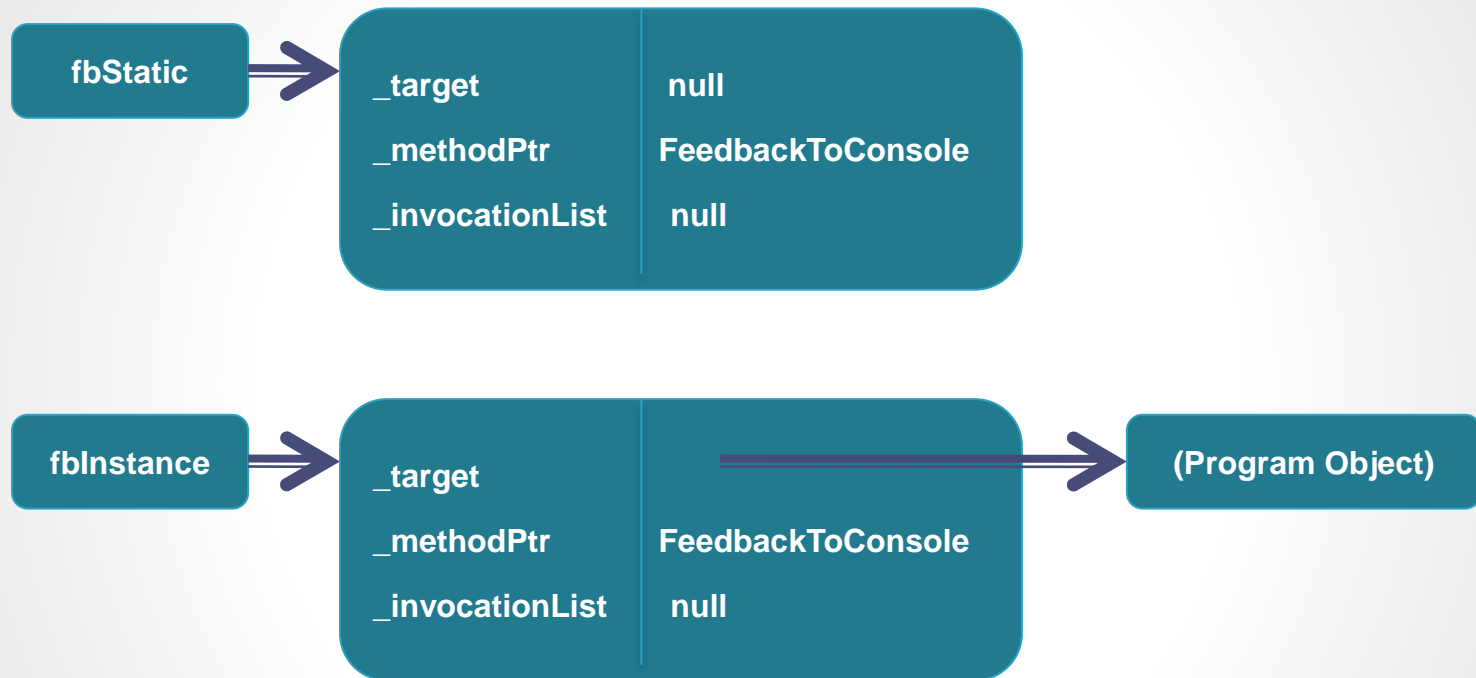
Определение делегата



Определение делегата



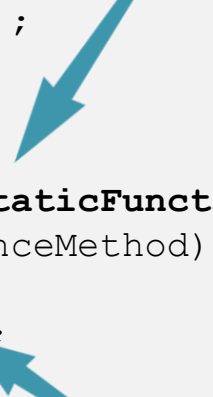
Определение делегата



Определение делегата

Инициализируются именем метода
или именем другого делегата

```
public delegate void Feedback(int value);  
...  
Feedback F;  
...  
fbStatic = new Feedback(ClassName.SomeStaticFunction);  
fbInstance = new Feedback(obj.SomeInstanceMethod);  
//or  
fbStatic = ClassName.SomeStaticFunction;  
fbInstance = obj.SomeInstanceMethod;
```



Использование упрощенного
синтаксиса – достаточно
указать имя метода без
применения new

