ПИШНИК: BECOME HUMAN

Глава 2: События. Паттерн Генерации Событий.

Автор презентации – Сагалов Даниил БПИ-196

События. Терминология

События позволяют уведомлять другие типы о возникновении различных ситуаций. По сути они являются инкапсулированными делегатами, их можно вызвать только в том типе/экземпляре, в котором их объявили.

Логично предположить, что при работе с событиями на одно событие может быть подписано множество различных подписчиков. Ровно таким же образом один подписчик может обрабатывать множество различных событий. События, у которых нет подписчиков, никогда не возникают – как и в случае с экземплярами делегатов, возникнет **NullReferenceException**. При возникновении события все обработчики события по умолчанию вызываются синхронно.

События внутри класса могут быть помечены как **static** (делает событие доступным без создания экземпляров класса), **abstract/virtual/override/new** (для переопределения при наследовании), а также **sealed** (вместе с override, запрещает последующее переопределение события). Вы не можете пометить события как **readonly**. Ещё одно условие создания событий – их модификаторы доступа должны быть меньше или равны уровню доступа типа делегата.

Алгоритм Генерации События

При работе с событиями следует помнить набор определённых понятий:

- **Издатель (Publisher):** определяет при каких условиях возникает событие.
- **Подписчик (Subscriber):** определяет, каким образом обработать событие.
- **EventArgs**: Базовый класс для создания собственных передаваемых данных события, сам содержит только поле только для чтения Empty, представляющее собой пустые аргументы события.
- **EventHandler**: делегат-тип, имеющий сигнатуру:
  public delegate void EventHandler(object sender, EventArgs e);
  public delegate void EventHandler<T>(object sender, T args);

И так, для того, чтобы сгенерировать событие, вам необходимо:

1) Делегат-тип, используемый для события.
2) Объявление события в классе-издателе.
3) Один или несколько методов, вызывающих событие в классе издателе (или за его пределами, при необходимости).
4) Обработчики данного события в каждом классе, каким-либо образом обрабатывающем данное событие.

```
class Program {
    // Объявляем статическое событие
    static public event EventHandler myEvent;
    Ссылок: 0
    public static void Main() {
        Number[] nums = new Number[2];
        for (int i = 0; i < nums.Length; i++) {
            nums[i] = new Number(i);
            // Подписываем на него чётные объекты Number
            if (nums[i].num % 2 == 0)
                myEvent += nums[i].MyEventHandler;
        }
        // Вы можете подписывать лямбды/анонимные методы
        myEvent += (object sender, EventArgs args) =>
        { Console.WriteLine($"Анонимный метод: {sender.ToString()}"); };
        // Вызываем, есди событие не null
        // Передаём тип вместо null в качестве sender
        myEvent?.Invoke(typeof(Program), EventArgs.Empty);
    }
}

Ссылок: 4
class Number {
    public int num;
    ссылка: 1
    public Number(int number) => num = number;
    ссылка: 1
    public void MyEventHandler(object sender, EventArgs args) {
        Console.WriteLine($"Number с чётным значением: {num}");
        Console.WriteLine($"Событие вызвано {sender.ToString()}");
    }
}
```

Работа с Событиями

Так как события представляют собой инкапсулированные делегаты, подписчики могут подписаться на них при помощи операторов += и -=.

Вы так же можете подписывать анонимные методы или лямбда-выражения на события. Рекомендуется это делать только в случае, если вы не планируете отписывать их в дальнейшем, т. к. единственный способ это сделать – заранее сохранить анонимный метод/лямбда-выражение в переменной делегата и потом отписаться через сам делегат или провести манипуляции со списком вызовов.

В данном случае вы можете увидеть, как можно определить событие, соответствующие рекомендациям .NET (с использованием **EventHandler**).

Использование EventArgs. Пример

На следующих слайдах будет представлен пример работы с событиями, основывающийся на использовании стандартного шаблона событий.

Для этого мы создадим класс **GhostSpottedEventArgs – наследник EventArgs** с передаваемой информацией о призраке – сообщение будет содержать информацию о типе, имени и показателе того, насколько он страшный.

Внутри класса **Ghost** определено событие, использующее делегат **EventHandler<GhostSpottedEventArgs>,** т. е. ему будут соответствовать методы с сигнатурой **void <Имя Метода>(object sender, GhostSpottedEventArgs args)**.

