**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Дисциплина «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы  ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Тахтамышева Анна |  | Гапанюк Ю.Е. |

**Часть 1**

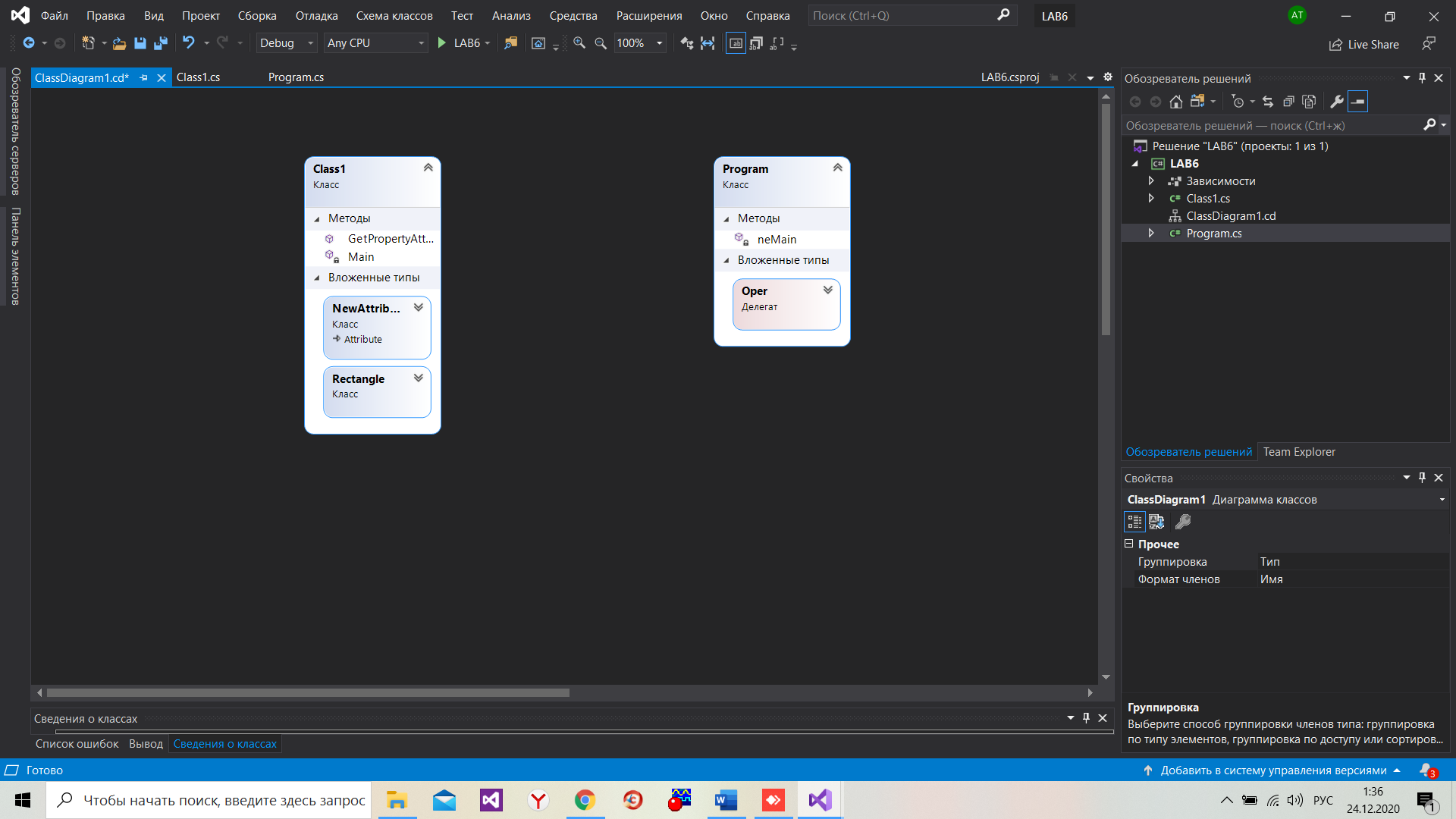
**Описание условия:**

**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:
   * метод, разработанный в пункте 3;
   * лямбда-выражение.

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

**Диаграмма классов**



**Текст программы**

using System;

namespace LAB6

{

class Program

{

delegate double Oper(double a, int b);

static void neMain(string[] args)

