Московский государственный технический университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

Выполнил:
студент группы РТ5-31Б
Борисочкин М.И.
Подпись и дата:

Проверил:
к.т.н., доцент
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2020 г.

Описание задания

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.

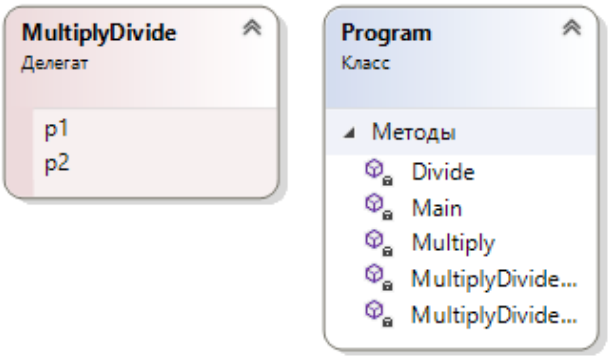
Часть 2. Разработать программу, реализующую работу с рефлексией.

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

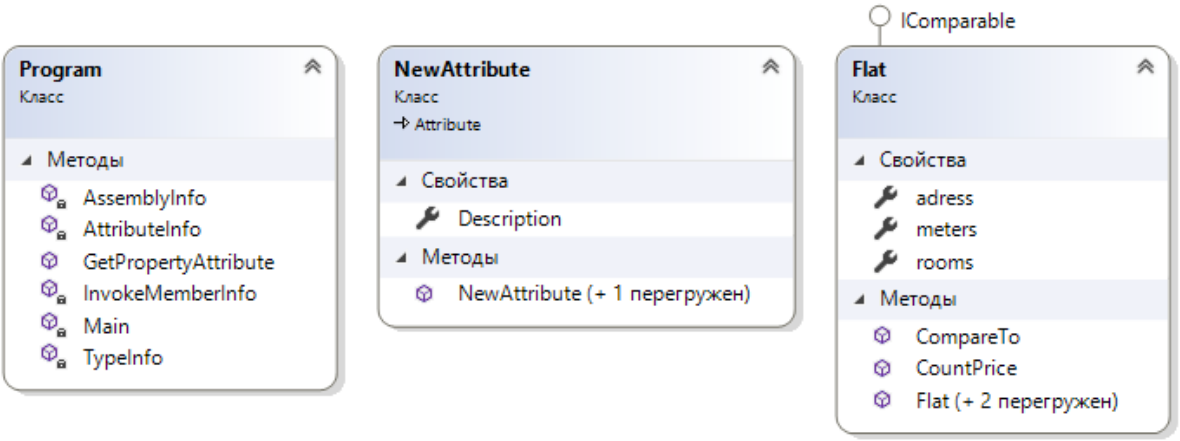
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5.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

Диаграмма классов

Часть 1. Делегаты



Часть 2. Рефлексия



Текст программы

Часть 1. Делегаты

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab6Delegates
{
    /// <summary>
    /// Делегат умножить/разделить целое на целое
    /// </summary>
    delegate double MultiplyDivide(int p1, int p2);

    class Program
    {
        /// <summary>
        /// Метод умножить, соответствующий данному делегату
        /// </summary>
        static double Multiply(int x, int y)
        {
            double res = x * y;
            return res;
        }

        /// <summary>
        /// Метод разделить, соответствующий данному делегату
        /// </summary>
        static double Divide(int x, int y)
        {
            double res = (double)x / (double)y;
            return res;
        }

        /// <summary>
        /// Метод с параметром-делегатом
        /// </summary>
        static void MultiplyDivideMethod(int x, int y, MultiplyDivide mdparam)
        {
            double res = mdparam(x, y);
            Console.WriteLine($"{res.ToString()}");
        }

        /// <summary>
        /// Метод с обощённым делегатом
        /// </summary>
        static void MultiplyDivideMethodFunc(int x, int y, Func<int,int, double> fparam)
        {
            double res = fparam(x, y);
            Console.WriteLine($"{res.ToString()}");
        }

        static void Main(string[] args)
        {
            int a, b;
```

```
            // Ввод данных
            Console.WriteLine("Введите целое a: ");
            a = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine("Введите целое b: ");
            b = Convert.ToInt32(Console.ReadLine());

            Console.WriteLine();

            // Вызов метода с параметром-делегатом
            Console.WriteLine("Вызов метода, примающий метод, с делегатом в параметре:");
            MultiplyDivideMethod(a, b, Multiply);
            MultiplyDivideMethod(a, b, Divide);

            // Вызов метода с параметром-делегатом с помощью лямбда-выражения
            Console.WriteLine("Вызов метода, примающий лямбда-выражение:");
            MultiplyDivideMethod(a, b, (x, y) => x * y);
            MultiplyDivideMethod(a, b, (x,y) => x/y);

            // Вызов метода с обощённым делегатом
            Console.WriteLine("Вызов метода, примающий обощёный делегат:");
            MultiplyDivideMethodFunc(a, b, Multiply);
            MultiplyDivideMethodFunc(a, b, Divide);

            Console.ReadLine();
        }
    }
}
```

Часть 2. Рефлексия

Flat.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab6Reflection
{
    class Flat : IComparable
    {
        // Конструкторы
        public Flat() { }
        public Flat(string adress) { this.adress = adress; }
        public Flat(string adress, double meters, int rooms)
        {
            this.adress = adress;
            this.meters = meters;
            this.rooms = rooms;
        }

        // Свойства
        [NewAttribute("Адрес")]
        public string adress { get; set; }

        public double meters { get; set; }

        [NewAttribute(Description = "Количество комнат")]
        public int rooms { get; set; }
```

```
        // Метод
        public double CountPrice(int r,double m)
        {
            double totalprice = (double)r * 1000000 + m * 100000;

            return totalprice;
        }

        /// <summary>
        /// Реализация интерфейса IComparable
        /// </summary>
        public int CompareTo(object obj)
        {
            return 0;
        }
    }
}
```

