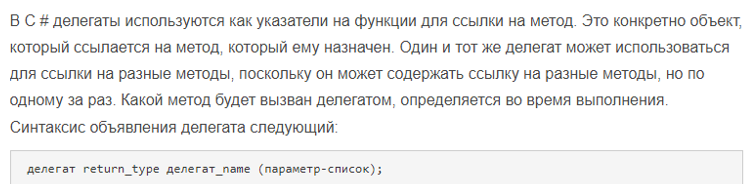
Делегаты и события

Лямбда-выражения и шаблон проектирования “наблюдатель”

Событийно-управляемые приложения

ДЕЛЕГАТЫ

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты - это указатели на методы и с помощью делегатов мы можем вызвать данные методы.



Для объявления делегата используется ключевое слово delegate, после которого идет возвращаемый тип, название и параметры. Например:

delegate void Message(T parametrs);

Определение Делегата

Делегат Message в качестве возвращаемого типа имеет тип void (то есть ничего не возвращает) и не принимает никаких параметров. Это значит, что этот делегат может указывать на любой метод, который не принимает никаких параметров и ничего не возвращает.

using System;

static class Program

{

delegate void Message();

static void Main()

{

Message mes; // 2. Создаем переменную делегата

mes = Hello; // 3. Присваиваем этой переменной адрес метода

mes(); // 4. Вызываем метод

void Hello() => Console.WriteLine("Hello, Deligate");

}

}

Если мы определяем делегат в прогаммах верхнего уровня (top-level program), которую по умолчанию представляет файл Program.cs начиная с версии C# 10, как в примере выше, то, как и другие типы, делегат определяется в конце кода. Но в принцие делегат можно определять внутри класса, либо вне класса

Добавление метода в делегат

Message? message = Hello;

message += HowAreYou; // теперь message указывает на два метода

message(); // вызываются оба метода - Hello и HowAreYou

Стоит отметить, что при удалении метода может сложиться ситуация, что в делегате не будет методов, и тогда переменная будет иметь значение null. Поэтому в данном случае переменная определена не просто как переменная типа Message, а именно Message?, то есть типа, который может представлять как делегат Message, так и значение null.

Другой способ вызова делегата представляет метод Invoke():

Message mes = Hello;

mes.Invoke(); // Hello

Operation op = Add;

int n = op.Invoke(3, 4);

Console.WriteLine(n); //3 7

Делегаты, как и другие типы, могут быть обобщенными, например:

Operation<decimal, int> squareOperation = Square;

decimal result1 = squareOperation(5);

Console.WriteLine(result1); // 25

Operation<int, int> doubleOperation = Double;

int result2 = doubleOperation(5);

Console.WriteLine(result2); // 10

decimal Square(int n) => n \* n;

int Double(int n) => n + n;

delegate T Operation<T, K>(K val);

Здесь делегат Operation типизируется двумя параметрами типов. Параметр T представляет тип возвращаемого значения. А параметр K представляет тип передаваемого в делегат параметра. Таким образом, этому делегату соответствует метод, который принимает параметр любого типа и возвращает значение любого типа.

В прогамме мы можем определить переменные делегата под определенный метод. Например, делегату Operation<decimal, int> соответствует метод, который принимает число int и возвращает число типа decimal. А делегату Operation<int, int> соответствует метод, который принимает и возвращает число типа int.

Также делегаты могут быть параметрами методов. Благодаря этому один метод в качестве параметров может получать действия - другие методы. Например:

DoOperation(5, 4, Add); // 9

DoOperation(5, 4, Subtract); // 1

DoOperation(5, 4, Multiply); // 20

void DoOperation(int a, int b, Operation op)

{

Console.WriteLine(op(a, b));