{

//Console.WriteLine("Hello World!");

double Plus(double a, int b)

{

return a + b;

}

void PlusMinus(

string str, double a, int b,

Oper param)

{

double Result = param(a, b);

Console.WriteLine(str + ' ' + '=' + ' ' + Result.ToString());

}

Console.WriteLine("Вызов через метод");

//Вызов через метод

PlusMinus("Плюс", 3.6, 3, Plus);

Console.WriteLine("Вызов через лямбда");

// вызов через лямбда

PlusMinus("Плюс", 3.6, 3, (a, b) => a + b);

//Func

Console.WriteLine("Вызов через Func");

void PlusMinus1(

string str, double a, int b,

Func<double, int, double> param)

{

double Result = param(a, b);

Console.WriteLine(str + ' ' + '=' + ' ' + Result.ToString());

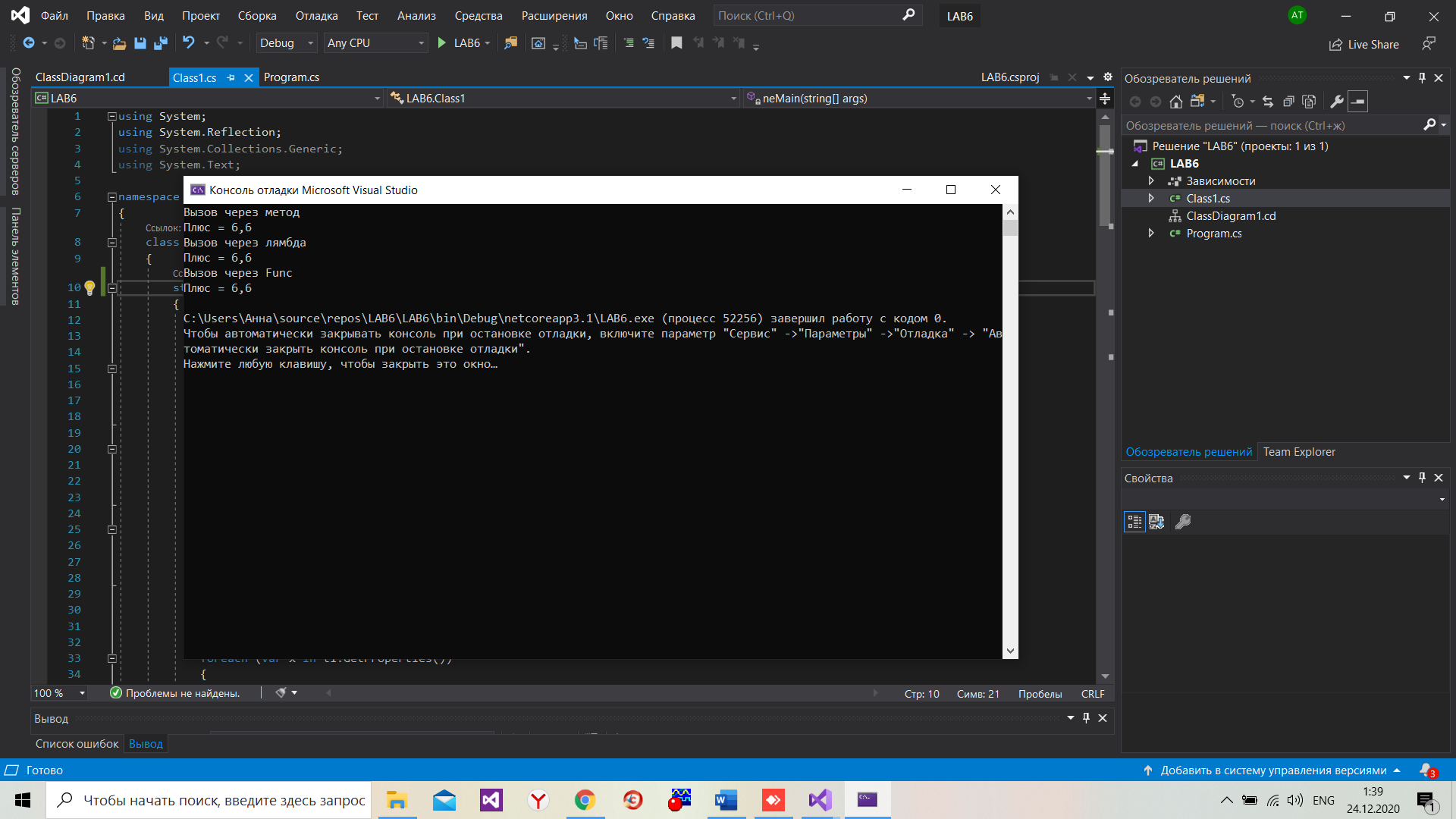
}

PlusMinus1("Плюс", 3.6, 3, Plus);

