4 ДЕЛЕГАТИ. ПОДІЇ. ЛЯМБДИ

4.1 Мета роботи

Навчитися створювати класи, які мають делегати, події, анонімні методи і лямбди. Реалізувати прослуховування подій і реагування на події.

4.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити делегати, події, анонімні методи і лямбди та ознайомитися з прослуховуванням та реагуванням на події.

Делегати

Крім властивостей та методів класи можуть мати делегати і події. Делегати – це такі об'єкти, які вказують на інші методи. Тобто делегати є вказівниками на методи. За допомогою делегатів ми можемо викликати певні методи у відповідь на деякі дії, що відбулися.

Методи, на які посилаються делегати, повинні мати такі ж параметри і такий же тип значення, що повертається. Створимо два делегата:

```
delegate int Operation(int x, int y);
delegate void GetMessage();
```

Для об'явлення делегата використовується ключове слово *delegate*, після якого йде тип значення, що повертається, назва і параметри. Перший делегат посилається на функцію, яка в якості параметрів приймає два значення типу *int* і повертає деяке число. Другий делегат посилається на метод без параметрів, який нічого не повертає.

Щоб скористатися делегатом, потрібно створити його об'єкт за допомогою конструктора, в який ми передаємо адрес метода, який викликається делегатом. Щоб викликати метод, на який вказує делегат, потрібно використати його метод *Invoke*. Крім того, делегати можуть виконуватися в асинхронному режимі, при цьому нам не потрібно створювати другий потік, а лише замість метода *Invoke* використати пару методів *BeginInvoke/EndInvoke*.

```
class Program
{
   delegate void GetMessage(); // 1. Об'являємо делегат
   static void Main(string[] args)
   {
      GetMessage del; // 2. Створюємо змінну делегата
      if (DateTime.Now.Hour < 12)
      {
         del = GoodMorning; // 3. Присвоюємо змінній адрес метода
      }
      else
      {
         del = GoodEvening;
      }
      del.Invoke(); // 4. Викликаємо метод
```

```
            Console.ReadLine();
        }
        private static void GoodMorning()
        {
            Console.WriteLine("Good Morning");
        }
        private static void GoodEvening()
        {
            Console.WriteLine("Good Evening");
        }
    }
```

За допомогою властивості *DateTime.Now.Hour* отримуємо поточний час. В залежності від часу в делегат передається адрес певного методу.

```
    class Program
    {
        delegate int Operation(int x, int y);
        static void Main(string[] args)
        {
            // присвоювання адреси метода через конструктор
            Operation del = new Operation(Add); // делегат вказує на метод Add
            int result = del.Invoke(4,5);
            Console.WriteLine(result);
            del = Multiply; // тепер делегат вказує на метод Multiply
            result = del.Invoke(4, 5);
            Console.WriteLine(result);
            Console.Read();
        }
        private static int Add(int x, int y)
        {
            return x+y;
        }
        private static int Multiply (int x, int y)
        {
            return x * y;
        }
    }
```

В цьому випадку делегату присвоюється адреса метода через конструктор. Оскільки метод, як і делегат, має два параметра, то при викликанні делегата в методі *Invoke* передаємо два параметра. Крім того, так як метод повертає значення *int*, то можна присвоїти результат роботи метода *Invoke* якій-небудь змінній.

Метод *Invoke*() при виклику делегата можна опустити і використати скорочену форму:

```
    del = Multiply; // тепер делегат вказує на метод Multiply
    result = del(4, 5);
```

Події

Події сигналізують систему про те, що відбулася певна дія. Події об’являються в класі за допомогою ключового слова *event*, після якого йде назва делегата.

```
    // Об’являємо делегат
    public delegate void AccountStateHandler(string message);
```

```
// Подія, яка виникла при виведенні грошей
public event AccountStateHandler Withdrowed;
```

Зв'язок з делегатом означає, що метод, який оброблює дану подію, повинен приймати такі ж параметри, що й делегат, і повертати той же тип, що і делегат.

```
class Account
{
   // Об'являємо делегат
   public delegate void AccountStateHandler(string message);
   // Подія, яка виникла при виведенні грошей
   public event AccountStateHandler Withdrowed;
   // Подія, яка виникла при додаванні на рахунок
   public event AccountStateHandler Added;
   int _sum; // Змінна для зберігання суми
   int _percentage; // Змінна для зберігання процента
   public Account(int sum, int percentage)
   {
      _sum = sum;
      _percentage = percentage;
   }
   public int CurrentSum
   {
      get { return _sum; }
   }
   public void Put(int sum)
   {
      _sum += sum;
      if (Added != null)
         Added("На рахунок поступило " + sum);
   }
   public void Withdraw(int sum)
   {
      if (sum <= _sum)
      {
         _sum -= sum;
         if (Withdrowed != null)
            Withdrowed("Сума " + sum + " знято з рахунку");
      }
      else
      {
         if (Withdrowed != null)
            Withdrowed("Недостатньо грошей на рахунку");
      }
   }
   public int Percentage
   {
      get { return _percentage; }
   }
}
```

В цьому прикладі створено дві події *Withdrowed* і *Added*. Обидві події об'явлено як екземпляри делегата *AccountStateHandler*, тому для реагування на ці