Событие призрака будет вызываться в методе F**righten( )** только в том случае, если призрак попытается напугать объект типа GhostBuster.

Объект **GhostBuster** будет обрабатывать полученные данные события и выводить соответствующую информацию о встретившимся призраке.

Пример

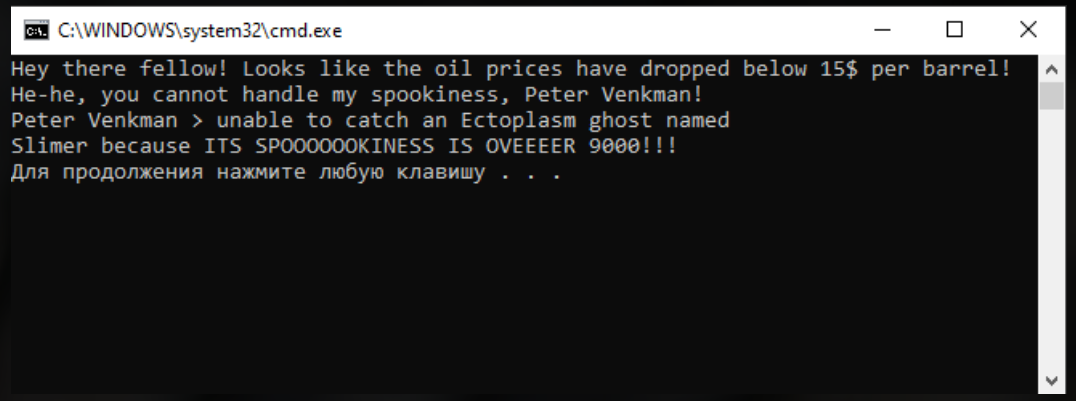
```
// Создаём объект - информацию о призраке
Ссылок: 4
public class GhostSpottedEventArgs : EventArgs
{
    Ссылок: 2
    public string GhostMessage { get; set; }
    ссылка: 1
    public GhostSpottedEventArgs(string message) => GhostMessage = message;
}
```

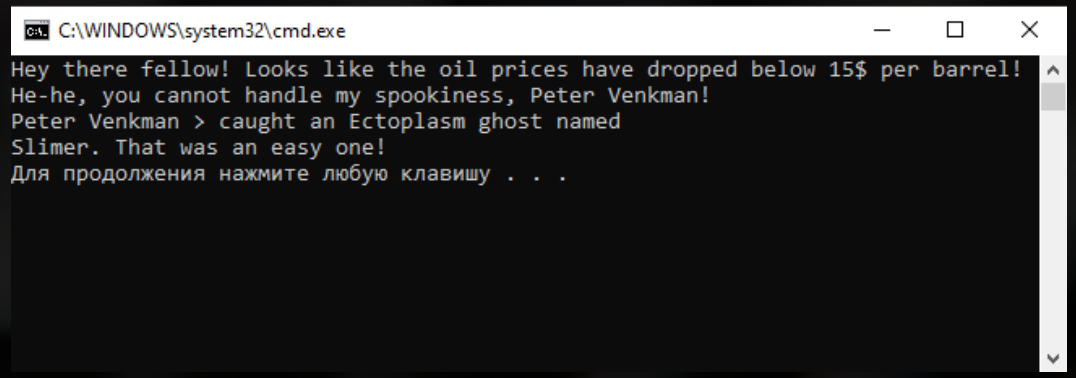
```
public class Ghost
{
    // Случайный генератор значения "страшности" призрака
    private Random spookyRandom;
    // Публичное событие призрака - использует делегат EventHandler<>
    public event EventHandler<GhostSpottedEventArgs> thereIsSomethingStrange;
    // Заметьте, что все автореализуемые свойства по умолчанию приватны
    // Это сделано специально, чтобы информацию о призраке можно было
    // узнать только из сообщения события.
    Ссылок: 2
    private string Name { get; set; }
    Ссылок: 2
    private string Type { get; set; }
    Ссылок: 2
    private ulong Spookiness { get; set; }
    ссылка: 1
    public Ghost(string name, string type)
    {
        Name = name;
        Type = type;
        spookyRandom = new Random();
        Spookiness = (ulong)spookyRandom.Next(5000, 10000);
    }
    // Метод "пугания" - призрак выдаёт страшную фразочку, если объект не GhostBuster,
    // в противном случае вызывается событие thereIsSomethingStrange.
    Ссылок: 2
    public void Frighten(Object target)
    {
        if (target is GhostBuster buster)
        {
            Console.WriteLine($"He-he, you cannot handle my spookiness, {buster.Name}!");
            thereIsSomethingStrange?.Invoke(this,
                // Генерация объекта данных события с информацией о призраке
                new GhostSpottedEventArgs($"{Type},{Name},{Spookiness}"));
        }
        else
            Console.WriteLine("Hey there fellow! Looks like the oil " +
                "prices have dropped below 15$ per barrel!");
    }
}
```

```
public class GhostBuster
{
    // Класс охотника за привидениями содержит обработчик события призрака
    // Данный обработчик обрабатывает данные призрака и в зависимости от их
    // значений выдаёт соответствующее сообщение
    Ссылок: 4
    public string Name { get; private set; }
    ссылка: 1
    public GhostBuster(string name) => Name = name;

    ссылка: 1
    public void CatchTheGhost(object sender, GhostSpottedEventArgs args)
    {
        if (sender is Ghost ghost)
        {
            string[] ghostInfo = args.GhostMessage.Split(',');
            if (int.Parse(ghostInfo[2]) > 9000)
            {
                Console.WriteLine($"{Name} > unable to catch an {ghostInfo[0]} ghost named ");
                Console.WriteLine($"{ghostInfo[1]} because ITS SPOOOOOOKINESS IS OVEEEER 9000!!!");
            }
            else
            {
                Console.WriteLine($"{Name} > caught an {ghostInfo[0]} ghost named");
                Console.WriteLine($"{ghostInfo[1]}. That was an easy one!");
            }
        }
    }
}
```

```
class Program {
    Ссылок: 0
    static void Main() {
        // Создаём призрака и охотника за привидениями
        Ghost ghost = new Ghost("Slimer", "Ectoplasm");
        GhostBuster buster = new GhostBuster("Peter Venkman");
        // Не забываем подписать охотника на событие призрака
        ghost.thereIsSomethingStrange += buster.CatchTheGhost;
        // Пробуем напугать 2 объекта (событие сработает только
        // при попытке напугать охотника)
        ghost.Frighten(new DateTime());
        ghost.Frighten(buster);
    }
}
```