}

}

}

**Пример экранных форм**

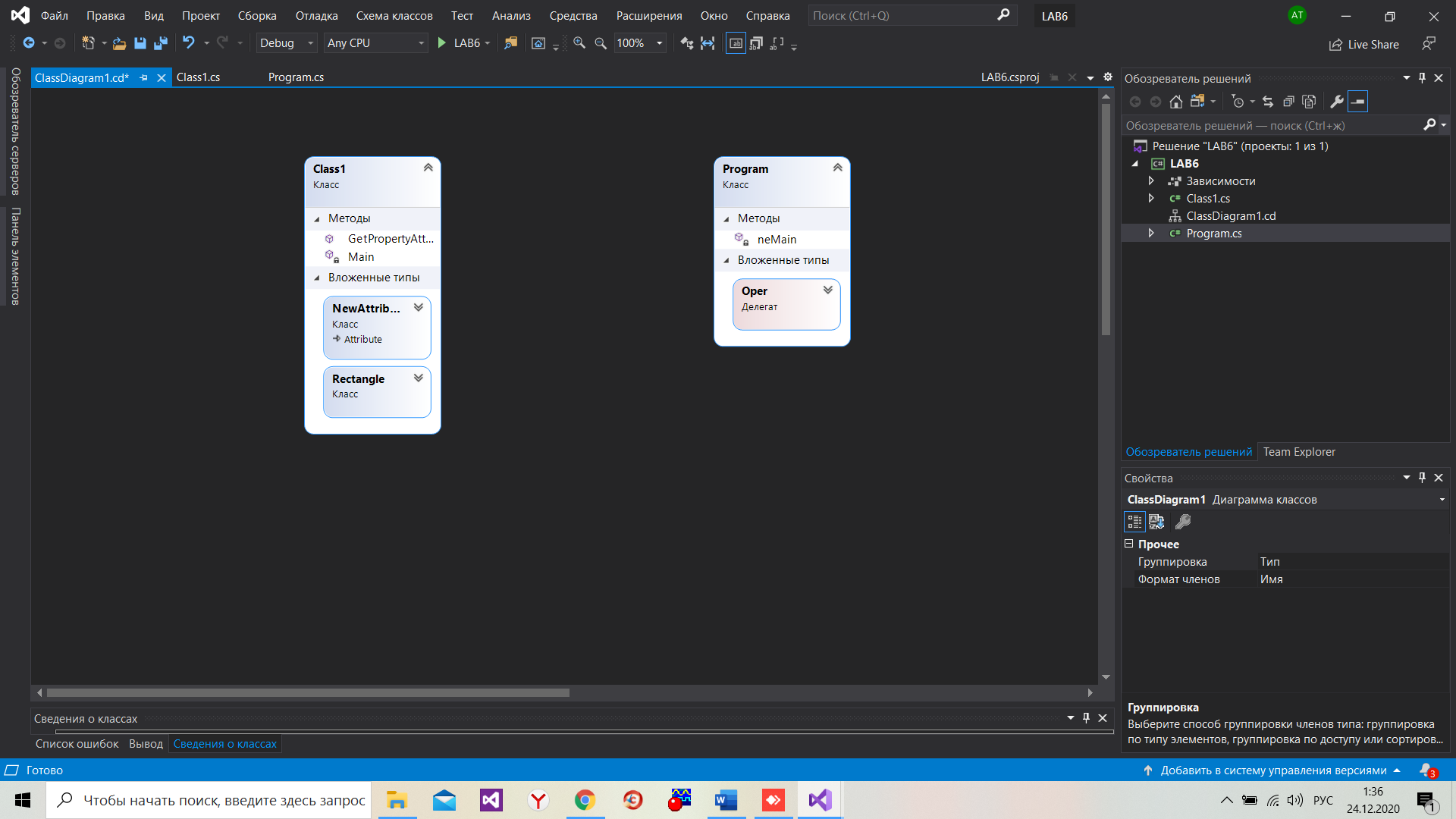
**Часть 2**

**Описание условия:**

**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.
6. Вызовите один из методов класса с использованием рефлексии.

**Диаграмма классов**



**Текст программы**

using System;

using System.Reflection;

using System.Collections.Generic;

using System.Text;

namespace LAB6

{

class Class1

{

static void Main(string[] args)

{

Rectangle obj = new Rectangle(3.1, 3.2);

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);

}

//содержат ли свойства атрибут NewAttribute

Type t1 = typeof(Rectangle);

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

foreach (var x in t1.GetProperties())

{

object attrObj;

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

{

NewAttribute attr = attrObj as NewAttribute;// Для приведения полученного значения типа object к требуемому типу NewAttribute

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

}

}

//вызов метода с использованием рефлексии

Type t2 = typeof(Rectangle);

Console.WriteLine("\nВызов метода:");

//Создание объекта

//ForInspection fi = new ForInspection();

//Можно создать объект через рефлексию

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

//Параметры вызова метода

object[] parameters = new object[] { 3.1, 3 };