}

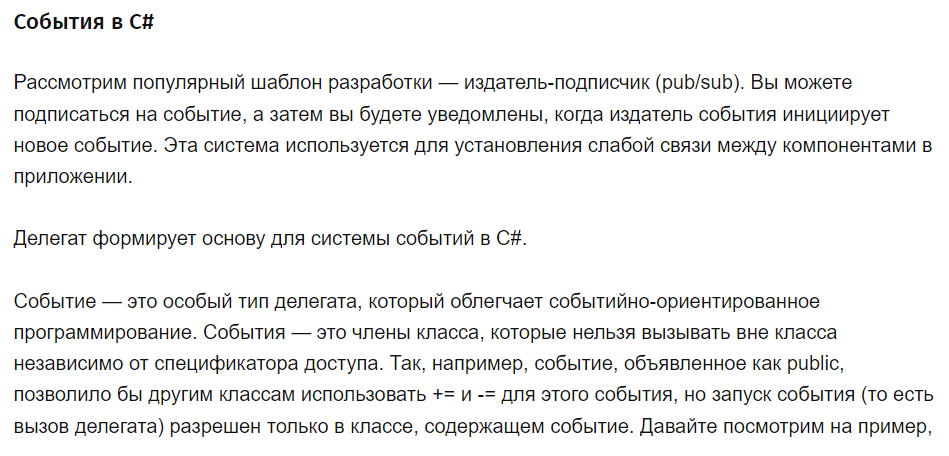
int Add(int x, int y) => x + y;

int Subtract(int x, int y) => x - y;

int Multiply(int x, int y) => x \* y;

delegate int Operation(int x, int y);

СОБЫТИЕ



Тип\_доступа event тип\_делегата название\_ивента = delegate { }

//Определяем класс-издатель как Pub

public class Pub

{

//Свойство OnChange содержит список всех callback-методов подписчиков

public event Action OnChange = delegate {тут};

public void Raise()

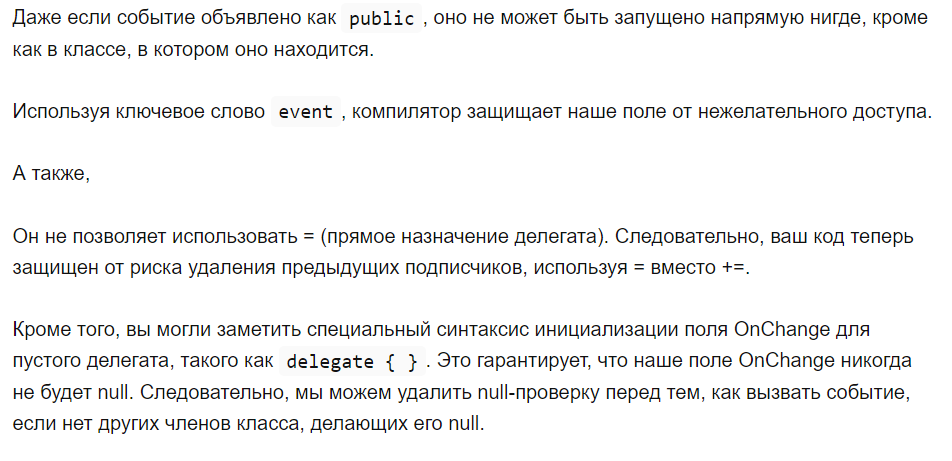
{

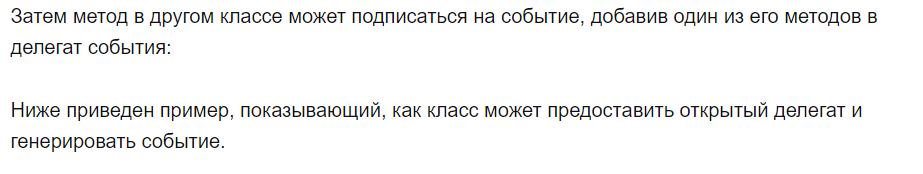
//Вызов OnChange

OnChange();

}

}





class Program

{

static void Main(string[] args)

{

//Инициализируем объект класса pub

Pub p = new Pub();

//подписываем вывод Subscriber 1 на событие OnChange

p.OnChange += () => Console.WriteLine("Subscriber 1!");

//подписываем вывод Subscriber 2 на событие OnChange

p.OnChange += () => Console.WriteLine("Subscriber 2!");

//генерируем событие

p.Raise();

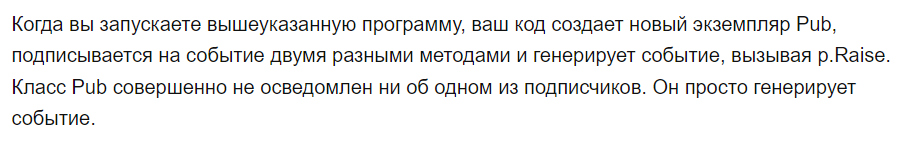
//После вызова метода Raise() все подписанные callback-методы та же будут вызваны