Пример

Задачи

Задание 1

В результате выполнения фрагмента программы:

```
using System;

class Program {
    delegate void Del();
    public static event Del MyDel;
    static void Main() {
        int x = 3;
        MyDel = delegate {
            for (int i = 0; i < 5; i++) {
                x *= i + i;
                Console.Write(x * i);
            }
        };
        MyDel();
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задание 2

Выберите верные утверждения (*укажите все верные ответы*)**:**

1) Событие – инкапсулированный делегат.
2) Стандартный шаблон событий имеет следующие входные аргументы: object sender и EventArgs args.
3) Событие может быть аргументом метода.
4) На событие можно подписать как методы экземпляра, так и статические методы.
5) Событие можно запустить только в классе, где это событие объявлено.

Задание 3

В результате выполнения фрагмента программы:

```
using System;

delegate int MyDel(int x);

class A {
    public event MyDel MyDelEventHandler;

    public static int Meth1(int x) {
        return x % 2;
    }

    public static int Meth2(int x) {
        return x % 3;
    }

    public static int Meth3(int x) {
        return x % 4;
    }

    public void Event() {
        Console.WriteLine(MyDelEventHandler(15));
    }
}

class Program {
    static void Main() {
        A a = new A();
        a.MyDelEventHandler += A.Meth1;
        a.MyDelEventHandler += A.Meth2;
        a.MyDelEventHandler += A.Meth3;
        a.Event();
    }
}
```

на экран будет выведено:

Задание 4

В результате выполнения фрагмента программы:

```
using System;

public delegate int[] MyDel(int x);
public class A {
    public event MyDel MyDelEventHandler;

    public static int[] Meth1(int x) {
        int[] args = new int[10];
        for (int i = 1; i < args.Length; i++) {
            args[i] = args[i - 1] + x * 2;
        }
        return args;
    }

    public int[] Meth2(int x) {
        int[] args = new int[5];
        for (int i = 0; i < args.Length; i++) {
            args[i] = x + i * (int)Math.Truncate(10.31 % 3);
        }
        return args;
    }

    public void Event() {
        int[] args = MyDelEventHandler?.Invoke(5);
        foreach (var item in args) {
            Console.Write(item);
        }
    }
}

class Program {
    static void Main() {
        A a = new A();
        a.MyDelEventHandler += A.Meth1;
        a.MyDelEventHandler += a.Meth2;
        a.Event();
    }
}
```

на экран будет выведено:

Примечание:

Если возникнет ошибка компиляции, введите: ***

Если ошибок и исключений нет, но на экран не выведется ничего, введите: ---

Если возникнет ошибка исполнения или исключение, введите: +++

Задачи

Задание 5

В результате выполнения фрагмента программы:

