МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

кафедра программного обеспечения и администрирования

информационных систем

Отчёт

по лабораторной работе №7

**«**Делегаты и события**»**

по дисциплине

**«**Объектно-ориентированные языки и системы»

|  |  |
| --- | --- |
| Выполнил: | студент группы 313  Мусонда Салиму |
| Проверил: | к.т.н., доцент  кафедры ПОиАИС  Макаров К.С. |

Курск

2020

# Цель: овладение основными приемами событийного программирования и их программной реализацией.

**Задачи**:

1. Изучить основные понятия Лекции 1.
2. Выполнить задания (см. раздел Задания) в соответствии с вариантом.

**Теоретические сведения**

***Делегат*** – это вид класса, предназначенный для хранения ссылок на методы. Делегат представляет собой тип данных. Его базовым классом является класс System.Delegate. Наследовать от делегата нельзя. Объявление делегата можно размещать непосредственно в пространстве имен или внутри класса.

Описание делегата задает сигнатуру методов, которые могут быть вызваны с его помощью:

[атрибуты] [спецификаторы] delegate <тип> <имя\_делегата> ([параметры])

*Спецификаторы* делегата имеют тот же смысл, что и для класса, причем допускаются только спецификаторы new, public, protected, internal и private.

*Тип* описывает возвращаемое значение методов, вызываемых с помощью делегата, а необязательными *параметрами* делегата являются параметры этих методов.

Делегат может хранить ссылки на несколько методов и вызывать их по очереди.

Пример описания делегата:

public delegate void D(int *i*);

Делегат может вызывать либо метод экземпляра класса, связанный с объектом, или статический метод, связанный с классом.

При реализации делегата создается ссылка типа делегата и ей присваивается ссылка на метод, который передается делегату в качестве параметра, причем используется только имя метода (параметры не указываются). Затем этот метод вызывается посредством экземпляра делегата.

Таким образом, вызов экземпляра делегата трансформируется в обращение к методу, на который он ссылается при вызове. И, следовательно, решение о вызываемом методе принимается во время выполнения программы, а не в период компиляции.

Многоадресатная передача (multicasting) – это способность создавать список вызовов (или цепочку вызовов) методов, которые должны автоматически вызываться при вызове делегата. Для этого достаточно создать экземпляр делегата, а затем для добавления методов в эту цепочку использовать оператор "+=". Для удаления метода из цепочки используется оператор " - = ".

Делегат с многоадресатной передачей имеет одно ограничение: он должен возвращать тип void.

На основе делегатов построено важное средство С#: ***событие*** (event). Синтаксис события:

[атрибуты][спецификаторы]event <тип> <имя>

Для событий применяются спецификаторы new, public, protected, internal, private, static, virtual, sealed, override, abstract и extern. Событие может быть статическим (static), тогда оно связано с классом в целом, или обычным – в этом случае оно связано с экземпляром класса.

Тип события – это тип делегата, на котором основано событие.

Пример описания делегата и соответствующего ему события:

public delegate void Del(object o); //объявление делегата

class A

{

public event Del Oops; //объявление события

}

События работают следующим образом. Объект, которому необходима информация о некотором событии, регистрирует обработчик для этого события, сигнатура которых соответствует типу делегата. Когда ожидаемое событие происходит, вызываются все зарегистрированные обработчики.

Если в качестве обработчика используется статический метод, уведомление о событии применяется к классу (и неявно ко всем объектам этого класса). Если же в качестве обработчика событий используется метод экземпляра класса, события посылаются к конкретным экземплярам этого класса.

Добавлять обработчики в список или удалять их можно с помощью операторов "+=" и " - = ".

Внутри класса, в котором описано событие, с ним можно обращаться, как с обычным полем, имеющим тип делегата: использовать операции отношения, присваивания и т. д. Значение события по умолчанию – null.

Подобно делегатам события могут предназначаться для многоадресатной передачи. В этом случае на одно уведомление о событии может отвечать несколько объектов.

В библиотеке .NET описано огромное количество стандартных делегатов, предназначенных для реализации механизма обработки событий. Большинство этих классов оформлено по одним и тем же правилам:

– имя делегата заканчивается суффиксом EventHandler;

– делегат получает два параметра: первый параметр задает источник события и имеет тип object; второй параметр задает аргументы события и имеет тип EventArgs или производный от него.

Если обработчикам события требуется специфическая информация о событии, то для этого создают класс, производный от стандартного класса EventArgs, и добавляют в него необходимую информацию. Если делегат не использует такую информацию, можно обойтись стандартным классом делегата System.EventHandler.

Имя обработчика события принято составлять из префикса On и имени события.