Console.WriteLine("Press enter to terminate!");

Console.ReadLine();

}

}



Другой способ создания событий

События объявляются в классе с помощью ключевого слова event, после которого указывается тип делегата, который представляет событие:

delegate void AccountHandler(string message);

event AccountHandler Notify;

В данном случае вначале определяется делегат AccountHandler, который принимает один параметр типа string. Затем с помощью ключевого слова event определяется событие с именем Notify, которое представляет делегат AccountHandler. Название для события может быть произвольным, но в любом случае оно должно представлять некоторый делегат.

Определив событие, мы можем его вызвать в программе как метод, используя имя события

Notify("Произошло действие");

При null

if(Notify !=null) Notify("Произошло действие");

или

Notify?.Invoke("Произошло действие");

Добавление ивента

Notify += обработчик события;

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. Например:

class Account

{

public delegate void AccountHandler(string message);

AccountHandler? notify;

public event AccountHandler Notify

{

add

{

notify += value;

Console.WriteLine($"{value.Method.Name} добавлен");

}

remove

{

notify -= value;

Console.WriteLine($"{value.Method.Name} удален");

}

}

public Account(int sum) => Sum = sum;

public int Sum { get; private set; }

public void Put(int sum)

{

Sum += sum;

notify?.Invoke($"На счет поступило: {sum}"); // 2.Вызов события

}

public void Take(int sum)

{

if (Sum >= sum)

{

Sum -= sum;

notify?.Invoke($"Со счета снято: {sum}"); // 2.Вызов события

}

else

{

notify?.Invoke($"Недостаточно денег на счете. Текущий баланс: {Sum}"); ;

}

}

}

Во второй части определяем акссесоры add и remove. Аксессор add вызывается при добавлении обработчика, то есть при операции +=. Добавляемый обработчик доступен через ключевое слово value. Здесь мы можем получить информацию об обработчике (например, имя метода через value.Method.Name) и определить некоторую логику. В данном случае для простоты просто выводится сообщение на консоль:

add

{

notify += value;

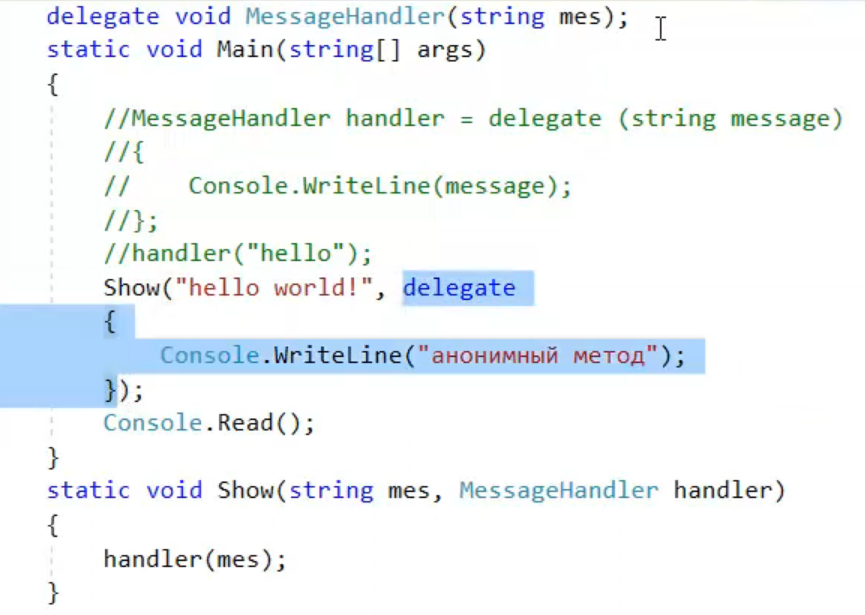
Console.WriteLine($"{value.Method.Name} добавлен");