```
using System;

delegate void Del(ref int d);
class Program {
    static event Del onRun;
    static void Main() {
        onRun += new Del(One);
        onRun += new Del(Two);
        int z = 5;
        while (++z < 7) {
            onRun(ref z);
            Console.Write(z);
        }
    }
    static void One(ref int x) {
        Console.Write(x += 3);
    }
    static void Two(ref int y) {
        Console.Write(y += 5);
    }
}
```

на экран будет выведено:

Задание 6

Допустимы следующие модификаторы для событий (*укажите все верные ответы*)**:**

1) extern.
2) static.
3) abstract.
4) virtual.
5) override.

Ответы

Ответ 1: ***

Делегат-тип приватен, однако само событие публично, ошибка компиляции.

Ответ 2: **12345**

Ответ 3: **3**

Ответ 4: **56789**

Ответ 5: **91414**

Ответ 6: **12345**

Настраиваемые События

Вы можете создавать настраиваемые события при помощи контекстно-ключевых слов **add** и **remove**. Такие события по синтаксису похожи на свойства:

[<Атрибуты>] <модификаторы> event <Делегат-Тип> <Имя События> { add {…} remove {…} }

Таким образом вы можете инкапсулировать события: например, на события смогут подписаться только объекты определённого типа или возможность отписаться будет только при определённых условиях.

В отличие от свойств для таких событий есть дополнительные ограничения: всегда должны одновременно быть оба метода доступа add и remove, нельзя указать модификаторы данных аксессоров, а так же нельзя сделать их автоматически реализуемыми или абстрактными.

Пример:

```
private event Func<int> hiddenEvent;
public event Func<int> HiddenEvent {
        add => hiddenEvent += value;
        remove => hiddenEvent -= value;
}
```

Настраиваемые События Пример

На следующих слайдах будет представлен ещё один пример работы с событиями. Данный пример показывает, как можно использовать настраиваемые события для взаимодействия издателя с подписчиками. На этот раз мы переместимся в Средиземье:

В классе **Wizard** определено инкапсулированное приватное событие **Func<Orc, int> somethingHasChangedInTheAirAgain**. Через методы доступа **add** и **remove** задаются ограничения: на событие могут подписаться только объекты класса **Character** или его производных классов, причём отписаться они могут только если подписанный метод вернёт значение меньше 100 (в качестве параметра Orc при данной проверке передаётся null). Волшебник активирует событие в методе **SmokeThePipe(Orc orc)**, указывая всем подписчикам на объект типа Orc. При подписке/отписке волшебник даёт соответствующие комментарии.

В классе **Orc** описано поведение орка в битве, а именно его метод защиты **Defend(int enemyAttack)**.

У абстрактного **Character** определены 2 наследника – **Gnome** и **Hobbit**. В них определён метод обработчик события волшебника int **FightTheEvil(Orc orc)**, осуществляющий сражение с выбранным орком.

В основной программе симулируется процесс сражения с орком – битва происходит до тех пор, пока здоровье орка не опустится до нуля. При данной реализации гномы и хоббит пытаются отписаться от события волшебника каждый ход, что успешно происходит, если их здоровье опустится меньше 100.

