**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Лабораторна робота №4

з дисципліни «Об’єктно орієнтоване програмування»

«СТВОРЕННЯ ТА ВИКОРИСТАННЯ ДЕЛЕГАТІВБ ПОДІЙ ТА ЛЯМБДА-ВИРАЗІВ»

Виконав студент:

групи ПІ-223Б

Карпов І.Б.

Перевірив викладач:

Дишлевий О.П.

Київ 2022

**Мета:** дослідити механізм обробки подій у мові С#, навчитися створювати та використовувати делегати.

**Завдання**

1. Описати на мові C# лямбда-вираз чи анонімний метод згідно з варіантом та викликати його через делегат з відповідним підписом.\*
2. Створити компонент багаторазового використання (клас), що містить член-подію. Події-об’єкти відповідають деяким подіям реального об’єкта, що моделюється даним компонентом (наприклад, якщо компонент є моделлю об’єкта «автомобіль», подіями можуть бути «початок руху», «зупинка», «закінчення пального» тощо). Для події створити клас-аргумент події. Для опису події створити новий делегат або використати наявний делегат бібліотеки FLC (наприклад EventHandler).\*\*
3. Використовуючи створений у п.3 компонент, створити додаток, у якому визначити метод-обробник події для цього компонента, що реалізує реакцію додатка на подію (наприклад, повідомлення користувачеві про виникнення події). Метод-обробник події повинний отримувати інформацію про об’єкт-ініціалізатор та аргумент події.\*\*

\* Для оцінки «добре» реалізувати і анонімний метод і лямбда-вираз.

\*\* Для оцінки «відмінно» обов’язково повинні бути: 1) клас-аргумент подій; 2) клас, в якому генерується подія (знаходяться сама подія та метод, що її викликає); 3) клас (-и), в якому є метод (-и) обробники подій; 4) метод-обробник події обов’язково повинен мати сигнатуру, як у стандартних подіях .Net Framework. Допускається використовувати generic EventHandler.

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер варіанта** | **Метод для виклику через делегат** | **Клас (функціональність)** | **Подія** |
| 5 | Визначення кількості цифр у рядку | Стек:  Ініціалізація стеку, розміщення та видалення даних зі стеку | Очищення стеку |

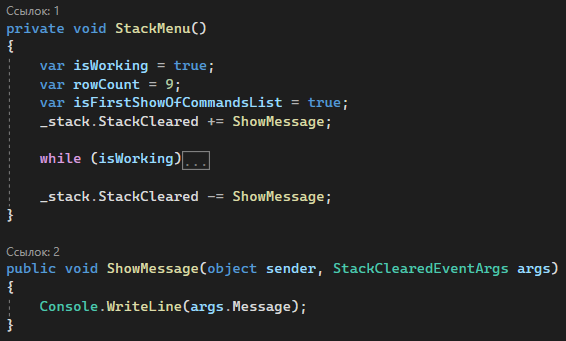
**Хід роботи**

Код програми та приклад роботи зберігається на GitHub за цим посиланням:  
[https://github.com/IvanKarpov-1/course2-term1-OOP-lab](https://github.com/IvanKarpov-1/course2-term1-OOP-lab3)4

Для підрахунку кількості цифр в рядку шляхом анонімного методу та лямбда-виразу було створено клас DigitCounter. В потрібний метод класу передається рядок та за допомогою регулярного виразу ведеться підрахунок.

Для виконання пунктів 2 та 3 було створено клас що представляє стек з усіма відповідними операціями (MyStack) та клас аргумент події (StackClearedEventArgs). В класі стеку було визначено подію StackCleared типу EventHandler<StackClearedEventArgs>. Подія викликається в методі OnStackCleared(StackClearedEventArgs args), який в свою чергу викликається коли відбувається очищення стеку, або витягування останнього елементу з нього.

