

## Лабораторна робота № 6

# ПОНЯТТЯ ДЕЛЕГАТИВ. СТВОРЕННЯ ТА ВИКОРИСТАННЯ ДЕЛЕГАТИВ.

## ПРИНЦИПИ РОБОТИ ПОДІЙ. ОТРИМАННЯ ПОДІЙ.

**Мета роботи** – ознайомитися з поняттям делегатів та подій в C#.

Навчитися використовувати делегати та події в програмних додатках.

### Короткі теоретичні відомості

#### *Загальні відомості про делегати*

Делегат - це тип, що являє собою посилання на методи з певним списком параметрів (сигнатура) і типом, що повертається. При створенні екземпляра (примірника) делегата цей примірник можна пов'язати з будь-яким методом з сумісною сигнатурою і типом, що повертається. В результаті метод можна викликати (активувати) за допомогою примірника делегата.

Делегати також можна використовувати для передачі методів в якості аргументів до інших методів. Всі делегати є об'єктами типу System.Delegate або System.MulticastDelegate, який є похідним від першого.

Різниця між цими класами полягає в тому, що екземпляри першого (делегати) можуть зберігати лише одне посилання на метод, а примірник другої можуть містити відразу кілька посилань на методи. Завдяки цьому, можна приєднувати до одного делегату кілька методів, кожен з яких при єдиному зверненні до делегату буде викликатися по ланцюжку. Таким чином, з програми буде видно лише один делегат, за яким ховається кілька методів.

Ця можливість дуже зручна для підтримки подій, оскільки дозволяє без використання додаткових механізмів приєднати до події декілька функцій обробників. Фактично, делегат являє собою об'єкт - чорний ящик, що приховує в своїх надрах покажчики на функції.

### *Оголошення класів – делегатів*

Не дивлячись на те, що всі делегати є нащадками класу `MulticastDelegate`, оголошення делегатів простим наслідуванням від системного класу – неприпустиме. Наступний код:

```
public class NewDelegate : MulticastDelegate {}
```

 викличе помилку при компіляції:

'NewDelegate' cannot derive from special class 'System.MulticastDelegate', що означає : 'NewDelegate' не може бути спадкоємцем спеціального класу 'System. MulticastDelegate'.

Для оголошення делегатів в C# існує спеціальна конструкція з ключовим словом `delegate`:

```
[<модифікатор рівня доступу>] delegate <тип результату> <ім'я класу>  
(<список аргументів>);
```

Приклади:

```
public delegate void NewDelegate(string s);  
public delegate int CalcDelegate(int a, int b);
```

Таким чином створюються два різних класи делегатів `NewDelegate` і `CalcDelegate`, що мають різні набори аргументів та різні типи результату.

### *Створення екземплярів (об'єктів) для делегатів*

Об'єкт делегату зазвичай створюється шляхом вказування імені методу, для якого делегат у подальшому буде слугувати оболонкою, або за допомогою анонімного методу. Після створення екземпляра делегата, виклик методу виконаний в делегаті, передається делегатом у цей метод. Параметри, що передаються делегату при виконанні, передаються в метод, а значення, що повертається (якщо воно є) повертається делегатом в об'єкт, який його викликав.

Ця процедура називається викликом делегату.

Приклад створення об'єкту делегата:

```
public delegate void NewDelegate(string s);
public delegate int CalcDelegate(int a, int b);
class Program
{
    // метод void
    WriteMessage(string str)
    {
        Console.WriteLine("Message : " + str);
    }
    static void Main(string[] args)
    {
        Program prog = new Program();
        // створення об'єкту делегата
        // з використанням конструктора
        NewDelegate del0 = new NewDelegate(prog.WriteMessage);
        // створення об'єкту делегата
        // без використання конструктора
        NewDelegate del1 = prog.WriteMessage;
        // виклик першого об'єкту делегата
        del0("OK");
        // виклик другого об'єкту делегата
        del1("ТАК");
        Console.ReadKey();
    }
}
```

Де *NewDelegate* - це клас делегат, *del0* та *del1* - об'єкти делегати, що створюються, *WriteMessage* - метод, на який посилаються обидва об'єкти. Відповідно, після створення об'єкта делегата можна звертатися до методів, на які він посилається. Виглядає це так. // виклик першого об'єкту делегата

```
del0("OK");
```

```
// виклик другого об'єкту делегата    del1("ТАК");
```

Результат роботи програми буде наступний:

Message : ОК

Message : ТАК

### ***Багатоадресні делегати***

Як вже вказувалося раніше, нащадки класу *MulticastDelegate* можуть містити відразу кілька посилань на методи. Делегати, що містять посилання на декілька методів - називаються багатоадресними.

