

Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»

Кафедра ИУ5. Группа 31.

Отчет по лабораторной работе 6

Выполнила:
студентка группы ИУ5-31
Качанюк Татьяна
Подпись и дата:

Проверил:
Гапанюк Ю.Е.
Подпись и дата:

г. Москва, 2017 г.

1. 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: 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.

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.

2. Текст программы

Program.cs

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

```
namespace ConsoleApp1  
{  
    class Program  
    {
```

```

delegate int PlusOrMinus(int a, int b);

// Методы, реализующие делегат (методы "типа" делегата)
static int Plus(int a, int b) { return a + b; }
static int Minus(int a, int b) { return a - b; }

// Использование обобщенного делегата Func<>
static void PlusOrMinusMethodFunc(string str, int i1, int i2,
Func<int, int, int> PlusOrMinusParam)
{
    int Result = PlusOrMinusParam(i1, i2);
    Console.WriteLine(str + Result.ToString());
}

// Использование делегата
static void PlusOrMinusMethod(string str, int i1, int i2, PlusOrMinus
PlusOrMinusParam)
{
    int Result = PlusOrMinusParam(i1, i2);
    Console.WriteLine(str + Result.ToString());
}

// Проверка, что у свойства есть атрибут заданного типа
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)
{
    // ДЕЛЕГАТЫ
    int i1 = 1;
    int i2 = 2;

    PlusOrMinusMethod("Плюс: ", i1, i2, Plus);
    PlusOrMinusMethod("Минус: ", i1, i2, Minus);

    //Создание экземпляра делегата на основе метода
    PlusOrMinus pm1 = new PlusOrMinus(Plus);
    PlusOrMinusMethod("Создание экземпляра делегата на основе
метода: ", i1, i2, pm1);

    //Создание анонимного метода
    PlusOrMinus pm2 = delegate(int param1, int param2){ return
param1 + param2; };

    PlusOrMinusMethod("Создание экземпляра делегата на основе
анонимного метода: ", i1, i2, pm2);
    PlusOrMinusMethod("Создание экземпляра делегата на основе
лямбда-выражения: ", i1, i2, (x, y) => x - y);

    //Func<>
    Console.WriteLine("\nИспользование обобщенного делегата
Func<>");
    PlusOrMinusMethodFunc("Создание экземпляра делегата на
основе метода: ", i1, i2, Plus);
    PlusOrMinusMethodFunc("Создание экземпляра делегата на
основе лямбдавыражения 3: ", i1, i2, (x, y) => x + y);

    // РЕФЛЕКСИЯ

    Type t = typeof(Class1);
```

```

        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("\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);
          }
        }
    }

    //Создание объекта через рефлексия
    Console.WriteLine("\nСвойства, помеченные атрибутом:");
    Class1 fi = (Class1)t.InvokeMember(null,
BindingFlags.CreateInstance, null, null, new object[] { });
    //Параметры вызова метода
    object[] parameters = new object[] { 1, 2 };
    //Вызов метода
    object Result = t.InvokeMember("Plus",
BindingFlags.InvokeMethod, null, fi, parameters);
    Console.WriteLine("Plus(1,2)={0}", Result);

    Console.ReadLine();
}
}
}

```

Class1.cs

```

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

```

```
namespace ConsoleApp1
{
    class Class1
    {
        int var1;
        int var2;

        // атрибут
        [NewAttribute("Описание для var1")]
        public int VAR1
        {
            get{ return this.var1; }

            set{ this.var1 = value; }
        }

        public int VAR2
        {
            get{ return this.var2; }

            set{ this.var2 = value; }
        }

        public Class1 ()
        {
            var1 = 0;
            var2 = 0;
        }

        public Class1(int a, int b)
        {
            var1 = a;
            var2 = b;
        }

        static public int Plus(int a, int b){ return a + b; }

        static public int Minus(int a, int b){ return a - b; }
```

```

        public void Print(){ Console.WriteLine(this.var1.ToString() + " и " +
this.var2.ToString()); }

