Отчет по лабораторной работе №6 по курсу C#

7

(количество листов)

Студент группы ИУ5-32

Яценко Антон

Дата: 06.12.2017

Руководитель:

Гапанюк Ю.Е.

Подпись:

Дата:

Лабораторная работа 6

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.

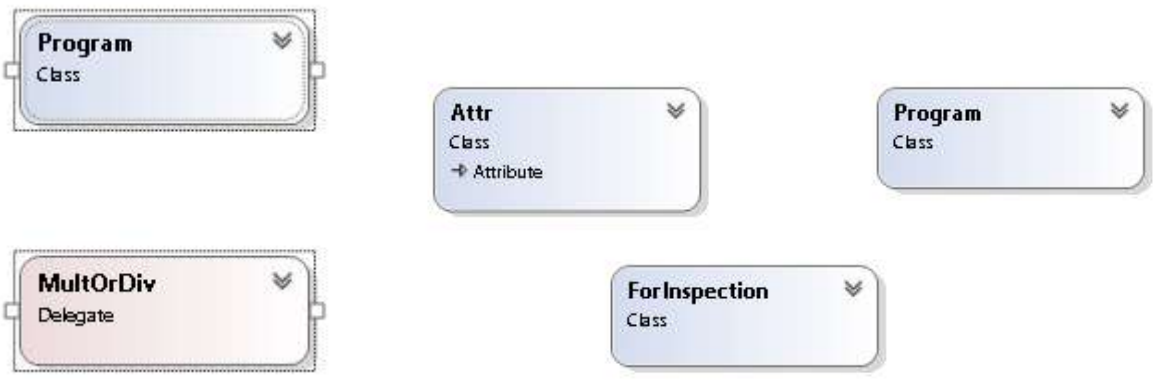
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.

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.

Диаграмма классов (6.1) и (6.2):



Текст программы:(6.1)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Laba6
{
    //Делегаты - аналог процедурного типа в Паскале.
    //Делегат - это не тип класса, а тип метода.
    //Делегат определяет сигнатуру метода (типы параметров и возвращаемого значения).
    //Если создается метод типа делегата, то у него должна быть сигнатура как у делегата.
    //Метод типа делегата можно передать как параметр другому методу.
    //Название делегата при объявлении указывается "вместо" названия метода
    delegate int MultOrDiv(int p1, int p2);
    class Program
    {
        //Методы, реализующие делегат (методы "типа" делегата)
        static int Mult(int p1, int p2) { return p1 * p2; }
        static int Div(int p1, int p2) { return p1 / p2; }
        /// <summary>
        /// Использование обощенного делегата Func<>
        /// </summary>
        static void MultOrDivFunc(string str, int i1, int i2, Func<int, int, int>
MultOrDivParam)
        {
            int Result = MultOrDivParam(i1, i2);
            Console.WriteLine(str + Result.ToString());
            // Func<int, string, bool> - делегат принимает параметры типа int и string и
возвращает bool
            // Если метод должен возвращать void, то используется делегат Action
            // Action<int, string> - делегат принимает параметры типа int и string и
возвращает void
            // Action как правило используется для разработки групповых делегатов,
которые используются в событиях
        }
        /// <summary>
        /// Использование делегата
        /// </summary>
        // метод с делегатным параметром
        static void MultOrDivMethod(string str, int i1, int i2, MultOrDiv MultOrDivParam)
        {
            //вызов (имя параметра как функция)
            int Result = MultOrDivParam(i1, i2);
            Console.WriteLine(str + Result.ToString());
        }
        static void Main(string[] args)
        {
            int i1 = 6; int i2 = 2;
            //вызов методов с делегатным параметром
            MultOrDivMethod("Умножение: ", i1, i2, Mult);
            MultOrDivMethod("Деление: ", i1, i2, Div);
            //Создание экземпляра делегата на основе метода (с помощью конструктора
делегатного типа)
            MultOrDiv md1 = new MultOrDiv(Mult);
            MultOrDivMethod("Создание экземпляра делегата на основе метода: ", i1, i2,
md1);
            //Создание экземпляра делегата на основе 'предположения' делегата
            //Компилятор 'пердполагает' что метод Mult типа делегата
            MultOrDiv md2 = Mult;
            MultOrDivMethod("Создание экземпляра делегата на основе 'предположения'
делегата: ", i1, i2, md2);
            //Создание анонимного метода
            MultOrDiv md3 = delegate (int param1, int param2)
```