}

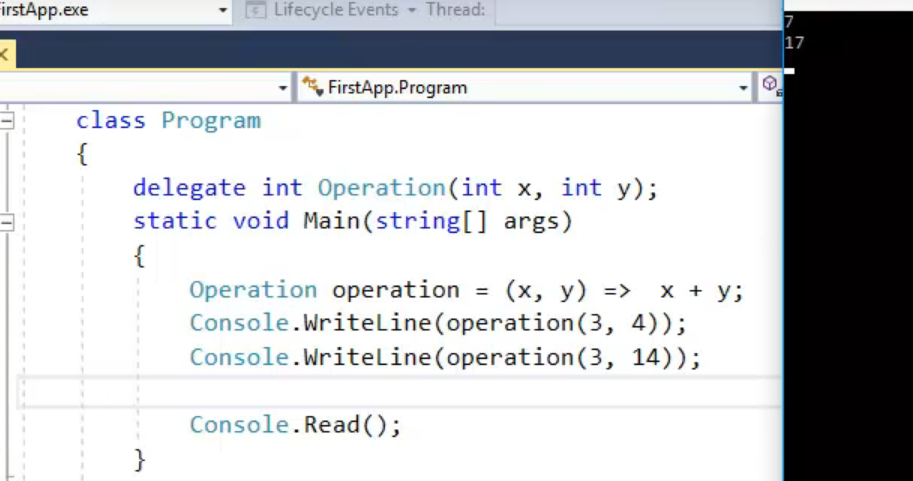
АНОНИМНЫЕ МЕТОДЫ

Анонимный метод используется для инициализации делегата



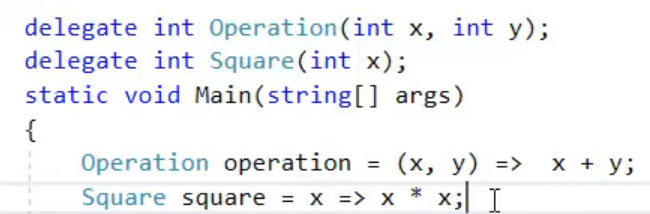


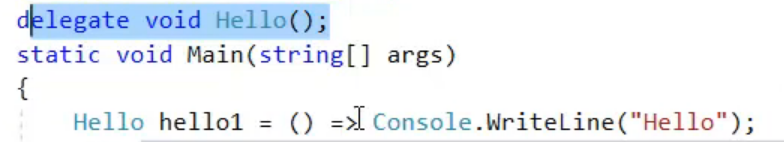
