**Лаборатоная работа № 9. Анонимные методы. Делегаты.**

Реализуем маленькое общение и тесты в сигнатуре делегатов и анонимных методов.

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab9.\_1

{

class Program

{

public delegate void Str(string s);//делегаты - это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

public delegate int Sum(int x, int y);// сначала идёт ключевое слово delegate, потому тип возращаемого типа, название и параметры в скобках.

public delegate void Ob();

static void Main(string[] args)

{

Str str = null;// создаем переменную делегата

str = MessageShow;// присваиваем адрес метода

MessageShow("Иди работай с кодом! Бестолочь");// он обратится к статическому методу в этом классе и выведет строчку, т.к. мы передаем в качестве параметра строку

Sum sum = SumF;

Console.WriteLine($"SumF(20, 17) = {SumF(20, 17)}");

Console.WriteLine();

Ob ob = null;

ob += Pr;// к этой переменной делегата прибавим строчку для вывода

ob();// Привет

Str str1 = new Str(MessageShow);// ещё один способ передачи методов, создание объекта делегата с помощью конструктора в который передается нужный метод

MessageShow("Нормально блин");

Console.WriteLine();

//ob = ob + Ds; // или ob += Ds;

//ob();// Привет, Всё с тобой понятно, пока.

//Console.WriteLine();

Ob s = delegate ()// создание анонимного метода

{//создание анонимного метода начинается с ключевого слова delegate, после которого в скобках передаётся количество параметров и идёт тело метода

Console.WriteLine("Мб лучше сыграем?:D");

};

s();

Console.WriteLine();

MessaShow2("Да пошли, не ломайся :P", delegate (string tr)

{

Console.WriteLine(tr);

});

ob -= Pr;

Console.WriteLine();

ob += Ds;

ob();// Всё с тобой понятно, пока.

Console.ReadKey();

}

public static void MessaShow2(string s, Str str)

{

str(s);

}

public static void MessageShow(string s)

{

Console.WriteLine(s);

}

public static int SumF(int x,int y)

{

return x + y;

}

public static void Pr()

{

Console.WriteLine("Привет");

}

public static void Ds()

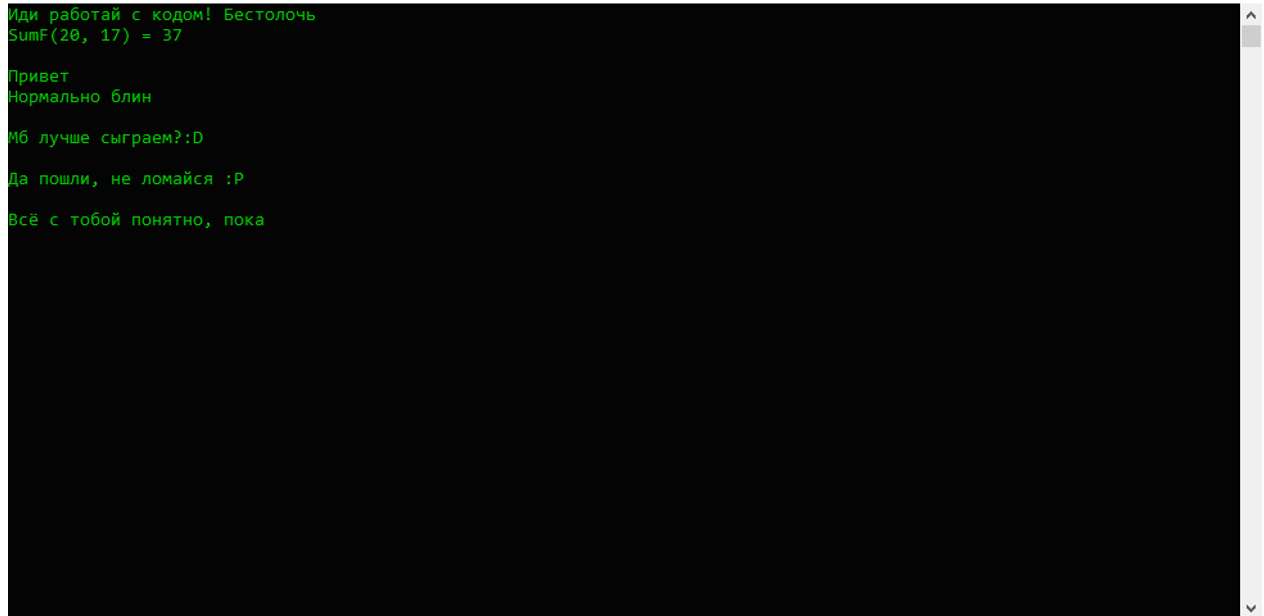
{

Console.WriteLine("Всё с тобой понятно, пока");

}

}

}



**Вывод:** Для определения делегата используется ключевое слово delegate, после которого идет сигнатура. Для использования делегата объявляется объект этого делегата. Объекту данного класса сигнатур можно присвоить метод со сходной сигнатурой. Для объектов класса сигнатур возможна операция декремента/инкремента. Также можно использовать анонимные делегаты путём объявления объекта класса сигнатур и присваивания ему анонимного метода со сходной у делегата сигнатурой.Делегаты можно передавать в качестве аргументоа методу, что довольно удобно.Использование делегатов позволяет добиться более адекватного моделирования и повторного использования кода.