Определение делегата

Класс `System.MulticastDelegate` (в сочетании с его базовым классом `System.Delegate`) обеспечивает своим наследникам доступ к списку, содержащему адреса методов, поддерживаемых типом делегата, а также несколькими дополнительными методами (и перегруженными операциями), чтобы взаимодействовать со списком вызовов

ВЫЗОВОВ

```
public abstract class MulticastDelegate : Delegate
{
    // Возвращает список методов, на которые "указывает"
    // делегат.
    public sealed override Delegate[] GetInvocationList ();
    // Перегруженные операции.
    public static bool operator ==(MulticastDelegate d1,
    MulticastDelegate d2);
    public static bool operator != (MulticastDelegate d1,
    MulticastDelegate d2);
    // Используются внутренне для управления списком методов,
    // поддерживаемых делегатом.
    private IntPtr _invocationCount;
    private object _invocationList;
}
```


Определение делегата

```
public abstract class Delegate : ICloneable, ISerializable
{
    // Методы для взаимодействия со списком функций.
    public static Delegate Combine(params Delegate[] delegates);
    public static Delegate Combine(Delegate a, Delegate b);
    public static Delegate Remove(Delegate source, Delegate
value);
    public static Delegate RemoveAll(Delegate source, Delegate
value);
    // Перегруженные операции.
    public static bool operator ==(Delegate d1, Delegate d2);
    public static bool operator !=(Delegate d1, Delegate d2);
    // Свойства, показывающие цель делегата.
    public MethodInfo Method { get; }
    public object Target { get; }
}
```

Статический метод `Combine` добавляет метод в список, поддерживаемый делегатом. В C# этот метод вызывается за счет использования перегруженной операции `+=`

Статические методы удаляют метод (или все методы) из списка вызовов делегата. В C# метод `Remove` может быть вызван неявно, посредством перегруженной операции `-=`

Вызов делегата

Следует всегда проверять делегат на null, прежде чем ссылаться на него

```
public IsValidDelegate isValid = null;  
.  
.  
.  
if (isValid != null)  
{  
    isValid();  
}
```

isValid();

**Visual C#
компилятор**

isValid.Invoke();

Групповые делегаты

```
DelegateIntro di = new DelegateIntro();  
...  
Feedback fb1 = new Feedback(FeedbackToConsole);  
Feedback fb2 = new Feedback(FeedbackToMsgBox);  
Feedback fb3 = new Feedback(di.FeedbackToFile);  
  
Feedback fbChain =  
(Feedback) Delegate.Combine(fb1, fb2);  
fbChain = (Feedback) Delegate.Combine(fbChain,  
fb3);
```

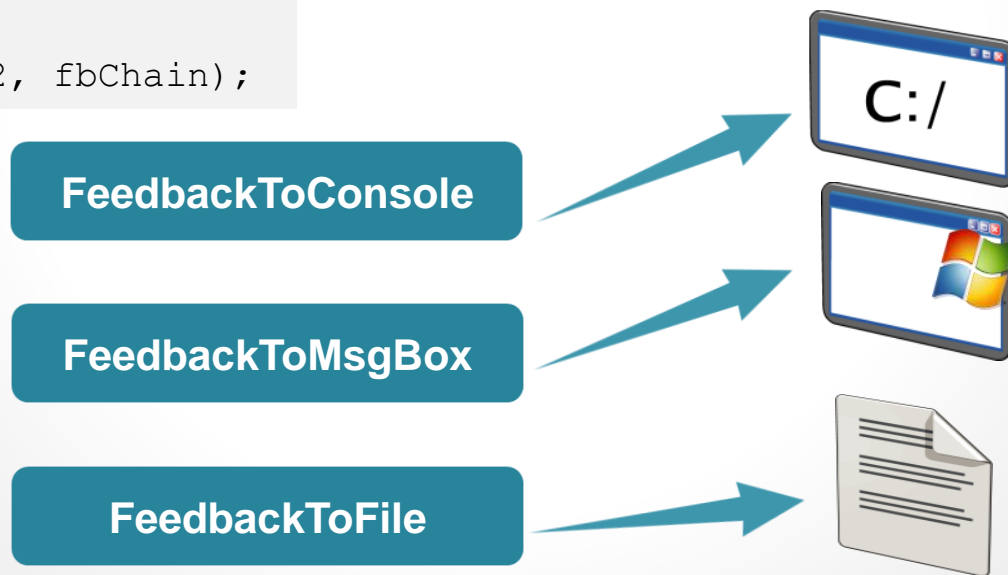
or

```
DelegateIntro di = new DelegateIntro();  
...  
Feedback fb1 = new Feedback(FeedbackToConsole);  
Feedback fb2 = new Feedback(FeedbackToMsgBox);  
Feedback fb3 = new Feedback(di.FeedbackToFile);  
  
Feedback fbChain = new  
Feedback(FeedbackToConsole);  
fbChain += FeedbackToMsgBox;  
fbChain += di.FeedbackToFile;
```