ЛЯМБДА-ВЫРАЖЕНИЕ





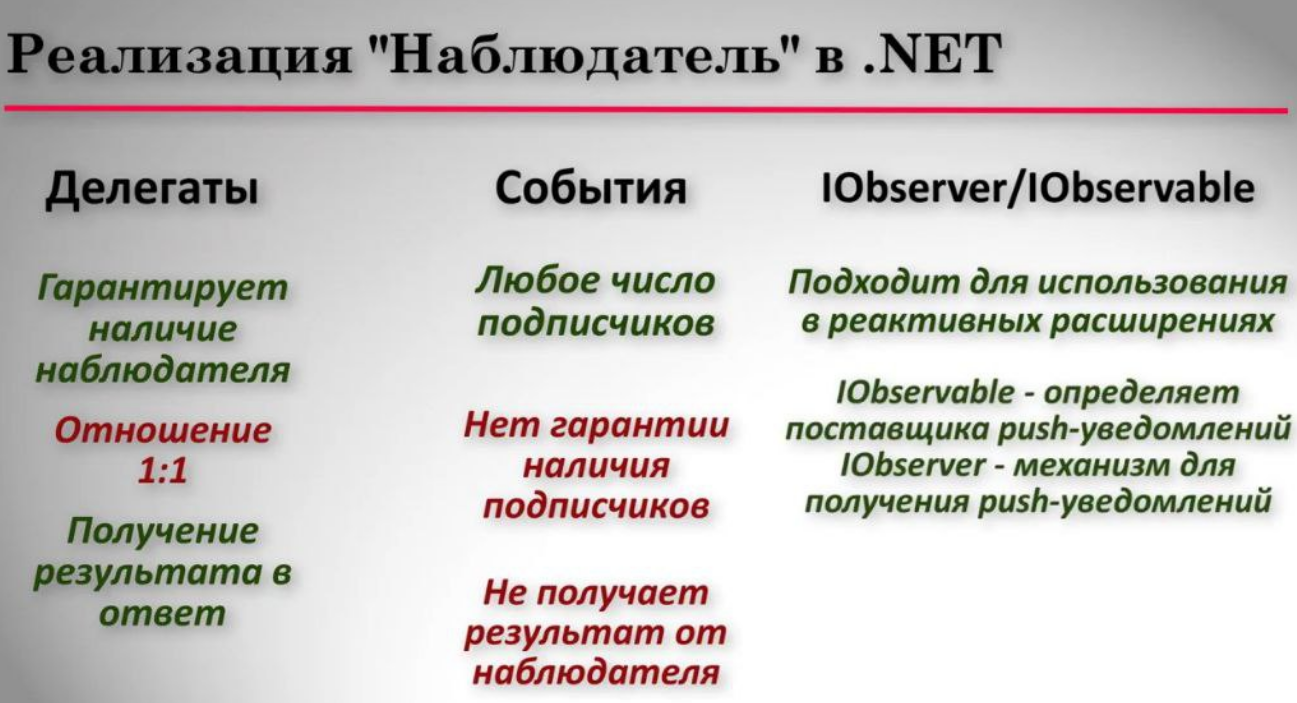
Упрощенная запись анонимных функций

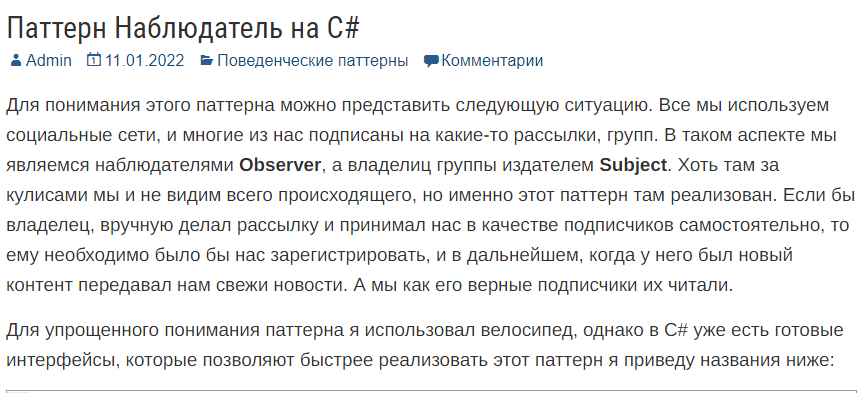


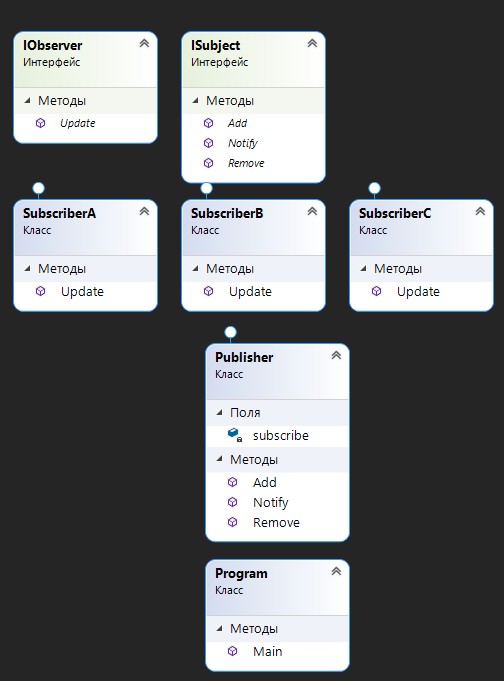


Шаблон “Наблюдатель”

