# Практична робота 14. Події

**Мета**. Вивчити та засвоїти методи обробки подій в мові C#

## Теоретичний матеріал

## 14.1. Події і делегати. Взаємодія між подіями

### 14.1.1. Визначення події в C#

**Подія** – це автоматичне повідомлення про те, що в програмі відбулась деяка дія.

На виконання тієї чи іншої події програма має відповідно реагувати. Реакція на подію здійснюється з допомогою так званих обробників події. Обробник події – це звичайний метод, що виконує деякі дії в програмі, у випадку коли відбулась (згенерувалась) подія. Події працюють в поєднанні з *делегатами*. Таке поєднання дозволяє формувати списки (ланцюжки) обробників події (методів), які мають викликатись при виклику (запуску, генеруванні) даної події. Такий підхід є ефективним при написанні великих програмних систем, оскільки він дозволяє впорядкувати великий складний програмний код в якому дуже легко допустити логічну помилку.

На рисунку 1 схематично відображено роботу події MyEvent.

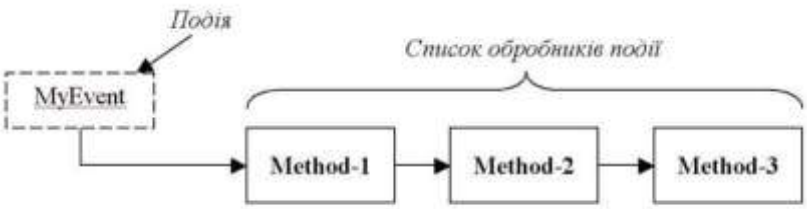



Рис. 1. Виклик ланцюга методів для події MyEvent

### 14.1.2. Загальна форма оголошення події. Ключове слово **event**

Оголошення події здійснюється на основі раніше оголошеного типу делегату. Загальна форма оголошення події:

event ***делегат_події ім'я_події***;

де

- *делегат_події* – ім'я типу делегату, що використовується для підтримки події;
- *ім'я_події* – конкретне ім'я об'єкту (змінної) типу "подія".

**Приклад.** Оголошення події з іменем MyEvent на основі типу делегату MyDelType.

```
event MyDelType MyEvent;
```

Для того, щоб можна було обробляти (запускати) списки обробників події, делегат не повинен повертати значення. Тобто, делегат має повертати тип void.

Якщо делегат повертає значення, то викликається останній метод (обробник) зі сформованого списку методів.

Метод має мати таку саму сигнатуру як і делегат, на основі якого оголошена подія.

Якщо потрібно організувати список методів, що викликаються при виклику події, тоді тип делегату і метод повинні повертати значення void (нічого не повертати).

Якщо є список методів що повертають значення, то при запуску події з цього списку викликається останній метод.

### 14.1.3. Реєстрація методу в події

Щоб зареєструвати метод для обробки даної події потрібно використати оператори '=' або '+='.

**Приклад.** Нехай в тілі деякого класу оголошено:

- подію з іменем MyEvent;
- метод обробки події (обробник події) MyMethod1();
- обробник події MyMethod2();
- обробник події MyMethod3().

Методи мають таку саму сигнатуру, як і тип делегату, на основі якого оголошена подія MyEvent. Щоб зареєструвати методи MyMethod1(), MyMethod2(), MyMethod3() для події MyEvent потрібно написати такий програмний код:

```
// реєстрація методів для події MyEvent
MyEvent = MyMethod1(); // MyEvent => MyMethod1
MyEvent += MyMethod2(); // MyEvent => MyMethod1 -> MyMethod2
MyEvent += MyMethod3(); // MyEvent => MyMethod1 -> MyMethod2 -> MyMethod3
```

У цьому випадку утвориться список (ланцюжок) з трьох методів. Методи в списку йдуть в тій послідовності, в якій вони були зареєстровані (додані) в події.

Якщо методи не повертають ніякого значення, то при виклику події будуть виконуватись усі методи зі списку в порядку їх додавання.

Якщо методи повертають значення, то при виклику події буде виконуватись останній метод зі списку.

