**Отчет по лабораторной работе №6 по курсу С#**

**7**

**(количество листов)**

Студентка группы ИУ5-32

Саватюгина Ю.Б.

Дата: 21.11.2017

Руководитель:

Гапанюк Ю.Е.

Подпись:

Дата:

**Задание:**

**Часть 1. Разработать программу, использующую делегаты.**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   1. метод, разработанный в пункте 3;
   2. лямбда-выражение.

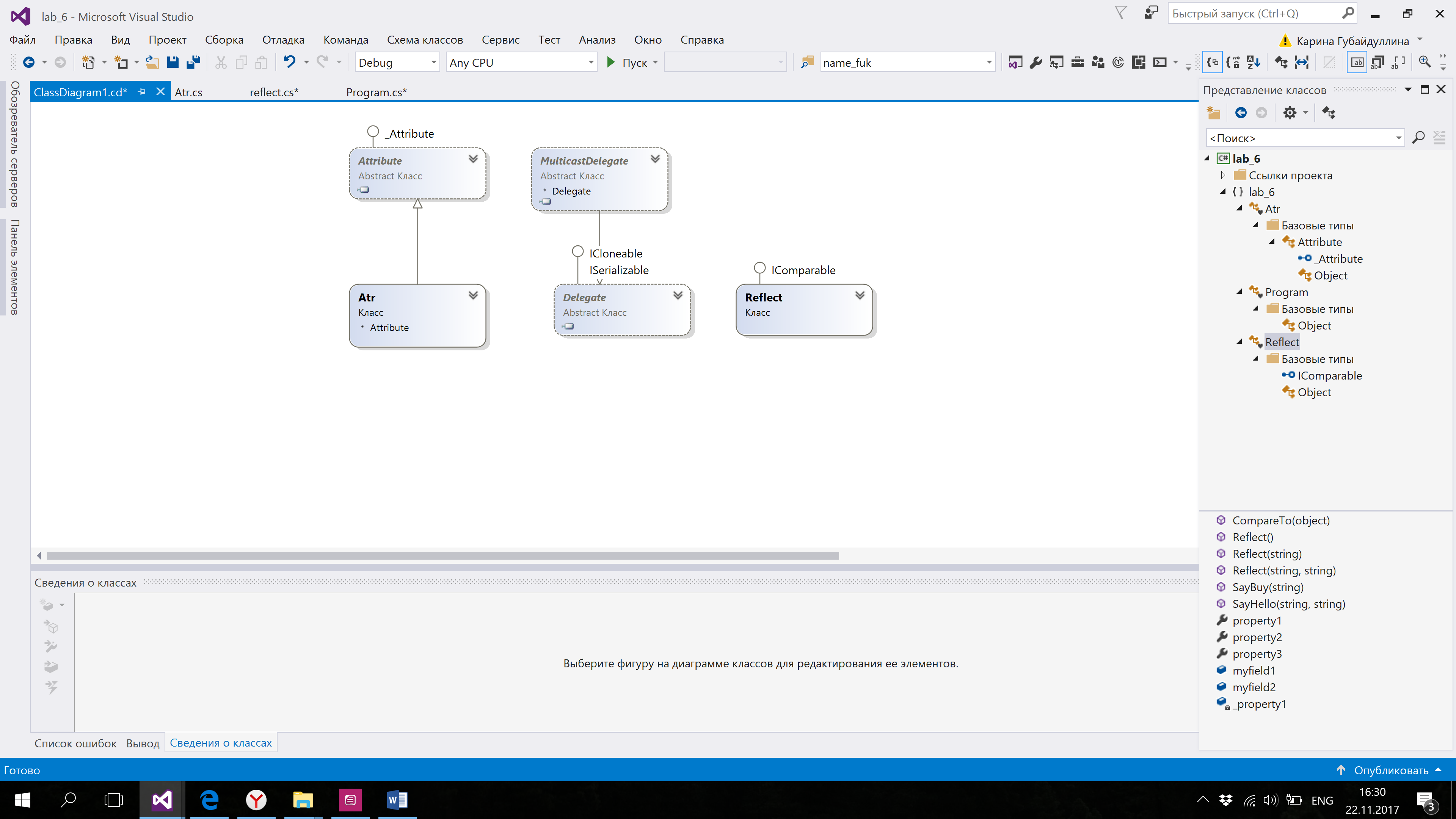
Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

**Часть 2. Разработать программу, реализующую работу с рефлексией.**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.