**Задания**

**Задание 1**

Разработать проект использование делегата для:

* вызова разных методов одним экземпляром делегата;
* многоадресной передачи.

В методах предусмотреть вывод результата.

*Варианты заданий*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№№**  **№** | **Параметры делегата** | **Методы возвращают:** | | | |
| **класс1** | | | **класс2** |
| *метод\_1 (статич.)* | *метод\_2 (экз.)* | |
| 10. | char, int | если символ является цифрой найти его числовое значение, иначе его код | | вернуть код символа | если символ является буквой ‘x’, вывести число 0 |

**Задание 2**

Создать событие на основе делегата (задание 1). В тестирующем классе организовать цепочку вызовов.

Текст программы для решения задач 1-2

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace delegate\_and\_event

{

public delegate void SimpleDe(char c);

public delegate bool SimpleDelegate(char c);

public class Klass1

{

public event SimpleDelegate MyEvent;

public void RaiseEventK1(char c)

{

MyEvent(c);

Console.WriteLine("Event Raised");

}

public bool Met1(char ch)

{

if (Char.IsDigit(ch))

{

Console.WriteLine("числовое значение {0}", ch);

}

else

{

Console.WriteLine("его код {0}", (int)ch);

}

return true;

}

public bool Met2(char ch)

{

Console.WriteLine("вернуть код символа {0}", (int)ch);

return true;

}

}

public class Klass2

{

public event SimpleDe MyEventt;

public void RaiseEventK2(char c)

{

MyEventt(c);

Console.WriteLine("2nd Event Raised");

}

public void Mett(char x)

{

char symbol = 'x';

if ((x == symbol))

{

Console.WriteLine("Символ является буквой {0}", x);

}

else

{

Console.WriteLine("Символ не является буквой x");

}

}

}

class Program

{

static void Main(string[] args)

{

Console.Write(" Введите символ или номер: ");

char cah = Char.Parse(Console.ReadLine());

Klass1 obj = new Klass1();

obj.MyEvent += new SimpleDelegate(obj.Met1);

obj.MyEvent += new SimpleDelegate(obj.Met2);

obj.RaiseEventK1(cah);

Klass2 obj2 = new Klass2();

obj2.MyEventt += new SimpleDe(obj2.Mett);

obj2.RaiseEventK2(cah);

}

}

}

Тестирование задач 1-2 представлено на рисунке 1 и 2.