Обробник події, метод ShowMessage(object sender, StackClearedEventArgs args), знаходиться в класі ConsoleMenu. Підписування на подію відбувається під час заходу в метод меню стеку, відписування – під час виходу з методу:



**Відповіді на контрольні запитання**

1. **Роз’ясніть сутність поняття делегата.**

Делегати представляють такі об'єкти, які вказують на методи. Тобто делегати - це покажчики на методи і за допомогою делегатів ми можемо викликати дані методи.

1. **Наведіть приклад опису делегата та виклику методу, використовуючи делегат.**

Для оголошення делегата використовується ключове слово delegate, після якого йде повертається тип, назва і параметри. Наприклад:

delegate void Message();

Делегат Message як повертається типу має тип void (тобто нічого не повертає) і не приймає ніяких параметрів. Це означає, що цей делегат може вказувати на будь-який метод, який не приймає ніяких параметрів і нічого не повертає.

Спочатку ми визначаємо делегат:

delegate void Message();

В даному випадку делегат визначається всередині класу, але також можна визначити делегат поза класом всередині простору імен.

Для використання делегата оголошується змінна цього делегата:

Message mes;

За допомогою властивості DateTime.Now.Hour отримуємо поточну годину. І в залежності від часу в делегат передається адреса певного методу. Зверніть увагу, що ці методи мають те ж значення що повертається і той же набір параметрів (в даному випадку відсутність параметрів), що і делегат.

mes = GoodMorning;

Потім через делегат викликаючи метод, на який посилається даний делегат:

mes();

Виклик делегата проводиться подібно викликом методу.

*Додавання методів в делегат*

У прикладах вище змінна делегата вказувала на один метод. В реальності ж делегат може вказувати на безліч методів, які мають ту ж сигнатуру і тип що повертається. Всі методи в делегата потрапляють в спеціальний список - список виклику або invocation list. І при виклику делегата всі методи з цього списку послідовно викликаються. І ми можемо додавати в цей список не оден, а кілька методів:

class Program

{

delegate void Message ();

static void Main (string [] args)

{

Message mes1 = Hello;

mes1 + = HowAreYou; // тепер mes1 вказує на два методу

mes1 (); // викликаються обидва методи - Hello і HowAreYou

Console.Read ();

}

private static void Hello ()

{

Console.WriteLine ( "Hello");

}

private static void HowAreYou ()

{

Console.WriteLine ( "How are you?");

}

}

В даному випадку в список виклику делегата mes1 додаються два методи - Hello і HowAreYou. При виклику mes1 викликаються відразу обидва цих методу.

*Виклик делегата*

У прикладах вище делегат викликався як звичайний метод. Якщо делегат брав параметри, то при її виклику для параметрів передавалися необхідні значення:

class Program

{

delegate int Operation (int x, int y);

delegate void Message ();

static void Main (string [] args)

{

Message mes = Hello;

mes ();

Operation op = Add;

op (3, 4);

Console.Read ();

}

private static void Hello () {Console.WriteLine ( "Hello"); }

private static int Add (int x, int y) {return x + y; }

}

Інший спосіб виклику делегата пердставляет метод Invoke ():

class Program

{

delegate int Operation (int x, int y);

delegate void Message ();

static void Main (string [] args)

{

Message mes = Hello;

mes.Invoke ();

Operation op = Add;

op.Invoke (3, 4);

Console.Read ();

}

private static void Hello () {Console.WriteLine ( "Hello"); }

private static int Add (int x, int y) {return x + y; }

}

Якщо делегат бере параметри, то в метод Invoke передаються значення для цих параметрів.

1. **Наведіть склад класу делегату.**

|  |  |
| --- | --- |
| **Член** | **Призначення** |
| **Method** | Це властивість повертає об'єкт System.Reflection.Method, який представляє деталі статичного методу, підтримуваного делегатом |
| **Target** | Якщо метод, який підлягає виклику, визначений на рівні об'єкта (тобто не є статичним), то Target повертає об'єкт, який представляє метод, підтримуваний делегатом. Якщо повернене Target значення дорівнює null, значить, підлягає викликом метод є статичним |
| **Combine()** | Цей статичний метод додає метод в список, який підтримується делегатом. У C # цей метод викликається за рахунок використання перевантаженої операції + = в якості скороченою нотації |
| **GetlnvokationList()** | Цей метод повертає масив типів System.Delegate, кожен з яких представляє певний метод, доступний для виклику |
| **Remove() RemoveAll()** | Ці статичні методи видаляють метод (або всі методи) зі списку дзвінків делегата. У C # метод Remove () може бути викликаний неявно, за допомогою перевантаженої операції - = |