Вызовите один из методов класса с использованием рефлексии.

**Диаграмма классов:**



**Текст программы:**

***Program.cs***

using System;

using System.Reflection;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab\_6

{

class Program

{

static string DoPation(int p1, string p2, double p3 = 14.5)

{

string str = p2 + p3.ToString();

for (int i = 0; i < p1; i++)

{

str += '!';

}

return str;

}

static string DoSad(int p1, string p2, double p3 = 12.6)

{

string str = p2 + p3.ToString();

for (int i = 0; i < p1; i++)

{

str += '(';

}

return str;

}

static void DosomeThingMethod(string str, int i1, double i2, DosomeThing imdoing)

{

string result = imdoing(i1, str, i2);

Console.WriteLine(result);

}

static void DosomeThingMethodFunc(string str, int i1, double i2, Func<int, string, double, string> imdoing)

{

string Result = imdoing(i1, str, i2);

Console.WriteLine(Result);

}

// 22222222222222222222222222222

static void TypeInfo()

{

Reflect obj = new Reflect();

Type t = obj.GetType();

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

}

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

var isAttribute = checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void AttributeInfo()

{

Type t = typeof(Reflect);

Console.WriteLine("\nСвойства, помеченные атрибутом:");

foreach (var x in t.GetProperties())

{

object attrObj;

if (GetPropertyAttribute(x, typeof(Atr), out attrObj))

{

Atr attr = attrObj as Atr;

Console.WriteLine(x.Name + " - " + attr.Description);

}

}

}

static void InvokeMemberInfo()

{

Type t = typeof(Reflect);

Console.WriteLine("\nВызов метода:");

Reflect fi = (Reflect)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

object[] parameters = new object[] { "Dan", "Katty" };

object Result = t.InvokeMember("SayHello", BindingFlags.InvokeMethod, null, fi, parameters);

Console.WriteLine("SayHello(Dan, Katty)={0}", Result);

}

static void Main(string[] args)

{

Console.WriteLine("ПЕРВАЯ ЧАСТЬ!!!!");

string str = "sos";

int i1 = 3;

double i2 = 13.3;

DosomeThingMethod(str, i1, i2, DoPation);

DosomeThingMethodFunc(str, i1, i2, DoSad);

DosomeThing p1 = (int i11, string str1, double i22) =>

{

string result = str1 + i22.ToString();

for (int i = 0; i < i11; i++)

{

result += "metod";

}

return result;

};

DosomeThingMethod("лямбда-выражения1", i1, i2, p1);

DosomeThingMethodFunc("лямбда-выражения2", i1, i2,

(int i11, string str1, double i22) =>

{

string result = str1 + i22.ToString();

for (int i = 0; i < i11; i++)

{

result += "Func";

}

return result;

}

);

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("ВТОРАЯ ЧАСТЬ!!!!");

TypeInfo();

InvokeMemberInfo();

AttributeInfo();

Console.ReadLine();

}

}

}

***reflect.cs***

using System;

namespace lab\_6

{

class Reflect : IComparable

{

public Reflect() { }

public Reflect(string i) { }

public Reflect(string i, string j) { }

public string SayHello(string str, string str1) { return "Hi " + str + " and " + str1; }

public string SayBuy(string str) { return "Buy" + str; }

public string property1

{

get { return \_property1; }

set { \_property1 = value; }

}

private string \_property1;

public int property2 { get; set; }

[Atr(Description = "Описание для property3")]

public double property3 { get; private set; }

public int myfield1;

public float myfield2;

public int CompareTo(object obj)

{

return 0;

}

}

}

***Atr.cs***

using System;

namespace lab\_6

{

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

class Atr : Attribute

{

public Atr() { }

public Atr(string Desc)

