

Московский авиационный институт
(национальный исследовательский университет)

**Институт информационных технологий и
прикладной математики**

Кафедра вычислительной математики и программирования

**Лабораторная работа №9 по курсу ООП:
Лямбда выражения. Делегаты. События.**

Работу выполнила:

М8О-209Б-19 Офицерова Т.И.

Группа

ФИО

Подпись

Вариант

Руководитель: _____/Кузнецова С.В./

Подпись:

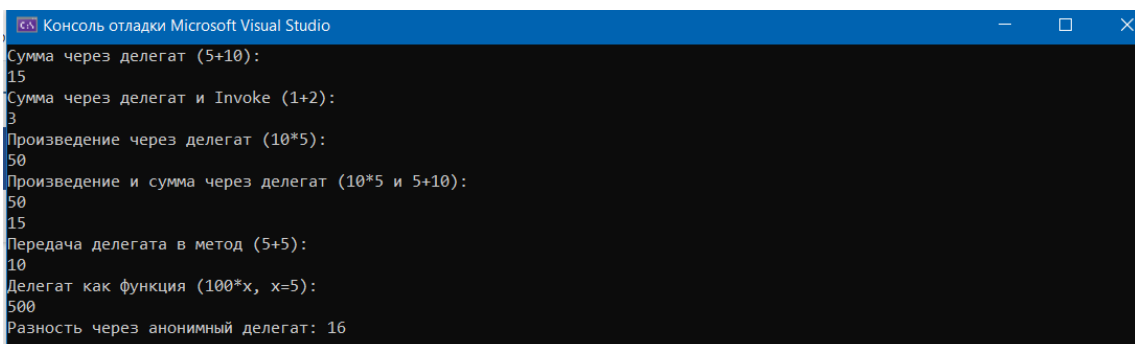
Дата: 17 октября 2020

Делегаты

Текст программы

```
using System;
namespace lab9
{
    class Program
    {
        public delegate void Operation(int x, int y);
        static void Main(string[] args)
        {
            Operation op = Sum; //ссылка на метод суммы
            Console.WriteLine("Сумма через делегат (5+10):");
            op(10, 5);
            Console.WriteLine("Сумма через делегат и Invoke (1+2):");
            op.Invoke(1, 2);
            op = Prod;
            Console.WriteLine("Произведение через делегат (10*5):");
            op(10, 5);
            op += Sum;
            Console.WriteLine("Произведение и сумма через делегат (10*5 и 5+10):");
            op(10, 5);
            op = Sum;
            Console.WriteLine("Передача делегата в метод (5+5):");
            FuncWithDelegate(op, 5, 5);
            Console.WriteLine("Делегат как функция (100*x, x=5):");
            Func<int, int> f = delegate (int x)
            {
                return 100 * x;
            };
            Console.WriteLine("{0}", f(5));
            op = delegate (int x, int y) //анонимный делегат
            {
                Console.WriteLine("Разность через анонимный делегат: {0}", x - y);
            };
            op(23, 7);
        }
        public static void Sum(int x, int y) //метод суммы
        {
            Console.WriteLine(x + y);
        }
        public static void Prod(int x, int y) //метод произведения
        {
            Console.WriteLine(x * y);
        }
        public static void FuncWithDelegate(Operation op, int x, int y)
        {
            op(x, y);
        }
    }
}
```

Результат работы



```
Консоль отладки Microsoft Visual Studio
Сумма через делегат (5+10):
15
Сумма через делегат и Invoke (1+2):
3
Произведение через делегат (10*5):
50
Произведение и сумма через делегат (10*5 и 5+10):
50
15
Передача делегата в метод (5+5):
10
Делегат как функция (100*x, x=5):
500
Разность через анонимный делегат: 16
```