//Вызов метода

object Result = t2.InvokeMember("Area1", BindingFlags.InvokeMethod, null, fi, parameters);

Console.WriteLine("Area1 (3.1 ,3)={0}", Result);

// Метод InvokeMember принимает различные

}

public class Rectangle

{

[NewAttribute("Описание для heigth")]

public double heigth

{

set

{

if (value < 0)

Console.WriteLine("Введено некоректное значение высоты");

else \_heigth = value;

}

get { return \_heigth; }

}

private double \_heigth;

[NewAttribute(Description = "Описание для width")]

public double width

{

set

{

if (value < 0)

Console.WriteLine("Введено некоректное значение ширины");

else

\_width = value;

}

get { return \_width; }

}

private double \_width;

public Rectangle() { }

public Rectangle(double width, double heigth)

{

this.width = width;

this.heigth = heigth;

}

public double Area()

{

return this.heigth \* this.width;

}

public double Area1(double heigth, double width)

{

return heigth \* width;

}

}

/// Класс атрибута

// /// </summary>

[AttributeUsage(AttributeTargets.Property, AllowMultiple =

false, Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

public static bool GetPropertyAttribute(PropertyInfo checkType, Type// информация о проверяемом свойстве,

attributeType, out object attribute)//тип проверяемого атрибута

{

bool Result = false;

attribute = null;