{

Description = Desc;

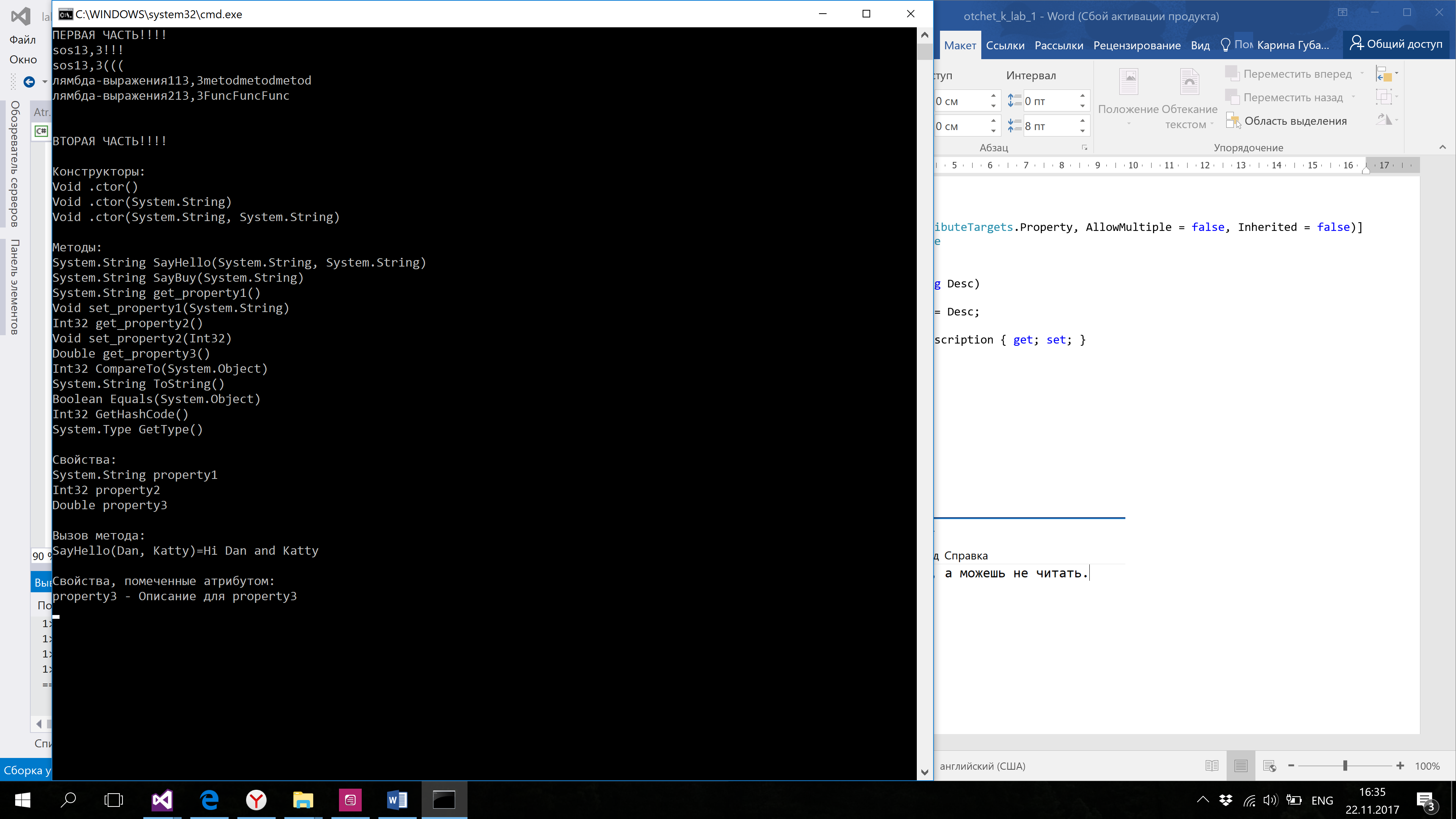
}

public string Description { get; set; }

}

}

Результаты:



|  |
| --- |
|  |