Групповые делегаты

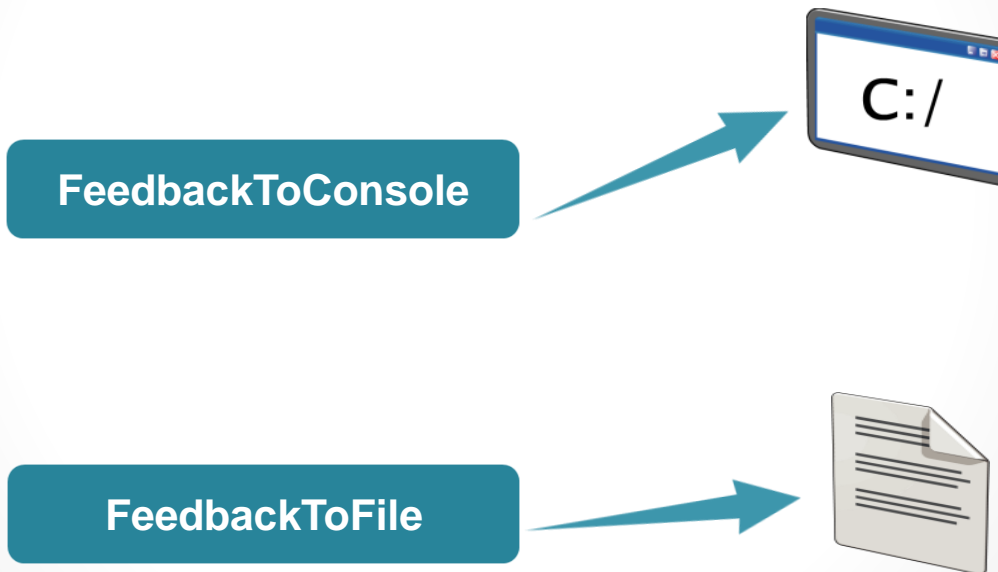
```
private static void Counter(Int32 from, Int32 to, Feedback fb)
{
    for (Int32 val = from; val <= to; val++)
    {
        // If any callbacks are specified, call them
        if (fb != null)
            fb(val);
    }
}
```

```
...
Counter(1, 2, fbChain);
```



Групповые делегаты

```
...  
fbChain = (Feedback) Delegate.Remove(fbChain, new  
Feedback(FeedbackToMsgBox));
```



Групповые делегаты

```
DelegateIntro di;
```

```
. . .
```

```
Feedback fb1 = new Feedback(FeedbackToConsole);
```

```
Feedback fb2 = new Feedback(FeedbackToMsgBox);
```

```
Feedback fb3 = new Feedback(di.FeedbackToFile);
```

```
fbChain -= new Feedback(FeedbackToMsgBox);
```

```
Feedback fbChain = null;
```

```
fbChain += fb1;
```

```
fbChain += fb2;
```

```
fbChain += fb3;
```

```
. . .
```

```
fbChain -= new Feedback(FeedbackToMsgBox);
```

```
fbChain = (Feedback) Delegate.Combine(fbChain, fb2);
```

```
fbChain = (Feedback) Delegate.Combine(fbChain, fb2);
```

```
fbChain = (Feedback) Delegate.Combine(fbChain, fb3);
```

```
. . .
```

```
fbChain = (Feedback) Delegate.Remove(fbChain, new  
Feedback(FeedbackToMsgBox));
```

Определение анонимных методов

Добавление
анонимного метода в
качестве обработчика
для делегата

Для определения
анонимного метода
используется ключевое
слово `delegate`

Для анонимного
метода можно
указать имена
параметров и типов

```
myDelegateInstance += new MyDelegate(delegate(int parameter)
{
    return 10;
});
```