**Анонимные методы. Лямбда-выражения.**

Реализуем использование лямбда-выражения с помощью делегатов с параметрами и без.

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab9.\_2

{

class Program

{

delegate void N();

delegate string Str(string s);

delegate int Sum(int x, int y);

delegate int Sum1(int x);

delegate int X(ref int x);

static void Main(string[] args)

{// лямбда выражения создаются с помощью делегатов

Sum sum = (x, y) => x + y;// самое простое лямбда выражение. В скобках передается количество параметров, после знака => выолняется блок выражений

Console.WriteLine($"sum(10,2) = {sum(10, 2)}");// параметры в скобках должны соответствовать типам и количеству параметров в делегате

Console.WriteLine();

Sum1 sum1 = x => x++;

Console.WriteLine($"sum1(1) = {sum1(1)}");

Console.WriteLine();

Str str = (s) => "Ну наконец-то";

Console.WriteLine(str);

Console.WriteLine();

N n = () => Console.WriteLine("А эти лямбда выражения прикольные");

n();

Console.WriteLine();

N n1 = () => Message();// создали объект n1 который вызывает метод Message()

n1(); // Ну привет

Console.WriteLine();

int n2 = 19;

X x1 = (ref int r) => r++; // при передаче по ссылке нужно указывать и в делегате что мы передаем по ссылке

Console.WriteLine($"x1(ref n2) = {x1(ref n2)}");

List<int> arr = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

Console.WriteLine(arr.First<int>());// вывод первого элемента массива в листе

arr.Add(15);

arr.ForEach(delegate (int i)// использование перебора элементов массива с помощью делегатов и вывод их

{

Console.Write($"{i} ");

});

arr.ForEach((i) => Console.WriteLine(i));// использование перебора элементов массива с помощью лямбда-выражений и вывод их

Console.ReadKey();

}

public static void Message()

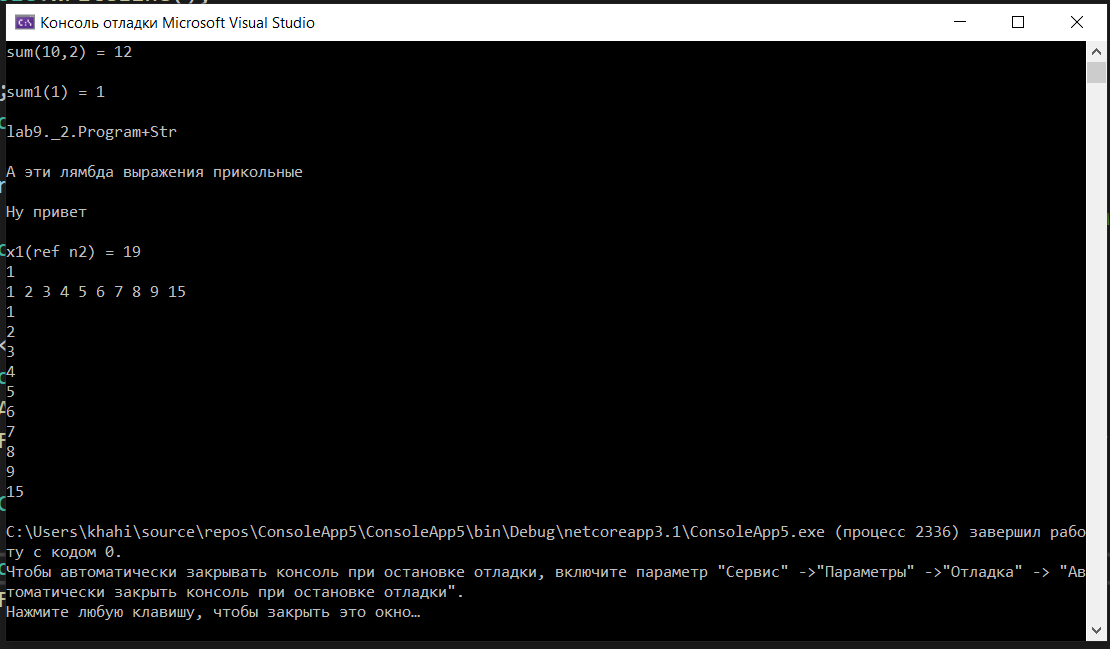
{

Console.WriteLine("Ну привет");

}

}

}



**Вывод:** Лямбда-выражения представляют упрощенную запись анонимных методов. Лямбда-выражения позволяют создать емкие лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы. Ламбда-выражения имеют следующий синтаксис: слева определяется список параметров, в середине лямбда-оператор =>, а справа блок выражений, использующий эти параметры: (список\_параметров) => выражение. Как и делегаты, лямбда-выражения можно передавать в качестве аргументов методу для тех параметров, которые представляют делегат, что довольно удобно. Как и делегаты, лямбда-выражения позволяют добиться повторного использования кода и адекватного моделирования.

