

Лабораторна робота №13

Створення і обробка подій

Мета роботи: закріпити практичні навички і теоретичні знання по роботі з делегатами та подіями.

Методичні рекомендації

Подія - це ситуація, при виникненні якої, відбудеться дія або кілька дій. Іншими словами подія - це іменований делегат, при виклику якого, будуть запущені всі підписані на момент виклику події методи заданої сигнатури.

Робота з подіями здійснюється в C# відповідно до моделі «видавець-передплатник». Клас, відповідальний за ініціалізацію подій публікує подію; будь-які класи можуть підписатися на цю подію.

При виникненні події середовище виконання повідомляє всіх передплатників про те, **що подія відбулася**. При цьому викликаються відповідні **методи-обробники подій передплатників**. Який обробник події буде викликаний - визначається делегатом.

Події є членами класу і оголошуються з використанням ключового слова *event*.

Синтаксис: event ім'яДелегату ім'яОб'єкту;

Приклад 1. Є три класи. Перший клас буде рахувати до 100, використовуючи цикл. Два інших класи чекатимуть, коли у першому класі лічильник дорахує, наприклад, до 80, і після цього кожен виведе в консоль фразу «80!».

Простіше кажучи, при виявленні значення 80, викликаються методи кожного класу.

Перший клас *Counter* і його метод *Count()* в якому буде проводиться рахунок.

Два інших класу (*HandlerOne*, *HandlerTwo*), які повинні реагувати на виникнення події методами *Message()*.

Файл Counter.cs

.....

```
class Counter
{
    public delegate void MethodContainer();
    public event MethodContainer onNumberGenerated;
    public void Count()
    {
        int c;
        Random rand = new Random();
        for (int i = 0; i < 10000; i++)
        {
            c = rand.Next(100);
            if (c == 80)           //подія випадкове число - 80
            {
                onNumberGenerated();    //запуск обробників події
                                     // onNumberGenerated
            }
        }
    }
}
```

```
    }  
}
```

.....

Файл HandlerOne.cs

```
class HandlerOne  
{  
    public void Message()  
    {  
        Console.ForegroundColor = ConsoleColor.Green;  
        Console.WriteLine("HandlerOne. The number 80 was generated!");  
        Console.ResetColor();  
    }  
}
```

Файл HandlerTwo.cs

```
class HandlerTwo  
{  
    public void Message()  
    {  
        Console.ForegroundColor = ConsoleColor.White;  
        Console.WriteLine("HandlerTwo. The number 80 was generated!");  
        Console.ResetColor();  
    }  
}
```

Файл Program.cs

```
static void Main(string[] args)  
{  
    Counter counter = new Counter();           //створюємо екземпляр класу лічильник  
    HandlerOne handlerOne = new HandlerOne(); //створюємо екземпляр класу HandlerOne  
    HandlerTwo handlerTwo = new HandlerTwo();  //створюємо екземпляр класу HandlerTwo  
  
    counter.onNumberGenerated += handlerOne.Message; //Підпис на подію  
    counter.onNumberGenerated += handlerTwo.Message;  
  
    counter.Count();                             //Запуск лічильника  
    Console.ReadKey();  
}
```

Завдання. //точка в круг - перша подія, інша зміна надпису лічильника

Відбувається це наступним чином:

<КласАбоОб'єкт>.<Ім'яПодії>+=

<КласЧийМетодПовиненЗапуститися>.<Мет одЩоПідходитьЗаСигнатурою>.

Порядок створення події та її обробки:

1. Визначте умову виникнення події і методи, які повинні спрацювати.
2. Визначте сигнатуру цих методів і створіть делегат на основі цієї сигнатури.
3. Опишіть загальнодоступну подію на основі цього делегата і викличте її обробники, коли умова спрацює.

4. Обов'язково підпишіться на цю подію тими методами, які повинні спрацювати і сигнатури яких відповідають делегату.

Клас, в якому створюється подія (генерується) називається *класом-видавцем*, а класи, чії методи підписуються на цю подію за допомогою "+ =" - *класами-передплатниками*.

Запам'ятайте! Якщо Ви не підписалися на подію і його делегат порожній, виникне помилка.

Щоб уникнути цього, необхідно підписатися, або не викликати подію взагалі, для наведеного прикладу це буде наступним чином:

```
if (c == 80)
{
    if (onNumberGenerated != null)
    {
        onNumberGenerated();
    }
}
```

Події широко використовуються для складання власних компонентів управління (кнопок, панелей, тощо).

