

СЕДЬМОЕ ЗАНЯТИЕ

ДЕЛЕГАТЫ

- Кроме свойств и методов классы могут содержать делегаты и события. Делегаты представляют такие объекты, которые указывают на другие методы. То есть делегаты - это указатели на методы. С помощью делегатов мы можем вызвать определенные методы в ответ на некоторые произошедшие действия. То есть, по сути, делегаты раскрывают нам функционал функций обратного вызова.

ПРИМЕРЫ ДЕЛЕГАТОВ

ДЕЛЕГАТ ОПРЕДЕЛЯЕТ ТО, КАКИЕ ПАРАМЕТРЫ ОН ПРИНИМАЕТ, И ЧТО ИМЕННО ВОЗВРАЩАЕТ. ДАЛЕЕ МЫ МОЖЕМ ПРИСВАИВАТЬ ПЕРЕМЕННЫМ ТИПА НАШЕГО ДЕЛЕГАТА МЕТОДЫ С ТАКОЙ ЖЕ СИГНАТУРОЙ

```
DELEGATE INT OPERATION(INT X, INT Y);
```

```
DELEGATE VOID GETMESSAGE();
```



```
CLASS PROGRAM {  
    DELEGATE VOID GETMESSAGE(); // 1. ОБЪЯВЛЯЕМ ДЕЛЕГАТ  
    STATIC VOID MAIN(STRING[] ARGS) {  
        GETMESSAGE DEL; // 2. СОЗДАЕМ ПЕРЕМЕННУЮ ДЕЛЕГАТА  
        IF (DATE TIME.NOW.HOUR < 12) {  
            DEL = GOODMORNING; // 3. ПРИСВАИВАЕМ ЭТОЙ ПЕРЕМЕННОЙ АДРЕС МЕТОДА  
        } ELSE {  
            DEL = GOODEVENING;  
        }  
        DEL.INVOKE(); // 4. ВЫЗЫВАЕМ МЕТОД  
        CONSOLE.READLINE();  
    }  
    PRIVATE STATIC VOID GOODMORNING() {  
        CONSOLE.WRITELINE("GOOD MORNING");  
    }  
    PRIVATE STATIC VOID GOODEVENING() {  
        CONSOLE.WRITELINE("GOOD EVENING");  
    }  
}
```


СОБЫТИЯ

События используются для выполнения определенного кода при наступлении некоторого события. Это лишь обертки над делегатами, но очень удобные

Мы можем подписываться на событие неограниченное количество обработчиков, при помощи +=, при необходимости, мы можем отписать обработчик, используя -=

```
DELEGATE VOID SAMPLEDELEGATE();
```

```
EVENT SAMPLEDELEGATE SAMPLEEVENT();
```

```
//ИМЕЕТСЯ МЕТОД
```

```
VOID SOMEACTION()
```

```
{ CONSOLE.WRITELINE("SOME");}
```

```
//НАШ КОД
```

```
MYTYPE v = NEW MYTYPE();
```

```
v.SAMPLEEVENT += SOMEACTION;
```

```
//ТЕПЕРЬ, КОГДА БУДЕТ НЕОБХОДИМО, МЕТОД SOMEACTION  
БУДЕТ ВЫЗВАН, И НАМ НЕ НАДО ПРО ЭТО ДУМАТЬ.
```


АНОНИМНЫЕ МЕТОДЫ

При подписке на событие или передаче делегата не всегда удобно писать непосредственный код в отдельных методах. Мы можем написать необходимый код прямо на месте

В скобках необходимо указывать имена принимаемых делегатом параметров, далее в фигурных скобках непосредственно исполняемый код

```
v.SAMPLEEVENT += DELEGATE()  
{  
    CONSOLE.WRITELINE("ANON METHOD");  
}
```


ЛЯМБДЫ

- 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.
- Лямбда-выражения имеют следующий синтаксис: слева от лямбда-оператора \Rightarrow определяется список параметров, а справа блок выражений, использующий эти параметры: (СПИСОК_ПАРАМЕТРОВ) \Rightarrow ВЫРАЖЕНИЕ.

ПРИМЕР ПРОСТОЙ ЛЯМБДЫ

```
CLASS PROGRAM
{
    DELEGATE INT SQUARE(INT X); // ОБЪЯВЛЯЕМ ДЕЛЕГАТ, ПРИНИМАЮЩИЙ INT И
    ВОЗВРАЩАЮЩИЙ INT
    STATIC VOID MAIN(STRING[] ARGS)
    {
        SQUARE SQUAREINT = I => I * I; // ОБЪЕКТУ ДЕЛЕГАТА ПРИСВАИВАЕТСЯ
        ЛЯМБДА-ВЫРАЖЕНИЕ

        INT Z = SQUAREINT(6); // ИСПОЛЬЗУЕМ ДЕЛЕГАТ
        CONSOLE.WRITELINE(Z); // ВЫВОДИТ ЧИСЛО 36
        CONSOLE.READ();
    }
}
```


ACTION, FUNC

- В C# имеются уже определенные обобщенные делегаты, которые мы можем использовать, не прибегая к написанию собственных.
- Так, делегат `Action<T>` определяет делегат, которые ничего не возвращает, но принимает параметр типа `T`. Параметром может быть несколько `Action<T1, T2>`
- А делегат `Func` используется для возвращения некоторого значения. Тип возвращаемого значения указывается последним. `Func<TResult>` ничего не принимает, возвращает `TResult`. А `Func<T1, T2, TResult>` принимает `T1` и `T2`, и возвращает `TResult`.

МНОГОПОТОЧНОСТЬ