**События.**

Используя события, создадим маленький тест для проверки приветствия на английском языке.

**Текст программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab4\_1

{

class Student

{

public delegate void EventStudentHandler(string s);

public event EventStudentHandler True;// события объявляются с помощью ключевого слова event, после которого следует тип делегата и потом название события

public event EventStudentHandler False;

public event EventStudentHandler Vvod;

public void vvod()

{

if (Vvod != null)

Vvod("Введите приветствие на английском языке : ");// определив событие, мы можем вызвать его как метод, используя имя события

}

public void TF(string s)

{

if (s == "hi" || s == "hello")

{

if (True != null)

{

Console.ForegroundColor = ConsoleColor.Blue;// устанавливаем цвет фона на вывод события. Это синий

True("Это правильно!");

Console.ResetColor();// сброс цвета

}

}

else

{

if (False != null)

{

Console.ForegroundColor = ConsoleColor.Red;

False("Это неправильно!");

Console.ResetColor();

}

}

}

}

class Program

{

static void Main(string[] args)

{

Student student = new Student();

student.Vvod += Show\_Message1; // добавление обработчика события. он добавляется только так : {событие} += {обработчик события}. также и удаление -=

student.True += Show\_Message;// нередко в качестве обработчиков события применяются Методы

student.False += Show\_Message;// каждый обработчик событий должен соответствовать по списку параметров и возвращаемому типу делегата

for (; ; )// здесь он для примера выводов обработчиков

{

student.vvod();

string s = Console.ReadLine();

if (s == "exit") break;

student.TF(s);

}

Console.ReadKey();

}

public static void Show\_Message(string s)

{

Console.WriteLine(s);

}

public static void Show\_Message1(string s)

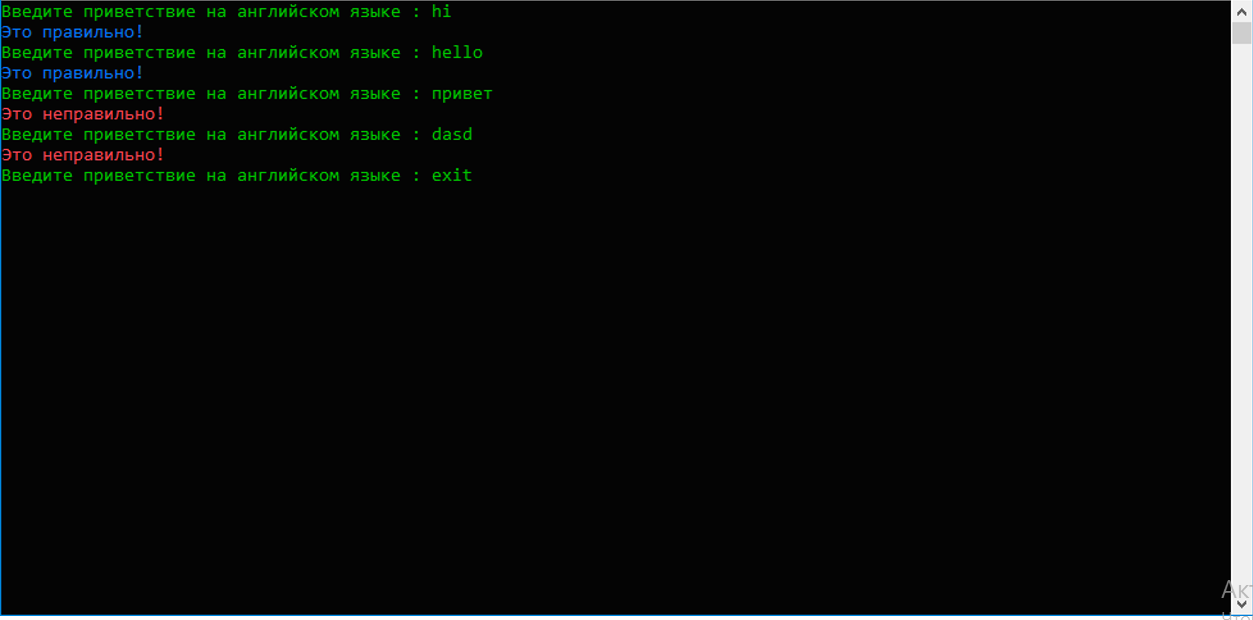
{

Console.Write(s);

}

}

}



**Вывод:** События позволяют сигнализируют системе о том, что произошло определенное действие. События объявляются в классе с помощью ключевого слова event, после которого идет название класса сигнатур, для которого объявляется это событие. Обработчик события – это метод, который выполняет некоторые действия в программе, в случае если событие произошло(сгенерировалось).