Приклад 2. Широкомовні події

Події можуть активізувати кілька обробників, в тому числі ті, що визначені в інших об'єктах. Такі події називаються ширококомовними. Широкомовні події створюються на основі багатоадресних делегатів.

```
using System;
namespace Lab132Example2
{
    static class Handlers
    {
        public static void handler()
        {
            Console.WriteLine("Подія. отримано об'єктом класу EventDemo. ");
        }
    }
    class X
    {
        public void Xhandler()
        {
            Console.WriteLine("Подія отримано об'єктом класу X. ");
        }
    }
    class Y
    {
        public void Yhandler()
        {
            Console.WriteLine("Подія отримано об'єктом класу Y. ");
        }
    }
}

namespace Lab132Example2
{

```

```

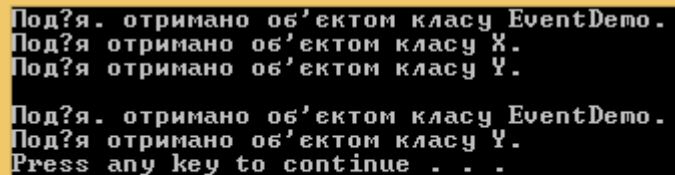
class MyEvent // Оголошення класу, в якому зініціюється подія.
{
    public event MyEventHandler activate;
    // ініціалізація події.
    public void fire()
    {
        if (activate != null) activate();
    }
}

using System;
namespace Lab132Example2
{
    delegate void MyEventHandler();
    class Program
    {
        public static void Main()
        {
            MyEvent evt = new MyEvent();

            X xOb = new X();
            Y yOb = new Y();

            // Додавання методів різних класів handler(), Xhandler() і Yhandler()
            // в ланцюжок обробників події
            evt.activate += new MyEventHandler(Handlers.handler);
            evt.activate += new MyEventHandler(xOb.Xhandler);
            evt.activate += new MyEventHandler(yOb.Yhandler);
            evt.fire();
            Console.WriteLine();
            evt.activate -= new MyEventHandler(xOb.Xhandler);
            evt.fire();
        }
    }
}

```



```

Под?я. отримано об'єктом класу EventDemo.
Под?я отримано об'єктом класу X.
Под?я отримано об'єктом класу Y.

Под?я. отримано об'єктом класу EventDemo.
Под?я отримано об'єктом класу Y.
Press any key to continue . . .

```

Приклад 3. Генератор події після генерації першої події генерує наступні події тільки в тому випадку, якщо приймач події повідомляє, що повідомлення прийнято.

```

using System;
namespace Lab132Example3
{
    class EventGeneration
    {
        public delegate void EventHandler(object sender, EventArgs e);
        public event EventHandler OnEventHandler;

        int eventAmount;
        bool index = false;

        public int EventAmount
        {
            set
            {
                eventAmount = value;
            }
        }
        public bool Index
        {
            set
            {
                index = value;
            }
        }
    }
}

```

```

    }
}

public void NewEvent()
{
    EventArgs e = new EventArgs();
    if (OnEventHandler != null)
    {
        Console.WriteLine("Перша подія згенерована");
        OnEventHandler(this, e);
    }
    if (index == true)
    {
        Console.WriteLine("Повідомлення отримано");
        for (int i = 0; i < eventAmount; i++)
        {
            Console.WriteLine("Подія {0} згенерована", i + 2);
            OnEventHandler(this, e);
        }
    }
}
}
}
}
}

```

```

using System;
namespace Lab132Example3
{
    class Receiver
    {
        void OnHandler(object source, EventArgs e)
        {
            EventGeneration a = source as EventGeneration;
            a.Index = true;
        }
        public Receiver(EventGeneration genEvent)
        {
            genEvent.OnEventHandler += new EventGeneration.EventHandler(OnHandler);
        }
    }
}

```

```

using System;
namespace Lab132Example3
{
    class Program
    {
        static void Main(string[] args)
        {
            EventGeneration genEvent = new EventGeneration();
            Receiver recEvent = new Receiver(genEvent);

            Console.Write("Кількість повідомлень, які потрібно згенерувати: ");
            genEvent.EventAmount = Convert.ToInt32(Console.ReadLine());
            genEvent.NewEvent();

            Console.ReadKey(true);
        }
    }
}

```

```

    }
}
}

```

Зміст роботи

Завдання 1. Створити подію на основі делегату (попередня лабораторна робота).

Завдання 2. Розробити застосування «Назва автомобіля», за допомогою якого можна відстежувати зміни в назві автомобіля. Обробник події повинен видавати повідомлення "Назва змінилася!" При внесенні зміни назви автомобіля.

Один з варіантів виконання програми представлений на рис. 1.



Рис. 1 - Результат виконання програми «Назва автомобіля»

Завдання 3. Модифікувати додаток «Назва автомобіля» таким чином, щоб користувачеві дати можливість прийняти рішення про зміну або відмову від зміни назви автомобіля.

Один з варіантів виконання програми представлений на рис. 2. При натисканні клавіші <N> має видаватися повідомлення: «Назва не змінилася!»