```
            {
                return param1 * param2;
            };
            MultOrDivMethod("Создание экземпляра делегата на основе анонимного метода: ",
i1, i2, md2);
            MultOrDivMethod("Создание экземпляра делегата на основе лямбдавыражения 1: ",
i1, i2, (int x, int y) => { int z = x * y; return z; });
            MultOrDivMethod("Создание экземпляра делегата на основе лямбдавыражения 2: ",
i1, i2, (x, y) => { return x * y; });
            MultOrDivMethod("Создание экземпляра делегата на основе лямбдавыражения 3: ",
i1, i2, (x, y) => x * y);
            ////////////////////////////////////////////////////////
            Console.WriteLine("\n\nИспользование обощенного делегата Func<>");
            MultOrDivFunc("Создание экземпляра делегата на основе метода: ", i1, i2,
Mult);
            string OuterString = "ВНЕШНЯЯ ПЕРЕМЕННАЯ";
            MultOrDivFunc("Создание экземпляра делегата на основе лямбдавыражения 1: ",
i1, i2, (int x, int y) => { Console.WriteLine("Эта переменная объявлена вне
лямбдавыражения: " + OuterString); int z = x * y; return z; });
            MultOrDivFunc("Создание экземпляра делегата на основе лямбдавыражения 2: ",
i1, i2, (x, y) => { return x * y; });
            MultOrDivFunc("Создание экземпляра делегата на основе лямбдавыражения 3: ",
i1, i2, (x, y) => x * y);
            ////////////////////////////////////////////////////////
            //Групповой делегат всегда возвращает значение типа void
            Console.WriteLine("Пример группового делегата");
            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(6, 2);
            Action<int, int> group2 = a1; Console.WriteLine("Добавление вызова метода к
групповому делегату");
            group2 += a2;
            group2(10, 5);
            Console.WriteLine("Удаление вызова метода из группового делегата");
            group2 -= a1;
            group2(20, 10);
            Console.ReadLine();
        }
    }
}
```

Текст программы:(6.2)

(Program.cs)

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

namespace lab6._2_csh
{
    class Program
    {
        //Метод принимает в качестве параметров информацию о проверяемом свойстве
«PropertyInfo checkType»
        //и тип проверяемого атрибута «Type attributeType»
        //Если проверяемое свойство содержит атрибуты данного типа (метод
checkType.GetCustomAttributes возвращает
```

```
        //коллекцию isAttribute из более чем одного элемента), то метод возвращает
истину,
        //а в выходной параметр метода «out object attribute» помещается первый элемент
коллекции isAttribute
        //(в соответствии с определением, свойство NewAttribute может применяться к
свойству не более одного раза).
        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("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("\nПоля данных (public):");
            foreach (var x in t.GetFields())                  Console.WriteLine(x);
            Console.WriteLine("\nСвойства, помеченные атрибутом:");
            foreach (var x in t.GetProperties())
            {
                object attrObj;
                if (GetPropertyAttribute(x, typeof(Attr), out attrObj))
                {
                    Attr attr = attrObj as Attr;
                    Console.WriteLine(x.Name + " - " + attr.Description);
                }
            }
            Console.WriteLine("\nВызов метода:");
            //Создание объекта
            //ForInspection fi = new ForInspection();
            //Можно создать объект 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
            //Метод InvokeMember класса Type позволяет выполнять динамические действия с
объектами классов:
            //создавать объекты, вызывать методы, получать и присваивать значения свойств
и др.
            //Его особенность заключается в том, что имена свойств, классов передаются
методу InvokeMember в виде строковых параметров.
            object Result = t.InvokeMember("Mult", BindingFlags.InvokeMethod, null, fi,
parameters);
            Console.WriteLine("Mult(3,2)={0}", Result);
            Console.ReadLine();
        }
```

```
    }
}
```

(ForInspection.cs)