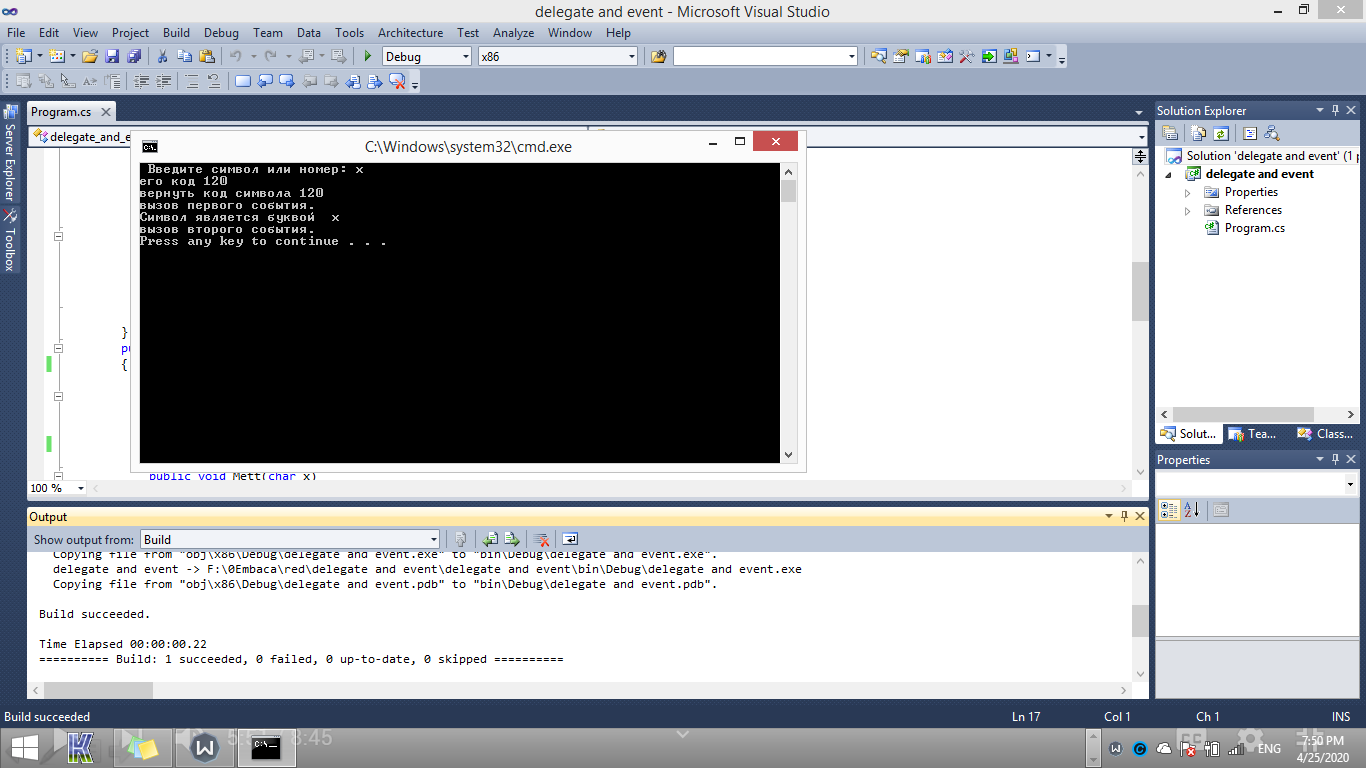


Рисунок 1 - Тестирование задач 1-2

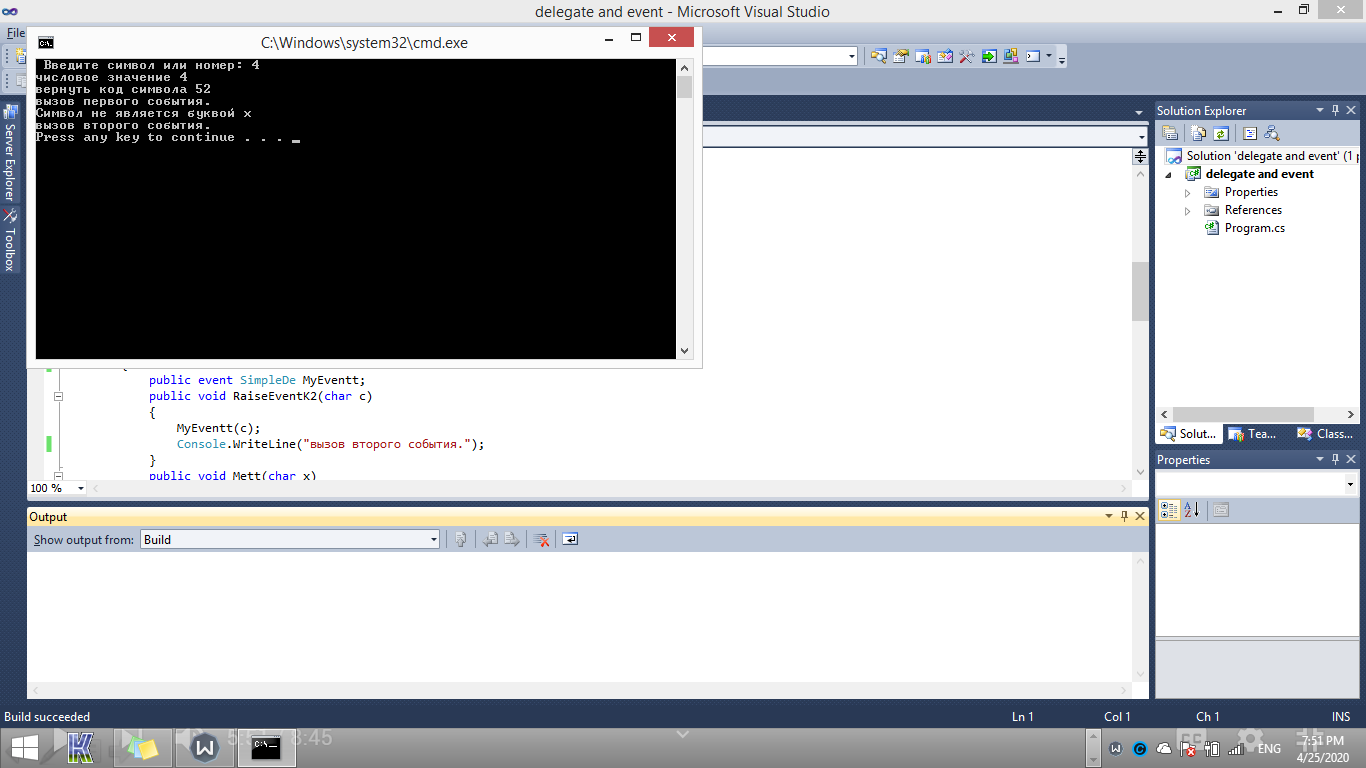


Рисунок 2 - Тестирование задач 1-2