### 14.1.4. Організація списку обробників подій

Щоб організувати список обробників події (методів), , що будуть викликатись при виклику події потрібно виконати таку послідовність кроків:

1. Оголосити тип делегату в класі.

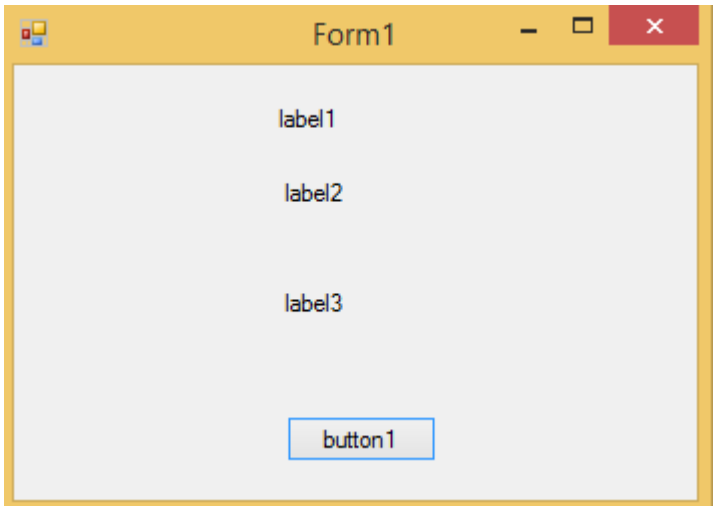
2. Оголосити подію в даному класі або створити інший клас, що містить оголошення події.

3. В деякому методі (програмному коді) створити список обробників (методів), які будуть викликатись при виклику даної події. Це здійснюється з допомогою операторів ‘=’ та ‘+=’. Створення списку означає реєстрацію обробників для даної події.

4. Викликати подію (запустити на виконання) з цього методу.

**Приклад.** Обчислення площ геометричних фігур.

1. Створимо Windows Forms проект з назвою EventExample2 і добавимо на форму керуючі елементи як вказано на малюнку

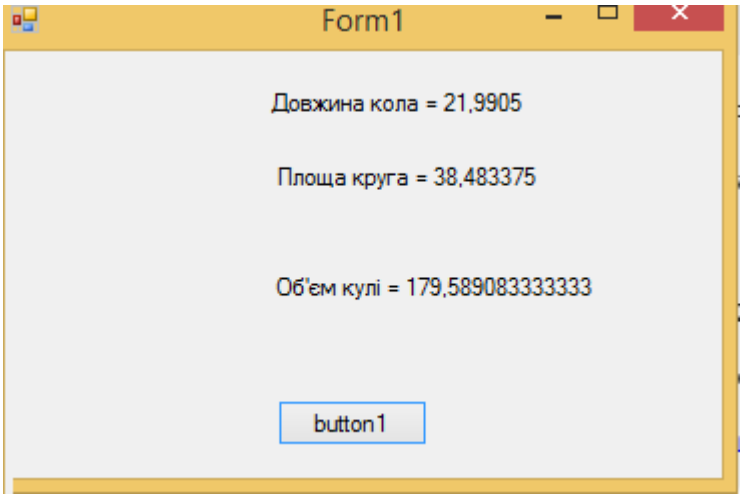



2. Виконаємо подвійний клік на кнопці button1 і в файлі Form1.cs замінимо код на наступний:

```
using System;
using System.Windows.Forms;

namespace EventExample2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        delegate void CalcFigure(double R); // 1. Оголосити тип делегату
        event CalcFigure ECF;               // 2. Оголосити подію з іменем ECF

        // 3. Методи обробки події - розміщуються в цьому ж класі
        //   Методи мають таку саму сигнатуру як тип делегату CalcFigure

        void GetLength(double R) // Довжина кола на основі радіусу R
        {
          double res; const double Pi = 3.1415; res = 2 * Pi * R;
          label1.Text = string.Format("Довжина кола = {0}", res); // вивести на форму
        }
```

```
        void GetArea(double R)    // Площа круга
        {
            double res;
            const double Pi = 3.1415;
            res = Pi * R * R;
            label2.Text = string.Format("Площа круга = {0}",res);
        }

        void GetVolume(double R) // Об'єм кулі
        {
            double res;
            const double Pi = 3.1415;
            res = 4.0 / 3.0 * Pi * R * R * R;
            label3.Text = string.Format("Об\'єм кулі = {0}",res);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // 4. Демонстрація роботи з методами з допомогою події

            // 4.1. Створити ланцюг методів, які будуть викликатись з події ECF
            ECF = GetLength; // ECF => GetLength()
            ECF += GetArea; // ECF => GetLength() -> GetArea()
            ECF += GetVolume; // ECF => GetLength() -> GetArea() -> GetVolume()

            // 4.2. Виклик події ECF з параметром 2.0
            ECF(2.0); // викликаються послідовно три методи GetLength(), GetArea(),
                      // GetVolume()

            // Виклик події для параметру 3.5
            ECF(3.5);
        }
    }
}
```