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

namespace lab6._2_csh
{
    class ForInspection
    {
        public ForInspection() { }
        public ForInspection(int i) { }
        public ForInspection(string str) { }
        public int Mult(int x, int y) { return x * y; }
        public int Div(int x, int y) { return x / y; }
        [Attr("Описание для property1")]
        public string property1
        {
            get { return _property1; }
            set { _property1 = value; }
        }
        private string _property1;
        public int property2 { get; set; }
        [Attr(Description = "Описание для property3")]
        public double property3 { get; private set; }
        public int field1;
        public float field2;
    }
}
```

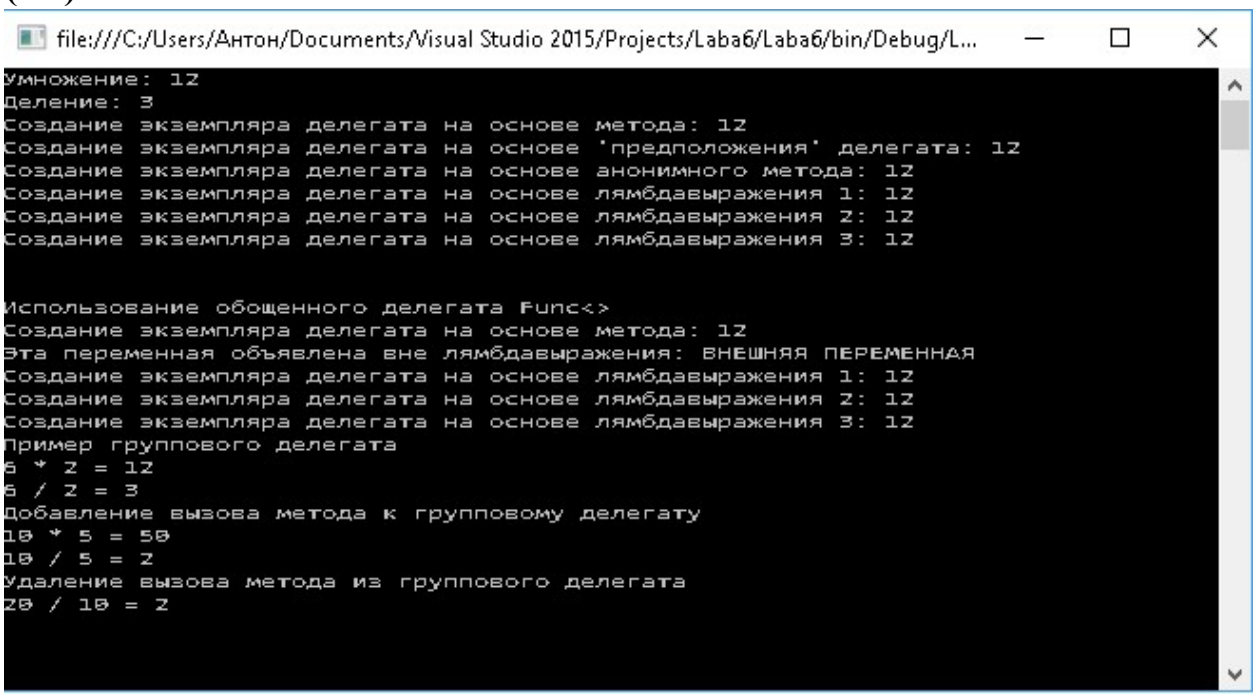
(Attr.cs)

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

namespace lab6._2_csh
{
    //класс атрибута должен быть, в свою очередь, помечен атрибутом AttributeUsage,
который принимает три параметра:
    //параметр типа перечисление AttributeTargets, которое указывает, к каким элементам
класса может применяться атрибут Attr (классам, свойствам, методам и т.д.); в данном
случае только к свойствам (Property);
    //логический параметр AllowMultiple, указывающий, может ли применяться к свойству
несколько атрибутов NewAttribute; в данном случае это запрещено;
    //логический параметр Inherited, указывающий, наследуется ли атрибут классами,
производными от класса с атрибутами; обычно используется значение false.
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
    class Attr: Attribute
    {
        //конструктор без параметров
        public Attr() { }
        //конструктор с параметром
        public Attr(string DescriptionParam)
        {
            Description = DescriptionParam;
        }
        //автоопределяемое свойство Description
        public string Description { get; set; }
    }}
```

Экранные формы:

(6.1)



(6.2)