1. **Поясніть, чим забезпечується контроль типів в делегатах.**
2. **У що розгортається делегат? (чим насправді є делегат, із яких компонентів він складається)**
3. **Поясніть сутність поняття анонімного методу.**

З делегатами тісно пов'язані Анонімні методи. Анонімні методи використовуються для створення екземплярів делегатів.

Визначення анонімних методів починається з ключового слова delegate, після якого йде в дужках список параметрів і тіло методу в фігурних дужках:

delegate (параметри)

{

// інструкції

}

наприклад:

class Program

{

delegate void MessageHandler (string message);

static void Main (string [] args)

{

MessageHandler handler = delegate (string mes)

{

Console.WriteLine (mes);

};

handler ("hello world!");

Console.Read ();

}

}

1. **Поясніть сутність лямбда-виразу, наведіть приклад лямбда-виразів.**

Лямбда-вирази подають спрощену запис анонімних методів. Лямбда-вирази дозволяють створити ємкі лаконічні методи, які можуть повертати деяке значення і які можна передати в якості параметрів в інші методи.

Ламбда-вирази мають наступний синтаксис: зліва від лямбда-оператора => визначається список параметрів, а праворуч блок виразів, що використовує ці параметри: (спісок\_параметров) => вираз. наприклад:

class Program

{

delegate int Operation (int x, int y);

static void Main (string [] args)

{

Operation operation = (x, y) => x + y;

Console.WriteLine (operation (10, 20)); // 30

Console.WriteLine (operation (40, 20)); // 60

Console.Read ();

}

}

1. **Поясніть побічні ефекти, що виникають при використанні анонімних методів та лямбда-виразів.**
2. **Наведіть приклад опису події.**

public delegate void Overflow();

public class ArifmeticFunc

{

public event Overflow EventOverflow;

public ArifmeticFunc()

{

EventOverflow += OutOfRangeHandler; // задать метод для вызова сообщения о событии

}

public void OutOfRangeHandler()

{

Console.WriteLine("Array is overflowed");

}

}

Подія – автоматичне повідомлення про виконання деякої дії.

Метод, який обробляє подію, повинен приймати ті самі параметри, як і делегат, і повертати той тип, що і делегат.

1. **Поясніть, яким чином генерується подія.**

Тип, в якому є визначені події, повинен мати генератор події. Об’єкт повинен задати метод обробки події і підписатись на дану подію, зареєструвавши цей обробник.

1. **Поясніть, яким чином виконується підписання на події та скасування підписки.**

Підпис делегата повинен відповідати сигнатурі метода-обробника події, який повинен бути зареєстрований або підписаний на дану подію.

1. **Поясніть особливості мультиадресних делегатів.**

Мультиадресні делегати наслідуються від класу MulticastDelegate простору імен System.

Для додавання делегатів застосовується операція + =. Однак варто відзначити, що в реальності станеться створення нового делегата об'єкта, який отримає методи старої копії делегата і нового методу, і новий створюваний делегат об'єкт буде присвоєно зміненій mes1.

При додаванні делегатів слід врахувати, що ми можемо додати посилання на один і той же метод декілька разів, і в списку виклику делегата тоді буде кілька посилань на один і той же метод. Відповідно при виклику делегата доданий метод буде викликаний стільки разів, скільки він був доданий.

1. **Чим відрізняється делегат та подія.**

Події :

* засновані на моделі делегата
* це повідомлення, надіслані об'єктом, щоб сигналізувати про виникнення дії
* є особливим видом мультиадресного делегата, який можна викликати лише з класу або структури, де вони оголошені
* є екземпляром делегата