3. Натиснемо F5 і запустимо код на виконання. Натиснемо кнопку button1 Результат буде наступний:




*Завдання.*

Видаліть в рядку

```
        event CalcFigure ECF;
```

ключове слово `event` і запустіть на виконання. Що змінилось?

### 14.1.5. Використання події, яка оголошена в окремому класі. Обробники події оголошені в іншому класі

1. В файлі CalcFigures.cs оголосимо клас

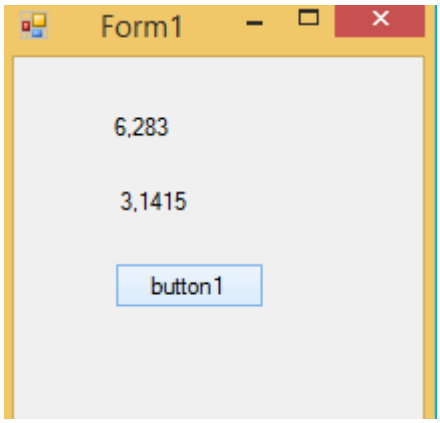
```
namespace EventExample5
{
    class CalcFigures
    {
        // клас, що містить методи обробки події
        public void GetLength(double r, ref double L)
        {
            L = (double)(2 * 3.1415 * r);
        }
        public void GetArea(double r, ref double S)
        {
            S = (double)(3.1415 * r * r);
        }
    }
}
```

2. В файлі Form1.cs оголосимо клас, що демонструє використання методів з класу CalcFigures.

```
using System;
using System.Windows.Forms;

namespace EventExample5
{
    public partial class Form1 : Form
    {
        // оголошення типу делегату
        delegate void CalculateFig(double r, ref double l);
        // оголошення події на основі типу CalcFig
        event CalculateFig eventCalcFig;

        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            // Демонстрація використання події EvCF
            CalcFigures calcFigures  = new CalcFigures(); // calcFigures - екземпляр
об'єкту класу

            double len, r;
            r = 1.0;
            len = 0;

            eventCalcFig   = calcFigures.GetLength;

            // запустити подію
            eventCalcFig(r, ref len); // len = 6.238 - довжина кола
            label1.Text = len.ToString();

            // видалити зі списку подію
            eventCalcFig -= calcFigures.GetLength;

            // додати до списку інший метод
```

```
            eventCalcFig += calcFigures.GetArea;

            // запустити подію
            eventCalcFig(r, ref len); // len = 3.1415 - площа круга
            label2.Text = len.ToString();
        }
    }
}
```

3. Результат




## 14.2. Анонімні методи та лямбда-вирази як обробники подій, що використовуються в середовищі .NET

### 14.2.1. Використання анонімного методу в якості обробника події

У прикладі продемонстровано використання анонімного методу в якості обробника події для додатку, створеного за шаблоном Windows Forms Application. В анонімному методі обчислюється площа круга на основі заданого радіусу ***r***.

Послідовність кроків наступна.