Багатоадресні делегати містять список призначених делегатів, кожен з яких містить посилання на свій метод. При виклику багатоадресний делегат викликає по черзі всі делегати зі списку. Делегати в списку повинні бути одного й того ж самого типу (класу).

Для роботи з багатоадресними делегатами визначено операції додавання та віднімання делегатів (+, -, +=, -=).

Для наочності наведемо приклад, що демонструє роботу з операціями над делегатами.

```
using System; namespace
MyDelegate02
{ // клас делегат
    public delegate void NewDelegate(string s);
class Program
    { // метод      void
WriteMessage(string str)
    {
        Console.WriteLine("Message : " + str);
    }
    static void Main(string[] args)
    {
        Program prog = new Program();
        // створення об'єкту делегата
        // для статичного методу
```

```

    NewDelegate del0 = prog.WriteMessage;
    NewDelegate del1 = Console.WriteLine;

    // додавання делегатів
    NewDelegate del2 = del0 + del1;
    Console.WriteLine("del2 = del0 + del1");

    // виклик об'єкту делегата
    del2("Test");

    Console.WriteLine("-----");

    // додамо, ще одного делегата
    Console.WriteLine("del2 += prog.WriteMessage");
    del2 += prog.WriteMessage;      del2("Test");
    Console.WriteLine("-----");

    // віднімемо делегата
    Console.WriteLine("del2 -= Console.WriteLine");
    del2 -= Console.WriteLine; del2("Test");
    Console.ReadKey();
}
}
}

```

Після виконання на екрані отримаємо:

```

del2 = del0 + del1
Message : Test Test

```

---

```

del2 += prog.WriteMessage
Message : Test
Test
Message : Test
-----
del2 -= Console.WriteLine
Message : Test
Message : Test

```

### *Обробники подій в C #*

Події – це особливий тип багатоадресних делегатів, які можна викликати тільки з класу або структури, в якій вони оголошені (клас видавця). Якщо на

подію підписані інші класи або структури, їхні методи обробників подій будуть викликані коли клас видавця ініціює подію.

Події дозволяють класу або об'єкту повідомляти інші класи чи об'єкти про виникнення будь-яких ситуацій. Клас, що відправляє (або викликає) подію, називається видавцем, а класи, які отримують (або оброблюють) подія, називаються передплатниками.

Події оголошуються за допомогою ключового слова *event*. Синтаксис оголошення події наступний: [модифікатори] event [тип - делегат][ім'я події];

Наприклад:

```
// оголошуємо клас-делегат public
delegate void MyEventHandler();

// оголошуємо подію типу MyEventHandler public
event MyEventHandler MyEvent;
```

*Властивості подій:*

Події мають наступні властивості:

1. Диспетчер визначає момент виклику події, клієнт визначає - яка дія виконується у відповідь.
2. Подія може мати декількох клієнтів. Клієнт може оброблювати декілька подій від декількох диспетчерів.
3. Події, що не мають диспетчерів, ніколи не виникають.
4. Події зазвичай використовуються для повідомлення про дії користувачів, такі як натискання кнопок або вибір пункту меню в графічних інтерфейсах користувача.
5. Якщо у події кілька підписників(клієнтів), то при її виконанні виклик обробників подій виконується асинхронно.
6. В бібліотеці класів .NET Framework, базовим класом для всіх подій є делегат *EventHandler*, а для аргументів подій базовий клас - *EventArgs*.