Тело анонимного метода

Тип возвращаемого значения не указывается; компилятор будет работать исходя из возвращаемого типа в зависимости от контекста

Определение анонимных методов

```
delegate int MyDelegate(int number);  
...  
MyDelegate myDelegateInstance = null;  
public void addAnonymousMethodsToDelegate()  
{  
    // Add an anonymous method to the delegate.  
    // Do not specify any parameters.  
    myDelegateInstance += new MyDelegate(delegate  
    {  
        // Perform operation.  
        return 5;  
    });  
    // Add an anonymous method to the delegate.  
    // Specify parameters; the parameters must match  
    // the signature of the delegate.  
    myDelegateInstance += new MyDelegate(delegate(int parameter)  
    {  
        return 10;  
    });  
    // Invoke the delegate.  
    int returnedValue = myDelegateInstance(2);  
    // = 10 (as second method is called last).  
}
```

ИСПОЛЬЗОВАНИЕ ЛЯМБДА- ВЫРАЖЕНИЙ

Что такое лямбда-выражения?

- Лямбда-выражением является выражение, возвращающее метод
- Лямбда-выражения широко используются в Visual C# особенно при определении Language-Integrated Query (LINQ) выражений
- Лямбда-выражение определяется с помощью операции «=>»

Набор параметров
лямбда-выражения

$x \Rightarrow x * x$

Тело лямбда-
выражения

Тело определяет функцию, которая может вернуть значение, в этом случае Visual C# компилятор выводит возвращаемый тип из определения метода

```
delegate int MyDelegate(int a);  
...  
MyDelegate myDelegateInstance = null;  
myDelegateInstance += new MyDelegate (x => x * x);  
  
myDelegateInstance += x => x * x;  
  
Console.WriteLine(myDelegateInstance(10));
```

Определение лямбда-выражений

- Лямбда-выражение может принимать более одного параметра

```
delegate int AddDelegate(int a, int b)
```

```
...
```

```
AddDelegate myAddDelegate = null;
```

```
myAddDelegate += (x, y) => x + y;
```

- Простое выражение, возвращающее квадрат параметров. Тип параметра x выводится из контекста.

```
x => x * x
```

- Семантически аналогично предыдущему выражению, но в качестве тела используется не простое выражение, а Visual C# операторный блок

```
x => {return x * x ; }
```

- Простое выражение, возвращающее значение параметра деленное на 2. Тип параметра x указано в явном виде

```
(int x) => x / 2 ;
```

Определение лямбда-выражений

- Вызов метода. Выражение не принимает никаких параметров. Выражение может возвращать или не возвращать значение.

```
() => myObject.MyMethod(0)
```

- Выражение принимает два параметра, компилятор выводит типы параметров. Параметр `x` передается по значению, следовательно эффект операции инкремента является для выражения локальным.

```
(x, y) => { x++; return x / y; }
```

- Выражение принимает два параметра с явным указанием типов. Параметр `x` передается по ссылке, следовательно операция инкремента имеет побочный эффект

```
(ref int x, int y) { x++; return x / y; }
```

Область действия переменной в лямбда-выражениях

- При определении лямбда-выражения в его теле можно определить переменные
- Областью видимости переменных, определенных в теле лямбда-выражения, является само лямбда-выражение
- Код вне лямбда-выражения не может получить доступ к переменным, определенным в нем, но в лямбда-выражении можно получить доступ к переменным, определенным вне этого выражения

```
MyDelegate del = null; ...  
void myMethod()  
{  
    int count = 0;  
    del += new MyDelegate(() =>  
    {  
        count++;  
        // Perform operation using count variable.  
        ...  
    });  
}
```

Захват внешнего
контекста

Использование внешней переменной в лямбда-выражении может расширить ее жизненный цикл

Обобщенные делегаты .NET Framework

Обобщенный делегат определяется с помощью параметров типа аналогично использованию параметров типа в объявлении метода

```
delegate void PrintDocumentDelegate<DocumentType>(DocumentType document);
```

- .NET Framework включает несколько встроенных обобщенных делегатов
- Основными обобщенными делегатами являются Action и Func