Нехай тип делегату та подія оголошені в деякому класі наступним чином:

```
// оголосити тип делегату CalcFigure
delegate double CalcFigure(double r);

 // оголосити подію
event CalcFigure eventCF
```

Тоді, програмний код, що демонструє подію, буде мати вигляд:

```
// використання анонімного методу в якості обробника події
CF += delegate(double r) {    // у методі обчислюється площа круга радіусу r
                           const double Pi = 3.1415;
                           double res;     res = Pi * r * r;
                           return res;
                         };
 // перевірка
double R = 3.0;
double Res;
Res = CF(R); // Res = 28.2735
// використання анонімного методу в якості обробника події
CF += delegate(double r) {     // у методі обчислюється площа круга радіусу r
                            const double Pi = 3.1415;
```

```
                            double res;
                            res = Pi * r * r;
                            return res;
                          };
 // перевірка
double R = 3.0;
double Res;
Res = CF(R); // Res = 28.2735
```

Якщо додаток створено за шаблоном Windows Forms Application, то усі ці дії можна виконати, наприклад, в обробнику події кліку на кнопці. У цьому випадку, програмний код модуля форми буде мати приблизно такий вигляд:

```
using System;
using System.Collections.Generic; using System.ComponentModel; using
System.Data; using System.Drawing; using System.Linq; using System.Text; using
System.Windows.Forms;
namespace TrainEvents03
{
    public partial class Form1 : Form
    {
        // оголосити тип делегату CalcFigure
       delegate double CalcFigure(double r);
        // оголосити подію CF          event CalcFigure CF;
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // використання анонімного методу в якості обробника події
            CF += delegate(double r)   {
                // у методі обчислюється площа круга радіусу r
                 const double Pi = 3.1415;
                double res;
                 res = Pi * r * r;
                 return res;
            };

            // перевірка
             double R = 3.0;
             double Res;
            Res = CF(R); // Res = 28.2735
             label1.Text = Res.ToString();
        }
    }
}
```

### 14.2.2. Використання лямбда-виразу в якості обробника події

У коді нижче демонструється використання лямбда-виразу в якості обробника події. Так само, як і в попередньому прикладі обчислюється площа круга.

Кроки демонстрації лямбда-виразу такі самі як і в попередньому прикладі.

Спочатку в класі потрібно оголосити тип делегату та подію:

```
// оголосити тип делегату CalcFigure
delegate double CalcFigure(double r);

 // оголосити подію CF
event CalcFigure CF;
```

Потім в деякому методі цього класу можна сформувати наступний код, що демонструє використання лямбда-виразу

```
// використання лямбда-виразу в якості обробника події
CF = (r) => {     // обчислюється площа круга радіусу r
                     double res;
                     res = 3.1415 * r * r;
                     return res;
                    };

 // перевірка
double RR, Res; RR = 2.0;
Res = CF(RR);
```

### 14.2.3. Обробники подій та забезпечення сумісності з середовищем .NET

Для того, щоб власні обробники подій були сумісні з середовищем .NET вони повинні мати два параметри:

перший параметр – посилання на об'єкт, що формує подію;

другий параметр – параметр типу EventArgs.

Загальна форма .NET-сумісного обробника події:

```
    void ім'я_обробника(object відправник, EventArgs e)
    {
        // ...
    }
```

де

*ім'я_обробника* – ім'я методу обробника події;

EventArgs – клас, що містить додаткову інформацію про подію.

У цьому випадку тип делегату, при оголошенні, має мати таку саму сигнатуру.

**Приклад.** Продемонструємо використання події, сумісною з середовищем .NET

Спочатку потрібно оголосити тип делегату, сигнатура якого є сумісною з .NET середовищем. Наприклад, це може виглядати так:

```
// Оголосити тип делегату
delegate void MyTypeDelegate(object sndr, EventArgs e);

// Оголосити подію
event MyTypeDelegate EMD;
```

Оголошення методів, що сумісні з подією може бути таким:

```
// Оголошення методів - обробників події
   void EventHandler1(object sndr, EventArgs e)
    {
```

```
        // команди, інструкції     // ...
    }
    void EventHandler2(object sndr, EventArgs e)
    {
        // команди, інструкції
        // ...
    }
    void EventHandler3(object sndr, EventArgs e)
    {
        // команди, інструкції     // ...
    }
```