```
public class Wizard {
    // Объявляем закрытое событие
    private event Func<Orc, int> somethingHasChangedInTheAirAgain;
    // Список подписчиков события, обновляется при добавлении/удалении подписчиков
    private Delegate[] members;
    // Настраиваемое событие с аксессорами add и remove
    public event Func<Orc, int> SomethingHasChangedInTheAirAgain {
        add {
            // На событие могут подписываться только объекты типа Character и их наследники
            if (!(value.Target is Character))
                Console.WriteLine("Only living being are allowed to join our journey!");
            else {
                // При подписке обновляется список вызовов
                Console.WriteLine($"Welcome, dear {value.Target.ToString()}!");
                somethingHasChangedInTheAirAgain += value;
                members = somethingHasChangedInTheAirAgain.GetInvocationList();
            }
        }
        remove {
            // Если объект содержится в списке подписчиков, он может отписаться только в случае,
            // когда его подписанный метод вернёт значение меньшее или равное 100
            if (members.Contains(value)) {
                if (value?.Invoke(null) > 100)
                    Console.WriteLine($"Gendalf > It is not time to give up yet, {value.Target.ToString()}.");
                else {
                    Console.WriteLine($"It's time to rest and cure your wounds, {value.Target.ToString()}.");
                    somethingHasChangedInTheAirAgain -= value;
                    // При отписке обновляется список вызовов
                    members = somethingHasChangedInTheAirAgain.GetInvocationList();
                }
            }
        }
    }
    ссылка: 1
    public void SmokeThePipe(Orc orc) {
        if (orc != null) {
            Console.WriteLine("Gendalf > The orc is still standing in our way! Prepare to defend!");
            somethingHasChangedInTheAirAgain?.Invoke(orc);
        }
    }
    Ссылок: 3
    public override string ToString() => "Wizard Gendalf";
}
```

Пример

Пример

```
public abstract class Character
{
    Ссылок: 4
    public string Name { get; private set; }
    protected int health;
    Ссылок: 4
    public int AttackStrength { get; private set; }
    Ссылок: 4
    public abstract int FightTheEvil(Orc orc);
    Ссылок: 2
    protected Character(string name, int health, int attackStrength) {
        Name = name;
        this.health = health;
        AttackStrength = attackStrength;
    }
}
```

```
Ссылок: 2
public class Hobbit : Character
{
    // Благодаря sealed наследники не смогут переопределить данную версию ToString()
    Ссылок: 3
    public sealed override string ToString() => $"Hobbit {Name}";
    Ссылок: 4
    public override int FightTheEvil(Orc orc)
    {
        if (orc != null && orc.Health > 0)
        {
            health -= orc.AttackStrength;
            orc.Defend(AttackStrength);
        }
        return health;
    }
    ссылка: 1
    public Hobbit(string name, int health, int attackStrength)
        : base(name, health, attackStrength) { }
}
```

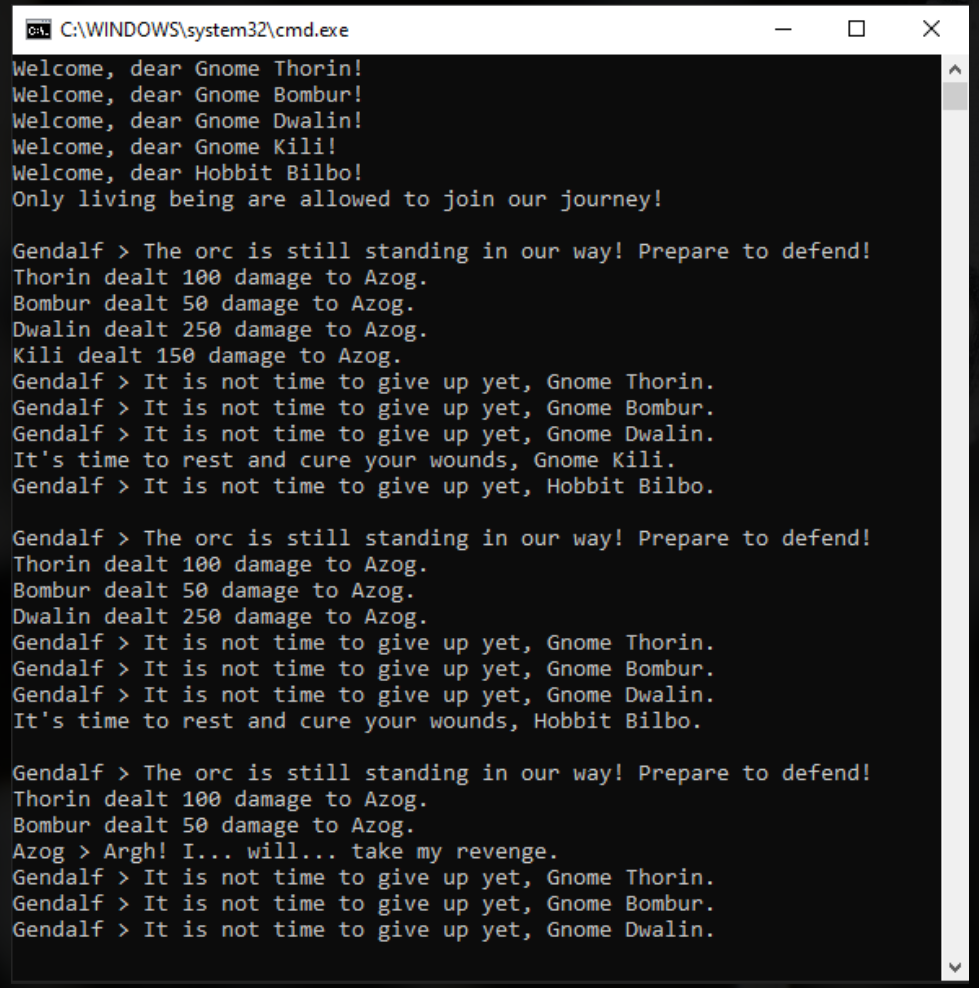
```
Ссылок: 5
public class Gnome : Character
{
    // Благодаря sealed наследники не смогут переопределить данную версию ToString()
    Ссылок: 3
    public sealed override string ToString() => $"Gnome {Name}";
    Ссылок: 4
    public override int FightTheEvil(Orc orc)
    {
        if (orc != null && orc.Health > 0)
        {
            health -= orc.AttackStrength / 2;
            Console.WriteLine($"{Name} dealt {AttackStrength / 2} damage to {orc.Name}.");
            orc.Defend(AttackStrength);
        }
        return health;
    }
    Ссылок: 4
    public Gnome(string name, int health, int attackStrength)
        : base(name, health, attackStrength) { }
}
```

Пример