## Використання подій

Розглянемо роботу з подіями на прикладі струмової відсічки. Нехай в нас є пристрій релейного захисту *Device*, який спрацьовує кожного разу як значення поточного струму *Current* досягне 10. І є вимикач, який вимикає лінію *Switch*. Відповідно *Device* – це диспетчер(ініціатор події), а *Switch* – буде клієнтом(підписником).

```
using System;

// Клас Пристрій
public class Device {

    // поле де записується поточний струм
    private double current = 0;

    // делегат    public delegate void Funct(double d);    // подія
спрацювання відсічки    public event Funct CutOff;

    // властивість, що моделює зняття показника струму
    // та спрацювання при стумі, що більше 10
    public double Current {        get { return
current; }        set {            current = value;

            if (value > 10 && CutOff != null) { CutOff(value);}

        } } //

// Клас Вимикач
//
public class Switch {

    // положення вимикача, true - включений
    public bool On = true;

    // вимикання    public void
Switch_Off(double p)
    {
        if (On) { Console.WriteLine(">>>Відключено при струмі {0:f3}<<<", p);
            On = false;
        } }
}
```

```

// Основна програма public
class Program
{
    static void
    Main()
    {
        Console.OutputEncoding = System.Text.Encoding.UTF8;
        // пристрій
        Device dev = new Device();
        // вимикач
        Switch swt = new Switch();
        // підключаємо спрацювання до події відсічка
        dev.CutOff += swt.Switch_Off; //підписка на подію
        // моделюємо зміну значення струму
        // випадковим чином
        Random rnd = new Random();
        for (int i = 0; i < 10; i++)
        {
            swt.On = true;
            // зміна струму          double current = rnd.Next(8,
            12)+ rnd.NextDouble());
            // зняття показника пристроєм
            dev.Current = current;
            Console.WriteLine("Струм = {0,7:f3} Вимикач {1}", current, swt.On ? "On" :
            "Off");
        }
        Console.ReadKey();
    }
}

```

Приклад результату роботи програми:

>>>Відключено при струмі 10,998<<<

Струм = 10,998 Вимикач Off

>>>Відключено при струмі 11,728<<<

Струм = 11,728 Вимикач Off

>>>Відключено при струмі 11,266<<<



Струм = 11,266 Вимикач Off

Струм = 9,804 Вимикач On

Струм = 9,834 Вимикач On

>>>Відключено при струмі 11,895<<<

Струм = 11,895 Вимикач Off

>>>Відключено при струмі 10,746<<<

Струм = 10,746 Вимикач Off

Струм = 9,112 Вимикач On

>>>Відключено при струмі 10,110<<<

Струм = 10,110 Вимикач Off

Струм = 9,324 Вимикач On

### Завдання до лабораторної роботи Завдання

1 Створіть набір методів для роботи з масивами.

Використовуйте механізми делегатів.

№	Завдання
1	Метод для отримання усіх парних чисел у масиві; Метод для отримання усіх чисел Фібоначчі в масиві.
2	Метод для отримання усіх непарних чисел у масиві; Метод для отримання усіх простих чисел у масиві;
3	Метод для отримання усіх чисел, кратних заданому числу у масиві; Метод для отримання усіх позитивних чисел в масиві;
4	Метод для отримання усіх чисел із заданою останньою цифрою Метод для отримання усіх чисел, що належать заданому діапазону
5	Метод для отримання усіх чисел в масиві, які є ступенями числа 2; Метод для отримання усіх чисел двозначних чисел в масиві
6	Метод для отримання усіх парних чисел у масиві; Метод для отримання усіх чисел Фібоначчі в масиві.

7	Метод для отримання усіх непарних чисел у масиві; Метод для отримання усіх простих чисел у масиві;
8	Метод для отримання усіх чисел, кратних заданому числу у масиві; Метод для отримання усіх позитивних чисел в масиві;
9	Метод для отримання усіх чисел із заданою останньою цифрою Метод для отримання усіх чисел, що належать заданому діапазону
10	Метод для отримання усіх чисел в масиві, які є ступенями числа 2; Метод для отримання усіх чисел двозначних чисел в масиві
11	Метод для отримання усіх парних чисел у масиві; Метод для отримання усіх чисел Фібоначчі в масиві.
12	Метод для отримання усіх непарних чисел у масиві; Метод для отримання усіх простих чисел у масиві;
13	Метод для отримання усіх чисел, кратних заданому числу у масиві; Метод для отримання усіх позитивних чисел в масиві;
14	Метод для отримання усіх чисел із заданою останньою цифрою Метод для отримання усіх чисел, що належать заданому діапазону
15	Метод для отримання усіх чисел в масиві, які є ступенями числа 2; Метод для отримання усіх чисел двозначних чисел в масиві