Демонстрація використання події з деякого методу, наприклад, обробника події кліку на кнопці button1_Click():

```
        private void button1_Click(object sender, EventArgs e)
        {
            // Демонстрація роботи події, що сумісна з середовищем .NET Framework
            // Сформувати список методів, що викликаються при запуску події
            EMD = EventHandler1;
            EMD += EventHandler3;
            EMD += EventHandler2;

            // Запуск події
            EMD(this, e);
            // Інший варіант запуску для події button1_Click     EMD(sender, e);

        }
```

Увесь код модуля Form1.cs, що демонструє даний приклад має вигляд:

```
using System;
System.Windows.Forms;
 namespace TrainEvents05
{
  public partial class Form1 : Form
    {
        // Оголосити тип делегату
        delegate void MyTypeDelegate(object sndr, EventArgs e);
       // Оголосити подію         event MyTypeDelegate EMD;

        public Form1()
        {
            InitializeComponent();             label1.Text = "";
        }

        // Оголошення методів - обробників події         void EventHandler1(object sndr,
EventArgs e)
        {              label1.Text += "EventHandler1 ";
        }
         void EventHandler2(object sndr, EventArgs e)
        {              label1.Text += "EventHandler2 ";
        }

        void EventHandler3(object sndr, EventArgs e)
        {
            label1.Text += "EventHandler3 ";
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Демонстрація роботи події, що сумісна з середовищем .NET Framework
            // Сформувати список метод, що викликаються при запуску події
            EMD = EventHandler1;
```

```
            EMD += EventHandler3;
            EMD += EventHandler2;
             // Запуск події
            EMD(this, e);
             // Інший варіант запуску для події button1_Click
            EMD(sender, e);
        }
    }
}
```

### 14.2.4. Узагальнений делегат EventHandler<TEventArgs>

Узагальнений делегат EventHandler<TEventArgs> є реалізований в середовищі .NET. Цей делегат призначений для заміни власного делегату події.

Наприклад, в п.5 подія

event MyTypeDelegate EMD;

може бути замінена на такий код

event EventHandler<EventArgs> EMD;

а тип делегату оголошувати не потрібно.

**Приклад.** Використання узагальненого делегата EventHandler

У прикладі використовується код, аналогічний розглянутому в попередньому пункті. Тільки замість власного типу делегату використовується узагальнений делегат EventHandler, з допомогою якого оголошується подія.

```
using System;
using System.Windows.Forms;
namespace TrainEvents05
{
    public partial class Form1 : Form
    {
        // Оголошувати власний тип делегату не потрібно
        // Оголосити подію з допомогою узагальненого делегату EventHandler
        event EventHandler<EventArgs> EMD;
        public Form1()
        {
            InitializeComponent(); label1.Text = "";
        }

        // Оголошення методів - обробників події
        void EventHandler1(object sndr, EventArgs e)
        {
            label1.Text += "EventHandler1 ";
        }

        void EventHandler2(object sndr, EventArgs e)
        {
            label1.Text += "EventHandler2 ";
        }
        void EventHandler3(object sndr, EventArgs e)
        {
            label1.Text += "EventHandler3 ";
        }

        private void button1_Click(object sender, EventArgs e)
```

```
        {
            // Демонстрація роботи події, що сумісна з середовищем .NET Framework
            // Сформувати список методів, що викликаються при запуску події
            EMD = EventHandler1;
            EMD += EventHandler3;
            EMD += EventHandler2;

            // Запуск події
            EMD(this, e);

            // Інший варіант запуску для події button1_Click
            EMD(sender, e);
        }
    }
}
```

У коді рядок

event EventHandler<EventArgs> EMD;

можна замінити спрощеним рядком

event EventHandler EMD;

### 14.2.5. Призначення об’єкту-заповнювача Empty з класу EventArgs

В середовищі .NET, в деяких випадках роботи з подіями, параметр EventArgs є непотрібним. У цих випадках доцільним є використання об’єкту-заповнювача Empty з класу EventArgs. Це здійснюється з метою спрощення створення коду.

Так, у попередньому прикладі рядки запуску події

```
// Запуск події
EMD(this, e);
// Інший варіант запуску для події button1_Click
```

EMD(sender, e);

```
// Запуск події
EMD(this, EventArgs.Empty);
// Інший варіант запуску для події
button1_Click EMD(sender, EventArgs.Empty);
```