Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5.

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

Выполнил:

студент группы ИУ5-31Б

Федоров Иван

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Юрий Евгеньевич

Подпись и дата:

г. Москва, 2020 г.

Постановка задачи

Часть 1. Разработать программу, использующую делегаты.

(В качестве примера можно использовать проект «Delegates»).

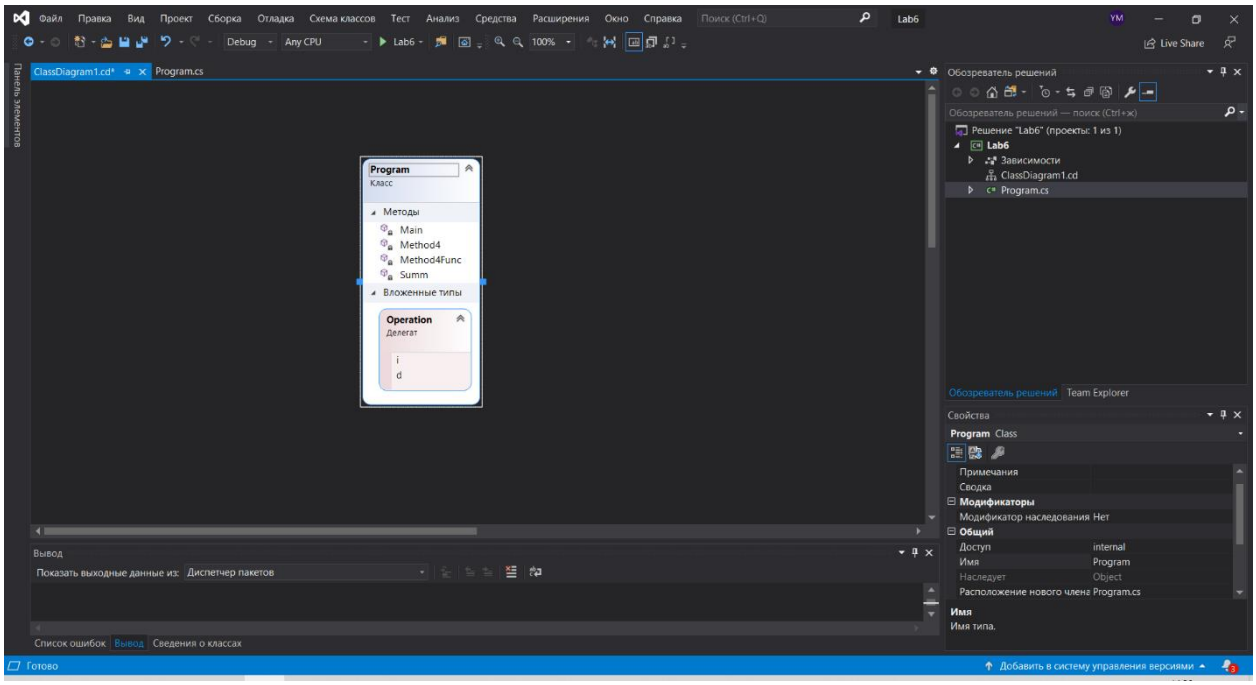
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

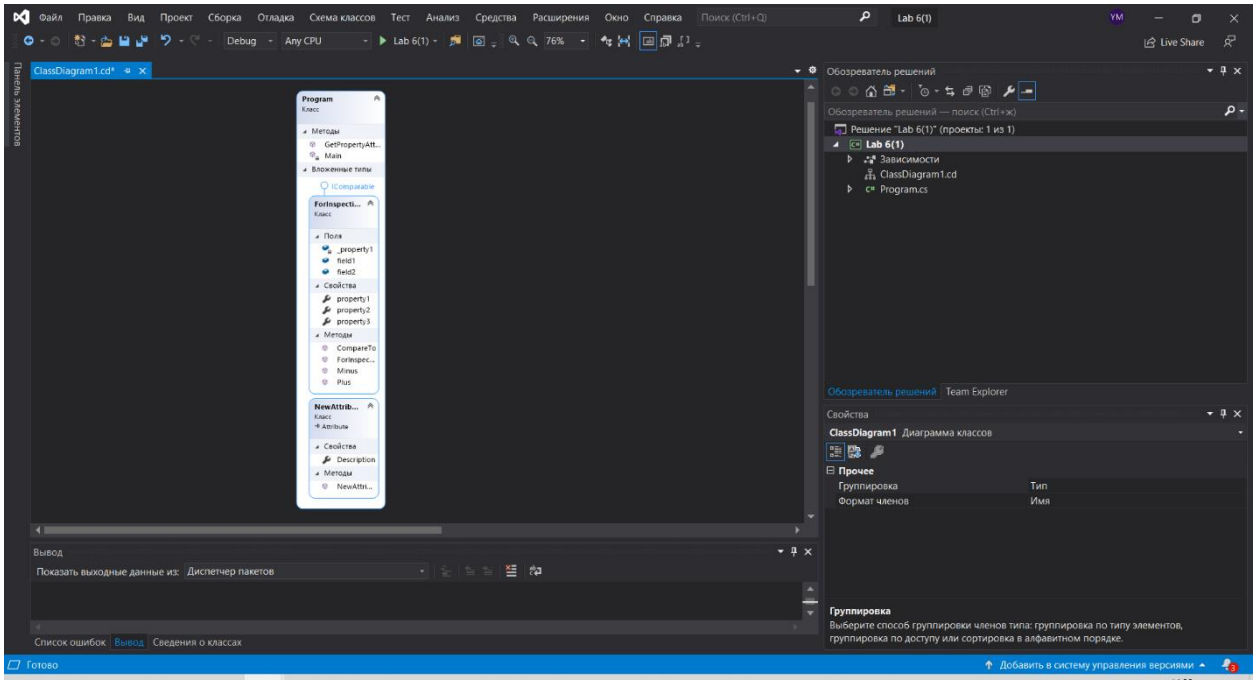
Часть 2. Разработать программу, реализующую работу с рефлексией.