//Поиск атрибутов с заданным типом

var isAttribute =

checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

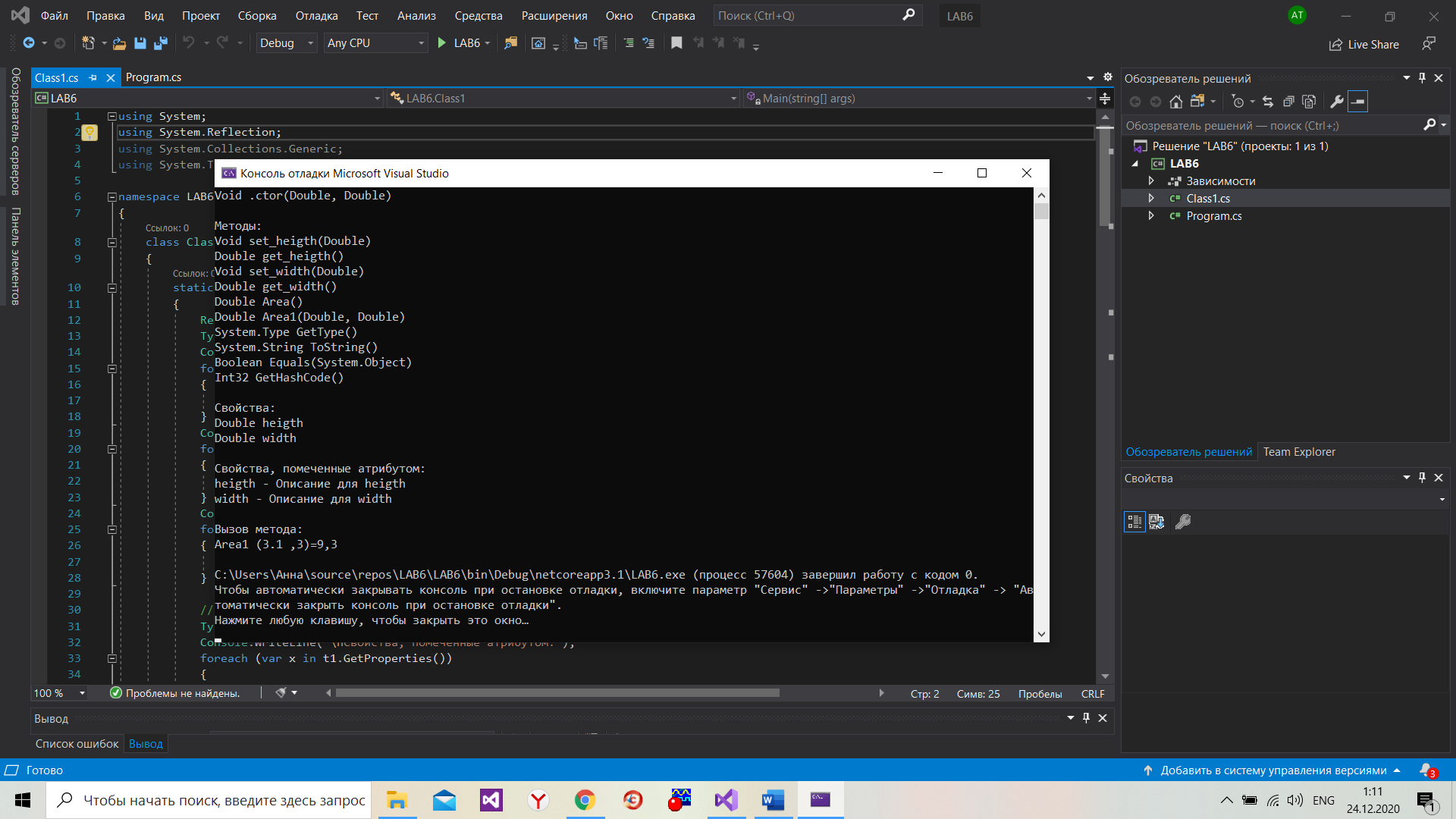
}

return Result;

}

}

}

**Пример экранных форм**