Код завдання 1(варіант 10):

```
using System;
using System.Collections.Generic;

namespace lab6_1{

    delegate bool Filter(int x);

    class NumArray {

        public static List<int> GetStepenTwo(int[] array)
        {
            return FilterNums(array, IsStepenTwo);
        }

        public static List<int> GetTwoDigit(int[] array)
```

```

    {
        return FilterNums(array, IsTwoDigit);
    }

    public static List<int> FilterNums(int[] nums, Filter f)
    {
        List<int> result = new List<int>();

        foreach (int x in nums)
        {
            if (f(x))
            {
                result.Add(x);
            }
        }
        return result;
    }

    private static bool IsStepenTwo(int x)
    {
        return (x!=0) && ((x & (x - 1)) == 0);
    }

    private static bool IsTwoDigit(int x)
    {
        return x>=10 && x<100;
    }
}

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello world!");
        int[] numbers = { 1, 2, 3, 4, 5, 10, 16, 25, 64, 99 };

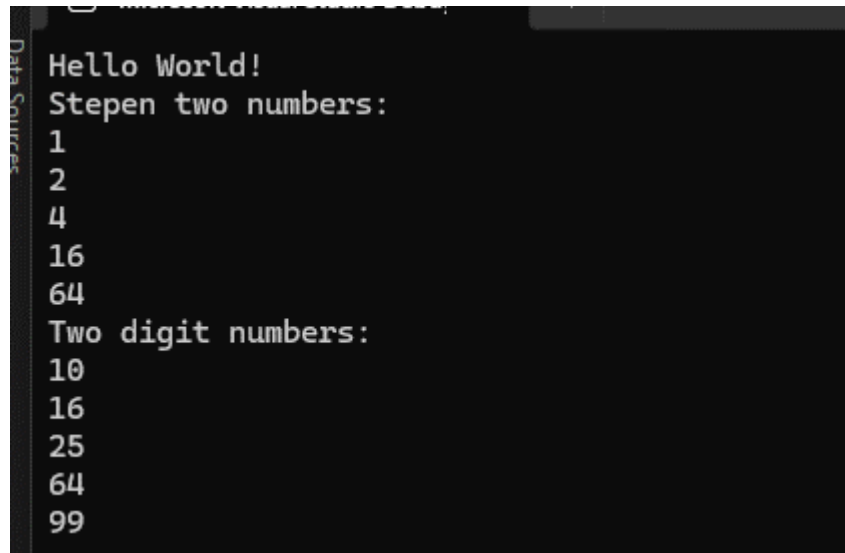
        List<int> stepenTwoNums = NumArray.GetStepenTwo(numbers);
        Console.WriteLine("Stepen two numbers: ");
        foreach (int x in stepenTwoNums)
        {
            Console.WriteLine(x);
        }

        List<int> twoDigitNums = NumArray.GetTwoDigit(numbers);
        Console.WriteLine("Two digit numbers: ");
        foreach (int x in twoDigitNums)
        {
            Console.WriteLine(x);
        }
    }
}

```

```
}  
}
```

Результат роботи завдання 1:



```
Hello World!  
Stepen two numbers:  
1  
2  
4  
16  
64  
Two digit numbers:  
10  
16  
25  
64  
99
```

Рисунок 6.1 – Результат роботи програми з використанням делегату

**Завдання 2.** Створіть анонімний делегат або лямбда-вираз для:

№	Завдання
1	перевірки числа на парність
2	підрахунку квадрата числа
3	перевірки, чи є заданий день днем програміста (256 день року)
4	пошуку максимуму в масиві
5	пошуку мінімуму в масиві
6	пошуку непарних чисел в масиві
7	перевірки, чи є поточний рік високосним

8	перевірки числа на парність
9	підрахунку квадрата числа
10	перевірки, чи є заданий день днем програміста (256 день року)
11	пошуку максимуму в масиві
12	пошуку мінімуму в масиві
13	пошуку непарних чисел в масиві
14	перевірки, чи є поточний рік високосним
15	перевірки, скільки днів у поточному місяці