(В качестве примера можно использовать проект «Reflection»).

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.

Разработка интерфейса класса





Листинг программы

```
using System;

namespace Lab6
{
    class Program
    {

        delegate object Operation(int i, double d);
```

```
        static object Summ(int i, double d)
        {
            object result = i + (int)d;
            return result;
        }

        static object Method4(int i, double d, Operation op)
        {
            object result = op(i, d);
            return result;
        }

        static object Method4Func(int i, double d, Func<int, double, object> func)
        {
            object result = func(i, d);
            return result;
        }

        static void Main(string[] args)
        {

            Console.WriteLine("Работа с делегатами");

            Console.WriteLine("Обычный делегат");

            object Result = Method4(2, 3.4, Summ);
            Console.WriteLine(Result.ToString());

            Console.WriteLine("Использование лямбда-функции");

            Result = Method4
                (
                5,
                6.78,
                (int i, double d) =>
                {
                    object result = i + (int)d;
                    return result;
                }
                );
            Console.WriteLine(Result.ToString());

            Console.WriteLine("Использование обобщённого делегата Func");

            Result = Method4Func(3, 4.56, (x, y) => x + (int)y);
            Console.WriteLine(Result.ToString());

            Console.WriteLine();
        }
    }
}
```

```
using System;
using System.Reflection;

namespace Lab_6_1_
```

```
{
    class Program
    {

        [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 class ForInspection : IComparable
        {
            public ForInspection() { }
            public ForInspection(int i) { }
            public ForInspection(string str) { }
            public int Plus(int x, int y) { return x + y; }


        public int Minus(int x, int y) { return x - y; }
        [NewAttribute("Описание для property1")]
        public string property1
        {
            get { return _property1; }
            set { _property1 = value; }
        }
        private string _property1;
        public int property2 { get; set; }
        [NewAttribute(Description = "Описание для property3")]
        public double property3 { get; private set; }
        public int field1 = 0;
        public float field2 = 0;

        public int CompareTo(object obj)
        {
            return 0;
        }

        }

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

            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;
        }

    static void Main(string[] args)
        {
```

```
            Type t = typeof(ForInspection);

            Console.WriteLine("Конструкторы:");

            foreach (var i in t.GetConstructors())
            {
                Console.WriteLine(i);
            }

            Console.WriteLine("\nСвойства");

            foreach (var i in t.GetProperties())
            {
                Console.WriteLine(i);
            }

            Console.WriteLine("\nМетоды");

            foreach (var i in t.GetMethods())
            {
                Console.WriteLine(i);
            }

            Type t1 = typeof(ForInspection);

            Console.WriteLine("\nСвойства, помеченные атрибутом:");
            foreach (var x in t1.GetProperties())
            {
                var AtandProp = x.GetCustomAttributes(typeof(NewAttribute), false);
                if ( AtandProp.Length > 0 )
                    Console.WriteLine(x);

                //object attrObj;
                //if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))
                //{
                //    NewAttribute attr = attrObj as NewAttribute;
                //    Console.WriteLine(x.Name + " - " + attr.Description);
                //}
            }


            Console.WriteLine("\nВызов метода:");

            ForInspection fi = (ForInspection)t.InvokeMember(
             null, BindingFlags.CreateInstance,
             null, null, new object[] { });
            //Параметры вызова метода
            object[] parameters = new object[] { 3, 2 };
            //Вызов метода
            object Result =
             t.InvokeMember("Plus", BindingFlags.InvokeMethod,

             null, fi, parameters);
            Console.WriteLine("Plus(3,2)={0}", Result);
        }
    }
}
```

Анализ результатов