Примеры

Заказ через курьерскую службу

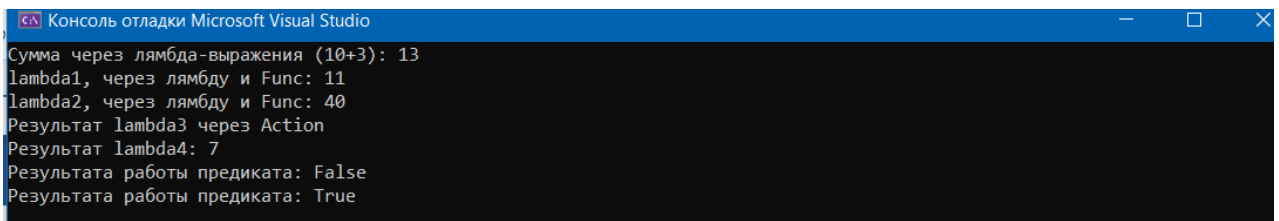
Лямбда выражения

Текст программы

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

namespace lab9
{
    class Program
    {
        delegate int Lambda(int x, int y);
        delegate void VoidLambda();
        static void Main(string[] args)
        {
            Lambda lambda = (x, y) => x + y; //лямбда
            Console.WriteLine("Сумма через лямбда-выражения (10+3): {0}", lambda(10, 3));
            Func<int, int> lambda1 = (int x) => x + 10; //с одним параметром
            Console.WriteLine("lambda1, через лямбду и Func: {0}", lambda1(1));
            Func<int, int, int> lambda2 = (x, y) => x * y; //с двумя параметрами (при
            конкретизации типа в func, его можно не указывать в параметре)
            Console.WriteLine("lambda2, через лямбду и Func: {0}", lambda2(10, 4));
            Action lambda3 = () => Console.WriteLine("Результат lambda3 через
            Action"); //Action инкапсулирует метод не имеющий входящих параметров и возвращаемого
            значения
            lambda3.Invoke(); //вызов lambda3
            Func<int, int> lambda4 = delegate (int x) { return x + 7; }; //Func может
            инкапсулировать анонимные методы
            Console.WriteLine("Результат lambda4: {0}", lambda4(0));
            Predicate<int> p = (x) => x == 7;
            Console.WriteLine("Результата работы предиката: {0}", p(1));
            Console.WriteLine("Результата работы предиката: {0}", p(7));
        }
    }
}
```

Результат работы



```
Консоль отладки Microsoft Visual Studio
Сумма через лямбда-выражения (10+3): 13
lambda1, через лямбду и Func: 11
lambda2, через лямбду и Func: 40
Результат lambda3 через Action
Результат lambda4: 7
Результата работы предиката: False
Результата работы предиката: True
```

События

Текст программы

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

namespace lab9
{
    public class EventClass
    {
        public delegate void Delegate(string mes);
        public event Delegate handler; //ссылка на обработчик событий
    }
}
```

```

    public EventClass() { }
    public void Sum(int x, int y)
    {
        handler("Сложение прошло успешно");
        Console.WriteLine("Сумма: {0}", x + y);
    }
}
class Program
{
    public static void DefaultHandler(string mes)
    {
        Console.WriteLine(mes + ", обработчиком был метод");
    }
    static void Main(string[] args)
    {
        EventClass eventClass = new EventClass();
        eventClass.handler += delegate (string mes)//обработчиком может выступать
анонимный метод
        {
            Console.WriteLine(mes + ", обработчиком был анонимный метод");
        };
        eventClass.handler += (mes) => Console.WriteLine(mes + ", обработчиком была
лямбда");//у события может быть несколько обработчиков
        eventClass.handler += DefaultHandler;
        eventClass.Sum(2, 2);
        Console.WriteLine();
        eventClass.handler -= DefaultHandler;//обработчик можно удалить
        eventClass.Sum(10, 5);
    }
}
}

```

Результат работы

```

Консоль отладки Microsoft Visual Studio
Сложение прошло успешно, обработчиком был анонимный метод
Сложение прошло успешно, обработчиком была лямбда
Сложение прошло успешно, обработчиком был метод
Сумма: 4

Сложение прошло успешно, обработчиком был анонимный метод
Сложение прошло успешно, обработчиком была лямбда
Сумма: 15

```

Примеры

Нажатие на кнопку, переход по ссылке

Контравариантность

Текст программы

```

using System;
using System.Collections.Generic;
using System.Text;

namespace lab9
{
    class Program
    {
        public interface IContravariant<in T>
        {
            public void Print();
        }
        public interface IExtContravariant<in T> : IContravariant<T> { }
        public class Printer<T> : IContravariant<T>
        {

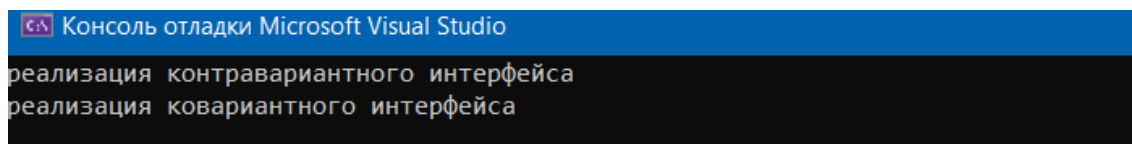
```

```

        public void Print()
        {
            Console.WriteLine("реализация контравариантного интерфейса");
        }
    }
    public interface ICovariant<out T>
    {
        public void PrintCov();
    }
    public interface ICovariantExtended<out T> : ICovariant<T> { }
    public class CovariantPrinter<T> : ICovariant<T>
    {
        public void PrintCov()
        {
            Console.WriteLine("реализация ковариантного интерфейса");
        }
    }
    static void Main(string[] args)
    {
        IContravariant<object> obj = new Printer<object>();
        IContravariant<string> str = obj; //контрвариантность, (в интерфейсе - in)
        str.Print();
        ICovariant<string> str1 = new CovariantPrinter<string>();
        ICovariant<object> obj1 = str1; //ковариантность, (в интерфейсе - out)
        obj1.PrintCov();
    }
}

```

Результат работы



Вывод

Делегат – это особый тип, хранящий ссылки на методы определенной сигнатуры, который позволяет делать программу более гибкой и универсальной. С их помощью также можно создавать и использовать анонимные методы.

Лямбда выражения – более краткая запись анонимных методов, которая позволяет улучшить читаемость кода.

Две эти конструкции могут выступать в качестве обработчиков событий. Преимущество такого подхода в том, что в классе мы вызываем обработчик, не зная его реализацию, а сам обработчик зачастую подключен извне.

Ковариантность и контравариантность связаны с возможностью использования вместо одного типа другой, расположенный выше или ниже в иерархии наследования. Контравариантность – более универсальный, ковариантность – более конкретный.