```
class Program {
    Ссылок: 0
    static void Main() {
        Random rnd = new Random();
        Wizard wizard = new Wizard();
        Orc orc = new Orc("Azog", 1200, 500);
        Character[] characters = new Character[5];
        // Заполняем массив весьма известными гномами и хоббитом
        characters[0] = new Gnome("Thorin", 2800, 200);
        characters[1] = new Gnome("Bombur", 4000, 100);
        characters[2] = new Gnome("Dwalin", 1000, 500);
        characters[3] = new Gnome("Kili", 300, 300);
        characters[4] = new Hobbit("Bilbo", 620, 100);
        // Подписываем всех персонажей на событие Гендальфа
        Array.ForEach(characters, character => wizard.SomethingHasChangedInTheAirAgain += character.FightTheEvil);
        // NaratorOrcDescription не подпишется на событие - оно статическое и не является частью Character
        wizard.SomethingHasChangedInTheAirAgain += NarratorOrcDescription;
        Console.WriteLine();
        // Сражаемся с орком и каждый раз пытаемся выйти из боя.
        while (orc.Health > 0) {
            wizard?.SmokeThePipe(orc);
            Array.ForEach(characters, character => wizard.SomethingHasChangedInTheAirAgain -= character.FightTheEvil);
            Console.WriteLine();
        }
    }

    ссылка: 1
    static int NarratorOrcDescription(Orc orc) {
        Console.WriteLine($"And now the brave gnomes are about to fight {orc.Name} with {orc.Health} health!");
        return orc.Health;
    }
}
```

Результат работы && Финальный Вопрос



В результате выполнения фрагмента программы:

```
using System;

public static class EventHolder {
    private static event Predicate<int> predictionEvent;
    public static event Predicate<int> PredictionEvent {
        add {
            if (value?.Invoke(10) ?? true) {
                predictionEvent += value;
                Console.Write(predictionEvent?.GetInvocationList().Length);
            }
        }
        remove {
            predictionEvent -= value;
            Console.Write(predictionEvent?.GetInvocationList().Length);
        }
    }
}

public class Program {
    static void Main() {
        EventHolder.PredictionEvent += (x) => x.CompareTo(5) > 0;
        EventHolder.PredictionEvent += (x) => x.CompareTo(20) > 0;
        EventHolder.PredictionEvent -= (x) => false;
        EventHolder.PredictionEvent += null;
        EventHolder.PredictionEvent += (x) => x.CompareTo(9) > 0;
        EventHolder.PredictionEvent -= (x) => x.CompareTo(9) > 0;
    }
}
```

на экран будет выведено:

ОТВЕТ:

11122