Отчет по лабораторной работе № 6 по курсу “Базовые компоненты интернет-технологий”

ИСПОЛНИТЕЛЬ:

студент группы ИУ5-33 _____

(подпись)

Желанкина А.С.

"__"_____2017 г.

Москва, МГТУ - 2017

Описание задания

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

Текст программы

Program.cs

```
using System;
using System.Reflection;

namespace Lab6_2_
{
    delegate int Multiply_Plus(int p1, int p2);
    class Program
    {
        static int Multiply(int p1, int p2) { return p1 * p2; }
        static int Plus(int p1, int p2) { return p1 + p2; }

        /// <summary>
        /// Использование обощенного делегата Func<>
        /// </summary>
        static void Multiply_Plus_MethodFunc(string str, int p1, int p2, Func<int, int,
int> Multiply_Plus_Param)
        {
            int Result = Multiply_Plus_Param(p1, p2);
            Console.WriteLine(str + Result.ToString());
        }

        /// <summary>
        /// Использование делегата
        /// </summary>
        static void Multiply_Plus_Method(string str, int p1, int p2, Multiply_Plus
Multiply_Plus_Param)
        {
            int Result = Multiply_Plus_Param(p1, p2);
            Console.WriteLine(str + Result.ToString());
        }

        /// <summary>
        /// Проверка, что у свойства есть атрибут заданного типа
        /// </summary>
        /// <returns>Значение атрибута</returns>
        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)
        {
            Console.WriteLine("Part 1. Delegates.");

            int p1 = 3;
            int p2 = 2;
```

```
            Multiply_Plus_Method("Multiplication: ", p1, p2, Multiply);
            Multiply_Plus_Method("Addition: ", p1, p2, Plus);

            //Создание экземпляра делегата на основе метода
            Multiply_Plus mp1 = new Multiply_Plus(Plus);
            Multiply_Plus_Method("Create a delegate exemplar based on the method: ", p1,
p2, mp1);
            //Создание экземпляра делегата на основе 'предположения' делегата
            //Компилятор 'предполагает' что метод Plus типа делегата
            Multiply_Plus mp2 = Plus;
            Multiply_Plus_Method("Create a delegate exemplar based on the delegate's
'assumption': ",
                p1, p2, mp2);
            //Создание анонимного метода
            Multiply_Plus mp3 = delegate (int param1, int param2)
            {
                return param1 + param2;
            };
            Multiply_Plus_Method("Create a delegate exemplar based on the anonymous
method: ", p1, p2, mp2);
            Multiply_Plus_Method("Create a delegate exemplar based on the lambda
expressions: ", p1, p2,
                (x, y) => x + y);

            Console.WriteLine("Using a Generic Delegate Func<>");
            Multiply_Plus_MethodFunc("Create a delegate exemplar based on the method: ",
p1, p2, Plus);

            string OuterString = "external variable";
            Multiply_Plus_MethodFunc("1) Create a delegate exemplar based on the lambda
expressions: ", p1, p2,
                (int x, int y) =>
                {
                    Console.WriteLine("This variable is declared outside the lambda
expression: " + OuterString);
                    int z = x + y;
                    return z;
                });

            Multiply_Plus_MethodFunc("2) Create a delegate exemplar based on the lambda
expressions: ",
                p1, p2, (x, y) => x + y);

            //Групповой делегат всегда возвращает значение типа void
            Console.WriteLine("Example of a group delegate");
            Action<int, int> a1 = (x, y) =>
            {
                Console.WriteLine("{0} * {1} = {2}", x, y, x * y);
            };
            Action<int, int> a2 = (x, y) =>
            {
                Console.WriteLine("{0} + {1} = {2}", x, y, x + y);
            };
            Action<int, int> group = a1 + a2;
            group(11, 7);
            Action<int, int> group2 = a1;
            Console.WriteLine("Adding a method call to a group delegate");
            group2 += a2;
            group2(17, 5);
            Console.WriteLine("Removing a method call from a group delegate");
            group2 -= a1;
            group2(13, 3);

            Console.WriteLine("Part 2. Reflection.");
```

```
            Type t = typeof(ForInspection);

            Console.WriteLine("A type " + t.FullName + " is inherited from " +
t.BaseType.FullName);
            Console.WriteLine("Namespace " + t.Namespace);
            Console.WriteLine("Is in the assembly " + t.AssemblyQualifiedName);
            Console.WriteLine("Constructors:");

            foreach (var x in t.GetConstructors())
            {
                Console.WriteLine(x);
            }
            Console.WriteLine("Methods:");
            foreach (var x in t.GetMethods())
            {
                Console.WriteLine(x);
            }
            Console.WriteLine("Properties:");
            foreach (var x in t.GetProperties())
            {
                Console.WriteLine(x);
            }
            Console.WriteLine("Data fields (public):");
            foreach (var x in t.GetFields())
            {
                Console.WriteLine(x);
            }
            Console.WriteLine("Properties marked with an attribute:");
            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);
                }
            }
            Console.WriteLine("Calling the method:");

            //Создание объекта через рефлексию
            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);

            Console.ReadKey();
        }
    }
}
```

NewAttribute.cs

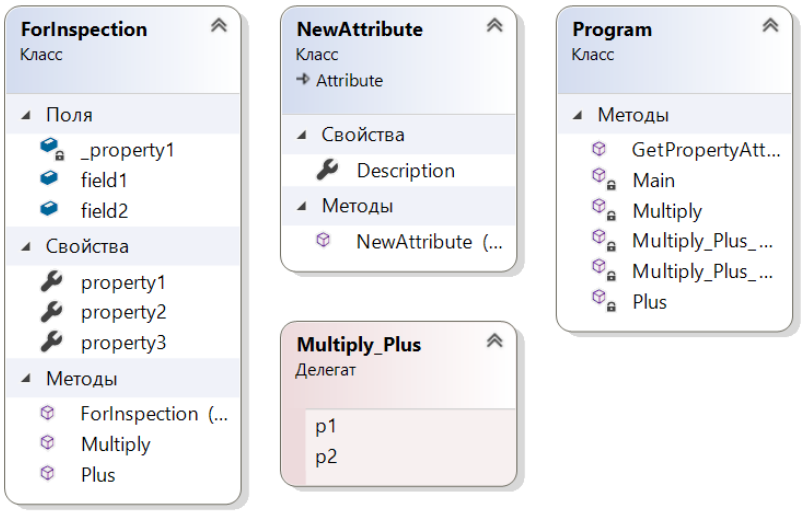
```
using System;

namespace Lab6_2_
{
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
    public class NewAttribute : Attribute
    {
```

```
        public NewAttribute() { }
        public NewAttribute(string DescriptionParam)
        {
            Description = DescriptionParam;
        }
        public string Description { get; set; }
    }
}
```

ForInspection.cs

```
namespace Lab6_2_
{
    class ForInspection
    {
        public ForInspection() { }
        public ForInspection(int i) { }
        public ForInspection(string str) { }

        public int Plus(int x, int y) { return x + y; }
        public int Multiply(int x, int y) { return x * y; }

        [New("Description for property1")]
        public string property1
        {
            get { return _property1; }
            set { _property1 = value; }
        }
        private string _property1;

        public int property2 { get; set; }

        [New(Description = "Description for property3")]
        public double property3 { get; private set; }
        public int field1;
        public float field2;
    }
}
```

Диаграмма классов



Результаты выполнения