події потрібно метод, який приймає строку в якості параметра. Потім в методах *Put* і *Withdraw* викликаються ці події. Використання подій в основній програмі.

```
class Program
{
   static void Main(string[] args)
   {
      Account account = new Account(200, 6);
      // Додаємо обробник події
      account.Added += Show_Message;
      account.Withdrowed += Show_Message;
      account.Withdraw(100);
      // Видаляємо обробник події
      account.Withdrowed -= Show_Message;
      account.Withdraw(50);
      account.Put(150);
      Console.ReadLine();
   }
   private static void Show_Message(string message)
   {
      Console.WriteLine(message);
   }
}
```

Анонімні методи

Анонімні методи представляють собою скорочений запис методів. Інколи методи потрібні для однієї події і більше ніде не використовуються. Анонімні методи дозоляють вбудувати код там, де він викликається:

```
Account account = new Account(200, 6);
// Додаємо обробник події
account.Added += delegate(object sender, AccountEventArgs e)
{
   Console.WriteLine("Сумма транзакции: {0}", e.sum);
   Console.WriteLine(e.message);
};
```

Вбудовування відбувається за допомогою ключового слова *delegate*, після якого йде список параметрів і далі сам код анонімного методу. Блок анонімних методів повинен закінчуватися крапкою з комою після закритої фігурної дужки.

Лямбди

Лямбда-вирази представляють спрощений запис анонімних методів. Лямбда-вирази дозволяють створювати лаконічні методи, які можуть повертати деяке значення і які можна передати в якості параметрів в інші методи.

Лямбда-вираз має наступний синтаксис: зліва від лямбди-оператора => визначається список параметрів, а справа блок виразів.

```
class Program
{
   delegate int Square(int x); // об'являємо делегат, який приймає int і повертає int
   static void Main(string[] args)
   {
      Square squareInt = i => i * i; // об'єкту делегата присвоюється лямбда-вираз
      int z = squareInt(6); // використовуємо делегат
```

```
        Console.WriteLine(z); // виводимо число 36
        Console.Read();
    }
}
```

Вираз *i => i * i* представляє собою лямбда-вираз, де *i* – це параметр, а *i*i* – вираз. При використанні потрібно враховувати, що кожний параметр в лямбда-виразі неявно перетворюється в відповідний параметр делегата, тому типи параметрів повинні бути однаковими. Крім того, кількість параметрів повинна бути однакова. І тип значення, що повертається, повинен бути таким же, що й в делегата. Коли параметри в лямбда-виразі не потрібні, то замість них використовують пусті дужки.

Також лямбда-вираз необов'язково повинен приймати блок операторів і виразів. Також він може приймати посилання на метод:

```
class Program
{
    delegate void message(); // делегат без параметрів
    static void Main(string[] args)
    {
        message GetMessage = () => Show_Message();
        GetMessage();
    }
    private static void Show_Message()
    {
        Console.WriteLine("Hello world!");
    }
}
```

Крім того, лямбда-вираз можна передавати в якості параметрів метода.

```
class Program
{
    delegate bool IsEqual(int x);
    static void Main(string[] args)
    {
        int[] integers = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
        // знайдемо суму чисел більше 5
        int result1 = Sum(integers, x => x > 5);
        Console.WriteLine(result1); // 30
        // знайдемо суму парних чисел
        int result2 = Sum(integers, x => x % 2 == 0);
        Console.WriteLine(result2);  //20
        Console.Read();
    }
    private static int Sum (int[] numbers, IsEqual func)
    {
        int result = 0;
        foreach(int i in numbers)
        {
            if (func(i))
                result += i;
        }
        return result;
```

```
    }
}
```

4.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 4.4.
3. Оформити звіт.
4. Здати практичну частину.

4.4 Індивідуальні завдання

Завдання №1

Напишіть «пінг-понг»:

1. Створіть 2 класи *Ping* і *Pong*;

2. Один надсилає повідомлення іншому про те, що «виконано пінг», другий – про те, що «виконано понг»;

3. Одна пара об’єктів грає між собою.

Примітка: для додавання паузи між повідомленнями використовуйте наступний метод:

System.Threading.Thread.Sleep(10000); // Пауза тривалістю 10000 мілісекунд

Завдання №2

Напишіть два класи Мисливець і Заєць.

В класі Заєць є два числових поля, які відповідають поточному місцезнаходженню і є метод, який випадковим чином змінює дані значення (імітація руху зайця). Також в класі Заєць повинна бути подія, підписавшись на яку, можна кожний раз отримати нове місцезнаходження, коли Заєць рухається.

На дану подію необхідно підписати Мисливця і кожний раз, коли Заєць змінює своє місцезнаходження, Мисливець виводить відповідне повідомлення на екран.

Завдання №3

В завданні 2 додатково напишіть на подію Зайця анонімний метод і лямбда-вираз, які виводять відповідне повідомлення на екран.

4.5 Зміст звіту

Звіт має містити:

– мету роботи;

– індивідуально виконані завдання;

– код програми;

– результат виконання програми;

– висновки.

Контрольні питання та завдання

1. Що таке делегат? Який синтаксис його об’явлення?
2. Для чого використовуються делегати?
3. Що таке подія? В чому полягає різниця між подією і делегатом?
4. Для чого використовують анонімні методи?
5. Що таке лямбда-вираз?
6. Який синтаксис лямбда-виразів? Де зручно використовувати лямбда-вирази?