NewAttribute.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab6Reflection
{
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
    class NewAttribute : Attribute
    {
        public NewAttribute() { }
        public NewAttribute(string DescriptionParam)
        {
            Description = DescriptionParam;
        }

        public string Description { get; set; }
    }
}
```

Program.cs

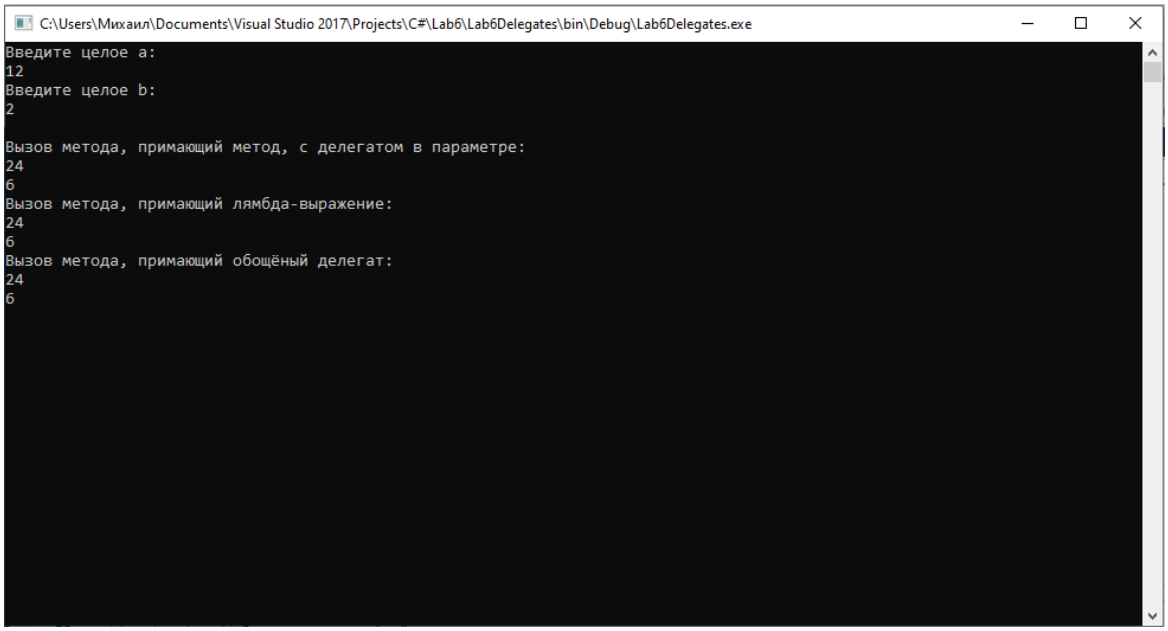
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace Lab6Reflection
{
    class Program
    {
        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;
        }

        /// <summary>
        /// Получение информации о текущей сборке
        /// </summary>
        static void AssemblyInfo()
        {
            Console.WriteLine("Вывод информации о сборке:");
            Assembly i = Assembly.GetExecutingAssembly();
            Console.WriteLine("Полное имя:" + i.FullName);
            Console.WriteLine("Исполняемый файл:" + i.Location);
        }

        /// <summary>
        /// Получение информации о типе
        /// </summary>
        static void TypeInfo()
        {
            Flat obj = new Flat();
            Type t = obj.GetType();

            //другой способ
            //Type t = typeof(ForInspection);

            Console.WriteLine("\nИнформация о типе:");
            Console.WriteLine("Тип " + t.FullName + " унаследован от " +
t.BaseType.FullName);
            Console.WriteLine("Пространство имен " + t.Namespace);
            Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

            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);
            }

            Console.WriteLine("\nFlat реализует IComparable -> " +
            t.GetInterfaces().Contains(typeof(IComparable))
            );
        }

        /// <summary>
        /// Работа с атрибутами
        /// </summary>
        static void AttributeInfo()
        {
            Type t = typeof(Flat);
            Console.WriteLine("\nСвойства, помеченные атрибутом:");
            foreach (var x in t.GetProperties())
            {
                object attrObj;
                if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))
                {
                    NewAttribute attr = attrObj as NewAttribute;
                    Console.WriteLine(x.Name + " - " + attr.Description);
                }
            }
        }

        /// <summary>
        /// Вызов метода класса с помощью рефлексии
        /// </summary>
        static void InvokeMemberInfo()
        {
            Type t = typeof(Flat);
            Console.WriteLine("\nВызов метода:");

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

            //Параметры вызова метода
            object[] parameters = new object[] { 3, 72.5 };
            //Вызов метода
            object Result = t.InvokeMember("CountPrice", BindingFlags.InvokeMethod, null,
fi, parameters);
            Console.WriteLine("CountPrice(3, 72.5)={0}", Result);
        }
```

```
static void Main(string[] args)
        {
            AssemblyInfo();
            TypeInfo();
            AttributeInfo();
            InvokeMemberInfo();

            Console.ReadLine();
        }
    }
}
```

Примеры выполнения программы

Часть 1. Делегаты



Часть 2. Рефлексия