Назва автомобіля - Ford »

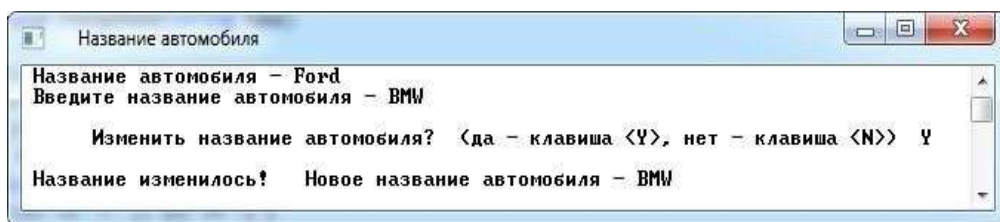


Рис. 2 - Результат виконання модифікованого додатка «Назва автомобіля»

1	<p>Створити програмний додаток, в якому генератор події «забезпечує» подію наступною інформацією: назва поїзда, час прибуття, номер вагона і місця.</p> <p>Приймач події виводить цю інформацію на екран.</p>
---	---

Завдання 4.

2	<p>Створити програмний додаток, в якому генератор події «забезпечує» подію наступною інформацією: назва поїзда, станція призначення, станція відправлення і час у дорозі.</p> <p>Приймач події виводить цю інформацію на екран.</p>
3	<p>Створити програмний додаток, в якому генератор події після генерації першої події генерує наступні події тільки в тому випадку, якщо приймач події повідомляє, що повідомлення доставлене («квитирование»).</p> <p>Для «квитирования» використовувати перший параметр обробника події.</p>
4	<p>Створити програмний додаток, в якому генератор події після генерації першої події генерує певну кількість подій. Кількість генерацій визначається шляхом повідомлення з боку приймача.</p> <p>Для повідомлення використовувати перший параметр обробника події.</p>
5	<p>Створити програмний додаток, в якому генератор події після генерації першої події генерує наступні події тільки в тому випадку, якщо приймач події повідомляє, що подія прийнята (квитирование).</p> <p>Для «квитирования» використовувати другий параметр обробника події.</p>
6	<p>Створити програмний додаток, в якому генератор події після генерації першої події генерує певну кількість подій. Кількість генерацій визначається шляхом повідомлення з боку приймача.</p> <p>Для повідомлення використовувати другий параметр обробника події.</p>
7	<p>Створити програмний додаток, в якому генератор події може генерувати три різних події. Приймачі подій виступають в якості абонентів поштового відділення і можуть пересилати один одному інформацію, використовуючи генератор в якості поштової скриньки. При цьому вони вказують номер (від 1 до 3) наступного приймача і деяке ціле число, яке передається одержувачу. Такий цикл передачі триває до тих пір, поки</p>
	<p>будь-якої з приймачів в якості отримувача не вкаже нуль. У цьому випадку додаток завершує свою роботу. При запуску програми перші поштові повідомлення завжди отримує від генератора перший приймач. Для адресації і передачі інформації використовувати другий аргумент обробника події.</p>

8	Виконати завдання з пункту 7, але для адресації і передачі інформації використовувати перший аргумент обробника події.
9	Створити програмний додаток, в якому генератор події, шляхом генерації однієї події запитує у трьох приймачів деякий ресурс. Кожен приймач повідомляє, скільки ресурсу він може виділити. Для передачі інформації використовувати другий аргумент обробника події.
10	Створити програмний додаток, в якому генератор події, шляхом генерації одної події запитує у трьох приймачів деякий ресурс. Кожен приймач повідомляє, скільки ресурсу він може виділити. Для передачі інформації використовувати перший аргумент обробника події.
11	Створити програмний додаток, в якому генератор події «забезпечує» подію наступною інформацією: назва групи, номер пари, прізвище викладача. Приймач події виводить цю інформацію на екран.
12	Створити програмний додаток, в якому генератор події «забезпечує» подію наступною інформацією: назва товару, назва виробника, кількість, ціна. Приймач події виводить цю інформацію на екран.
13	Створити програмний додаток, в якому публікувати події, що виникають при зміні товарних запасів внаслідок закупівлі або продажі.
14	Створити клас Cat і подію, що виникає при кожній черговій десятці спійманих мишок.
15	Створити програмний додаток, в якому генератор події після генерації першої події (розрахунок довжини кола) генерує наступну подію (розрахунок площі кола), після повідомлення з боку приймача.

Контрольні запитання:

1. Чи можна по сигнатурі оголошення делегата визначити сигнатуру функції, яку представляє делегат.
2. Які обмеження накладаються на функції, які може представляти багатоадресний делегат?

3. Як включити або виключити задану функцію зі списку функцій, які подаються багатоадресних делегатом?
4. Як оголошується подія?
5. Що таке подія?
6. Чим відрізняється подія від багатоадресного делегата?
7. Яка загальноприйнята сигнатура обробника події?
8. Як здійснюється генерація події?
9. Яким чином оброблювачу події передається додаткова інформація про подію, що відбулася?
10. Як здійснюється «підписка» на подію?

Література

Джон Скит С# для професіоналов: тонкості програмування, 3-е издание, Діалектика Вільямс, 2017, 608 стр.

<https://docs.microsoft.com/en-us/dotnet/standard/events/how-to-raise-and-consume-events>