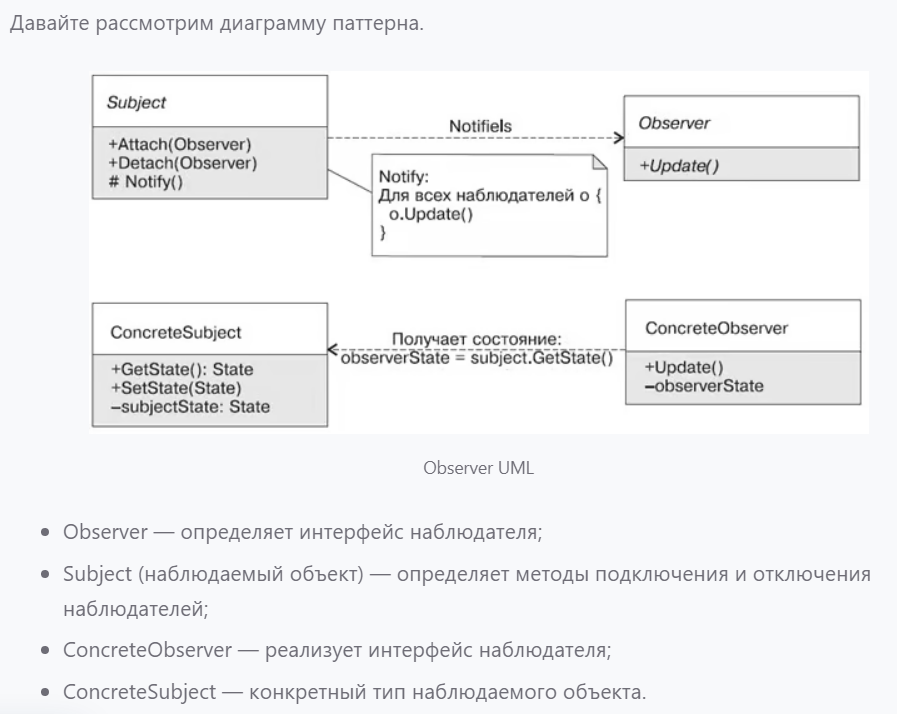
Сделать наблюдателя обязательно

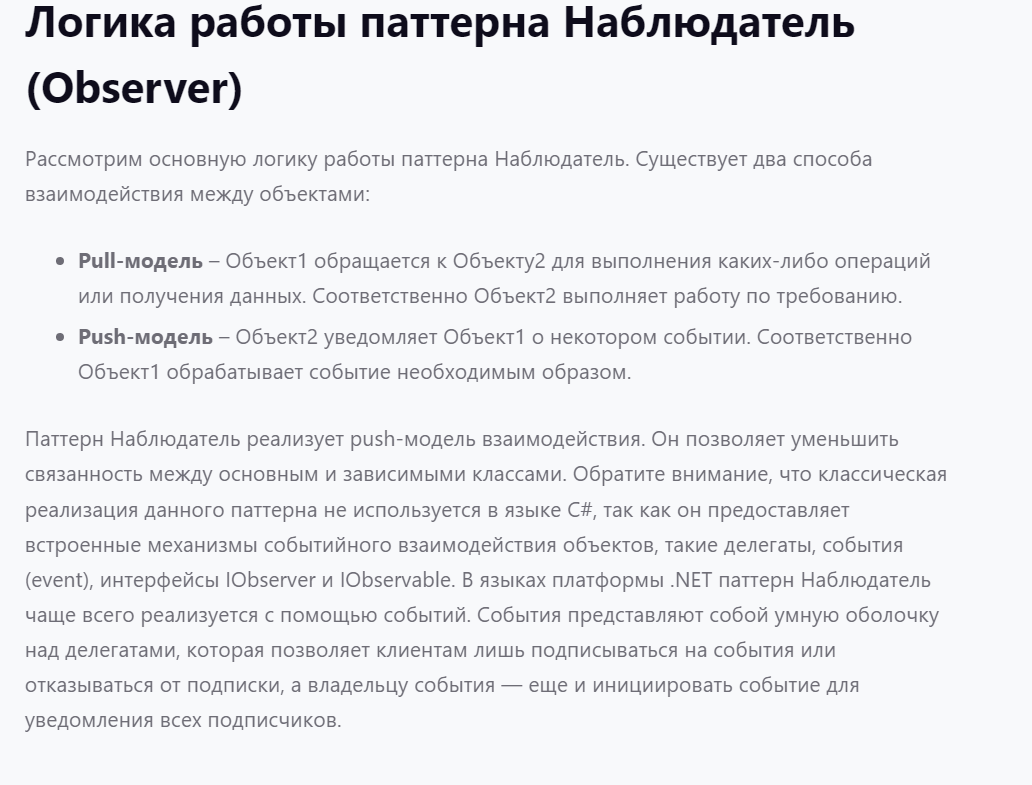




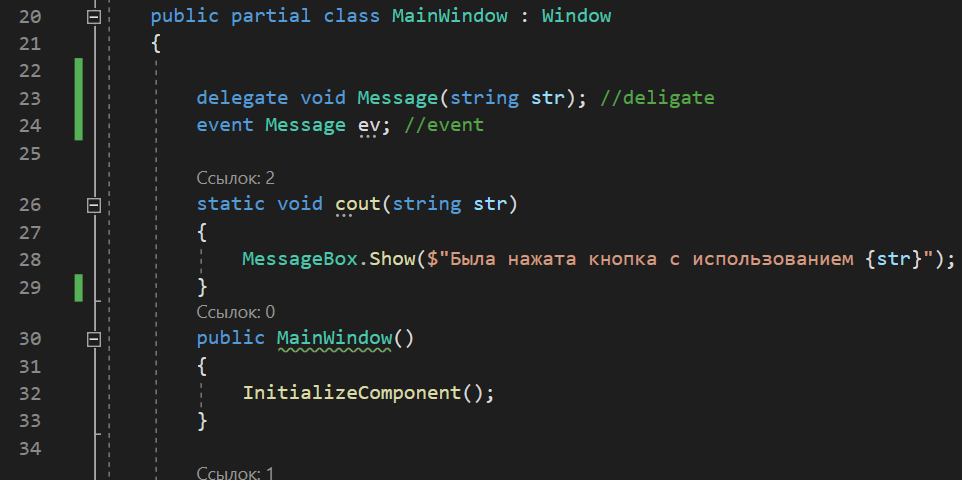


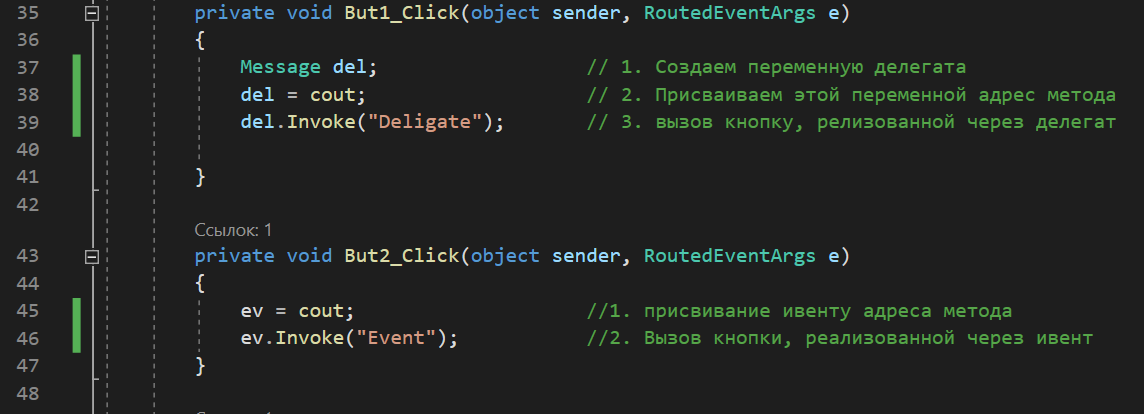
{сделать}

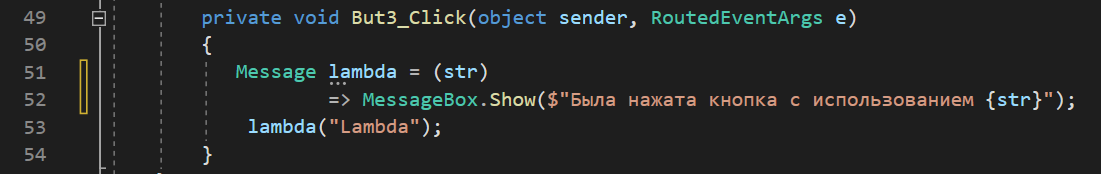




СОБЫТИЙНО-УПРАВЛЯЕМОЕ ПРИЛОЖЕНИЕ = кнопочки, события которых обрабатываются через события (делегаты), применить лямбды







СТАТЬЯ С ВИКИПЕДИИ