- **Action<T>**

Делегат, используемый для инкапсуляции методов, не возвращающих значения

- **Func<T, TResult>**

Делегат, используемый для инкапсуляции методов, возвращающих значение

Обобщенные делегаты .NET Framework

Примеры использования делегатов Action и Func

```
Action<string, int> myDelegate = null;
myDelegate += ((param1, param2) =>
{
    Console.WriteLine("{0} : {1}", param1,
param2.ToString());
});
if (myDelegate != null)
{
    myDelegate("Value", 5);
}

Func<string, int, string> myDelegate = null;
myDelegate += ((param1, param2) =>
{
    return String.Format("{0} : {1}", param1,
param2.ToString());
});
if (myDelegate != null)
{
    string returnedValue;
    returnedValue = myDelegate("Value", 5);
    Console.WriteLine(returnedValue);
}
```

Обобщенные делегаты .NET Framework

- `Func< in T, out TResult>`
- `Func<in T1, in T2, out TResult>`
- ...
- `Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, TResult>`

универсальный делегат, инкапсулирующий метод, который принимает 16 параметров обобщенных типов T1 – T16 и возвращает значение типа, указанного в параметре TResult.


- `Action<in T>`
- `Action<in T1, in T2>`
- ...
- `Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16>`

универсальный делегат, инкапсулирующий метод, который принимает 16 параметров и не возвращает значений

ОБРАБОТКА СОБЫТИЙ

Что такое событие?

Событие (event) – это автоматическое уведомление о выполнении некоторого действия. Событие является особым типом элементов класса. Если в типе определен член–событие, то его экземпляр может уведомлять о наступлении этого события, а другие объекты могут получать эти уведомления и реагировать в ответ определенными действиями.



Что такое событие?

Тип, в котором определены события, должен иметь генератор события – «движок», который эти события инициирует. Если другой объект «интересуется» данным событием и «желает» в ответ на его возникновение определенным образом реагировать, то этот объект должен, во-первых, определить реакцию – метод-обработчик события и, во-вторых, подписаться на данное событие, зарегистрировав этот обработчик.

Что такое событие?

События обеспечивают механизм информирования приложений, использующих тип, о изменении состояния или других проявлениях в типе

- События основаны на делегатах
- Любой объект, имеющий доступ к делегату, может ссылаться на такой делегат, однако только тип, определяющий событие, может вызвать это событие
- Когда тип определяет событие, другие типы могут определить метод, соответствующий сигнатуре делегата, связанного с событием, и подписаться на событие, указав, что их метод должен быть запущен при возникновении события
- Классы в Microsoft .NET Framework широко используют события, в том числе почти все элементы управления Windows Presentation Foundation (WPF)

Определение события

Делегат, описывающий метод обработки события; делегат должен быть, по крайней мере, уровня доступа события

Делегат не должен возвращать значение

```
public delegate void MyEventDelegate(object sender, EventArgs e);  
  
public event MyEventDelegate MyEvent = null;
```

События, как правило, определяются уровня доступа public

Для определения события используется ключевое слово event и указывается делегат

Делегат должен принимать два параметра

- Делегат может существовать на уровне пространства имен или класса, событие может определяться только на уровне класса
- События можно определить в интерфейсе, делегат – нет

Использование события

После того как событие определено, его можно использовать в приложении, для этого на событие нужно подписаться в использующих типах и приложениях и генерировать событие в типе

- Для подписки на событие используется составная операция присваивания «+=»

```
MyEvent += new MyEventHandler(myHandlingMethod);
```

- Отказаться от подписки на событие можно с помощью составной операции присваивания «-=»

```
MyEvent -= myHandlingMethod;
```

- При вызове события необходимо сначала проверить, что оно не является нулевым

```
if (MyEvent != null)
{
    EventArgs args = new EventArgs();
    MyEvent(this, args);
}
```

```
delegate void DelEventType();
```

```
class myClassEvent
{
    public myClassEvent(string objName)
    {
        name = objName;
    }

    public event DelEventType MyEvent;

    protected virtual void OnMyEvent()
    {
        if (MyEvent != null)
            MyEvent();
    }

    public void Counter()
    {
        for (int i = 1; i <= 100; i = i + 1)
        {
            Thread.Sleep(3000);
            if (i % 10 == 0)
                OnMyEvent();
            i = i + 1;
        }
    }

    private string name;
}
```

```
class Program
{
    static void EventHandler()
    {
        Console.WriteLine("Event's happened!");
    }

    static void Main(string[] args)
    {
        myClassEvent objEvent = new myClassEvent("Object1");
        objEvent.MyEvent += new DelEventType(EventHandler);
        objEvent.Counter();
        objEvent.MyEvent -= new DelEventType(EventHandler);
        objEvent.Counter();
    }
}
```

OUTPUT (10 times):

Event's happened!