```

```

    }
}

```

NewAttribute.cs

```

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

```

```

namespace ConsoleApp1
{

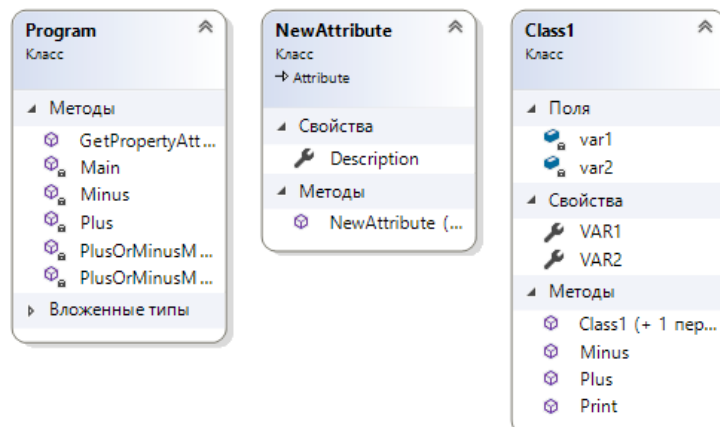
```

```

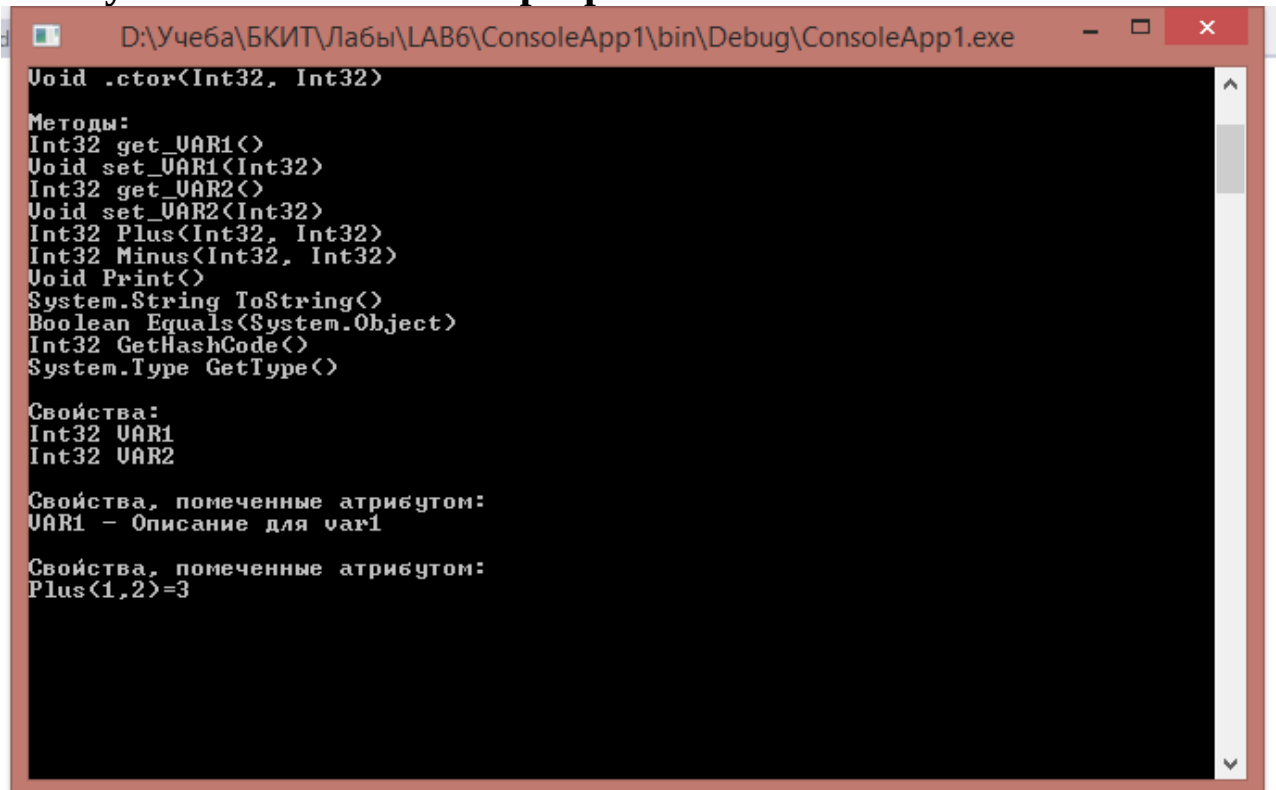
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false,
Inherited = false)]
    public class NewAttribute : Attribute
    {
        public NewAttribute() { }
        public NewAttribute(string DescriptionParam) { Description =
DescriptionParam; }
        public string Description { get; set;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

```

3. Диаграмма классов



4. Результат выполнения программы



```
Void .ctor(Int32, Int32)

Методы:
Int32 get_VAR1()
Void set_VAR1(Int32)
Int32 get_VAR2()
Void set_VAR2(Int32)
Int32 Plus(Int32, Int32)
Int32 Minus(Int32, Int32)
Void Print()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()

Свойства:
Int32 VAR1
Int32 VAR2

Свойства, помеченные атрибутом:
VAR1 - Описание для var1

Свойства, помеченные атрибутом:
Plus(1,2)=3
```