Код до завдання 2(варіант 10):

```
using System;

namespace lab6_2
{
    class Program
    {
        delegate bool IsDay(DateTime date);
        static void Main(string[] args)
        {
            IsDay isProgrammerDay = delegate (DateTime date)
            {
                return date.DayOfYear == 256;
            };

            DateTime checkDate = new DateTime(2024, 9, 12);

            if (isProgrammerDay(checkDate))
            {
                Console.WriteLine("This is programmer day!");
            }
            else
            {
                Console.WriteLine("This is not programmer day!");
            }
        }
    }
}
```

Результат роботи:

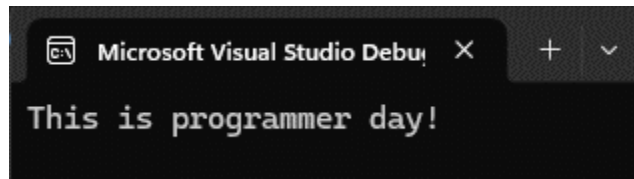


Рисунок 6.2 – Результат роботи програми з анонімним делегатом-функцією

### Завдання 3.

<b>Парні варіанти</b>	<p>Створіть клас «Кредитна картка». Клас повинен містити:</p> <ul style="list-style-type: none"><li>■ Номер картки;</li><li>■ ПІБ власника;</li><li>■ Термін дії карти;</li><li>■ PIN;</li><li>■ Кредитний ліміт; ■</li></ul> <p>Сума грошей.</p> <p>Створіть потрібний набір способів класу. Реалізуйте події для наступних ситуацій:</p> <ul style="list-style-type: none"><li>■ Поповнення рахунку;</li><li>■ Витрата коштів з рахунку;</li><li>■ Старт використання кредитних коштів;</li></ul>
	<ul style="list-style-type: none"><li>■ Досягнення ліміту заданої суми грошей;</li><li>■ Зміна PIN</li></ul> <p>При спробі перевищення ліміту використання грошей генерувати виключення</p>

<b>Непарні варіанти</b>	<p>Створити клас «Рюкзак». Характеристики рюкзака:</p> <ul style="list-style-type: none"> <li>■ Колір рюкзака;</li> <li>■ Марка і виробник;</li> <li>■ Вага рюкзака;</li> <li>■ Об'єм рюкзака;</li> <li>■ Вміст рюкзака (список об'єктів у кожного об'єкта, крім назви, потрібно враховувати зайнятий об'єм).</li> </ul> <p>Створіть методи для заповнення характеристик. Створіть події для додавання об'єкту в рюкзак, видалення об'єкту з рюкзака, перегляду вмісту рюкзака, визначення вільного об'єму рюкзака. Якщо при спробі додавання об'єкту може бути перевищено обсяг рюкзака, потрібно генерувати виняток.</p>
-----------------------------	--

### Контрольні запитання

1. Що таке делегат в C#?

Делегат в C# - це тип, що описує посилання на методи з певною сигнатурою (список параметрів та тип повернення).

2. Нащадками яких класів є делегати?

Делегати не є нащадками жодного класу в C#.

3. Синтаксис оголошення делегатів C#.

```
delegate void MathOperation(int x, int y);
```

4. Які типи методів можна присвоювати делегатам?

Делегату можна присвоїти будь-який метод, чия сигнатура сумісна з сигнатурою делегата.

5. Що означає термін "багатоадресні делегати"?

Багатоадресні делегати - це тип делегатів, що дозволяють присвоювати їм кілька методів.

6. Які операції допустимі для багатоадресних делегатів?

Призначення: Додавання та видалення методів з списку делегата.

Об'єднання: З'єднання двох багатоадресних делегатів в один.

Від'єднання: Роз'єднання двох багатоадресних делегатів.

Виклик: Запуск послідовного виконання всіх методів, прив'язаних до делегата.

7. Що таке анонімні методи?

Анонімні методи - це безіменні методи, що визначаються безпосередньо в місці їх використання.

8. Що таке події в C#?

Події в C# - це механізм, що дозволяє об'єктам повідомляти про те, що з ними відбуваються певні події (наприклад, клік кнопки, зміна значення).

10. Синтаксис оголошення подій.

```
public event EventHandler OnButtonClicked;
```

11. Які основні властивості подій?

Інкапсуляція: Внутрішнє реалізованя події приховано від зовнішнього світу, доступні лише методи підписки та відписки.

Делегування: Події реалізовані за допомогою делегатів.