Event's happened!

...

Event's happened!

Параметры события

Если в программе существует несколько объектов, инициирующих событие, то часто важно знать не только о том, что событие произошло, но и **источник** генерации события, и дополнительную **информацию**, характеризующую событие (например, для события нажатия клавиши – ее код).


```
delegate void DelEventType1(object sender, int x);
```

```
class myClassEvent1
```

```
{
```

```
    public myClassEvent1(string objName)
```

```
    {
```

```
        name = objName;
```

```
    }
```

```
    public event DelEventType1 MyEvent;
```

```
    public void OnMyEvent(int i)
```

```
    {
```

```
        if (MyEvent != null)
```

```
            MyEvent(this, i);
```

```
    }
```

```
    public string name;
```

```
}
```

```
...
```

```
static void EventHandler1(object sender, int x)
{
    Console.WriteLine("Object {0} throw event : x*x =
{1}", ((myClassEvent1)sender).name, x * x);
}

...
myClassEvent1 objEvent1 = new myClassEvent1("Object1");
myClassEvent1 objEvent2 = new myClassEvent1("Object2");

objEvent1.MyEvent += new DelEventType1(EventHandler1);
objEvent2.MyEvent += new DelEventType1(EventHandler1);

objEvent1.OnMyEvent(5);
objEvent2.OnMyEvent(6);
```

OUTPUT:

Object Object1 threw event : x*x = 25

Object Object2 threw event : x*x = 36

Стандартные события

Для совместимости со средой .Net Framework лучше использовать стандарт поддержки событий Microsoft. Согласно стандарту требуется указывать два параметра для обработчика любого события.

Первый должен быть ссылкой на объект, генерирующий событие, второй должен иметь тип EventArgs или производный от него и содержать информацию, характерную для события:

```
void name_event_handler(object source, EventArgs  
arg)
```

// Derive a custom EventArgs class that holds the key.

```
class KeyEventArgs : EventArgs
{
    public char ch;
}
```

// Declare a delegate type for an event.

```
delegate void KeyHandler(object source, KeyEventArgs arg);
```

// Declare a keypress event class.

```
class KeyEvent
{
    public event KeyHandler KeyPress;

    // This is called when a key is pressed.
    public void OnKeyPress(char key)
    {
        KeyEventArgs k = new KeyEventArgs();

        if (KeyPress != null)
        {
            k.ch = key;
            KeyPress(this, k);
        }
    }
}
```


```
static void Main()
{
    KeyEvent kevt = new KeyEvent();
    ConsoleKeyInfo key;
    int count = 0;

    // Use a lambda expression to display the keypress.
    kevt.KeyPress += (source, arg) =>
        Console.WriteLine(" Received keystroke: " + arg.ch);

    // Use a lambda expression to count keypresses.
    kevt.KeyPress += (source, arg) =>
        count++; // count is an outer variable

    Console.WriteLine("Enter some characters. " +
        "Enter a period to stop.");
    do
    {
        key = Console.ReadKey();
        kevt.OnKeyPress(key.KeyChar);
    } while (key.KeyChar != '.');

    Console.WriteLine(count + " keys pressed.");
}
```

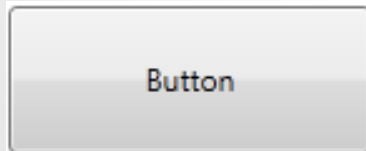


Использование событий в графических приложениях

Графические приложения используют большое количество событий и обычно включают много взаимодействий с пользователем



```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Логика обработки события нажатия кнопки
}
```



Рекомендации по использованию событий

- При определении делегата для события следует использовать стандартную сигнатуру события
- Для вызова события следует использовать защищенный виртуальный метод
- Не следует передавать событиям нулевые значения

ДЕЛЕГАТЫ И СОБЫТИЯ

Ресурсы:

1. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/>
2. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/>
3. **Шилд Г. С# : учебный курс.- СПб.:Питер;К.: Издательская группа ВHV, 2002.-512с.**