**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Белова Е.О. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

г. Москва, 2018 г.

**Описание задания:**

**Часть 1. Разработать программу, использующую делегаты.**

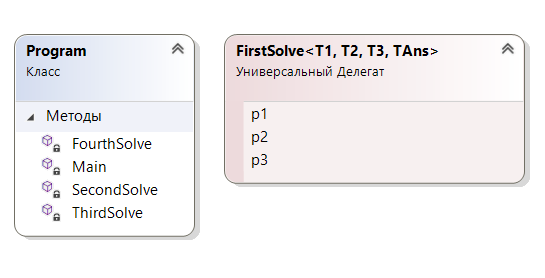
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, 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.**

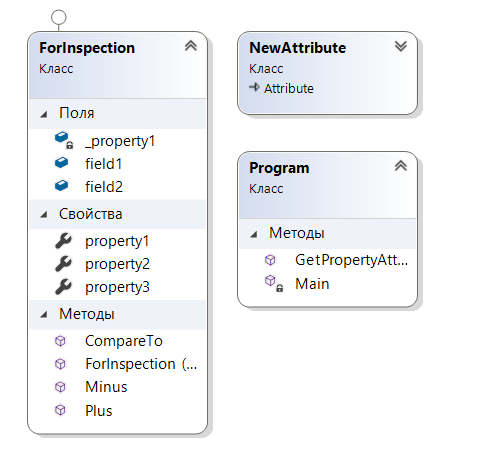
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute). Назначьте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

**Диаграмма классов:**

Часть 1



Часть 2



**Текст программы:**

Часть 1:

**Program.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegate

{

//2

public delegate TAns FirstSolve<T1, T2, T3, TAns>(T1 p1, T2 p2, T3 p3);

class Program

{

//3

static string SecondSolve(int a, double b, char c)

{

return a.ToString() + ' ' + b.ToString() + ' ' + c;

}

//4.0

static void ThirdSolve(

int a, FirstSolve<int, double, char, string> now\_ans)

{

if (a == 42)

Console.WriteLine(now\_ans(42, 42.42, '4'));

else

Console.WriteLine(now\_ans(666, 666.666, '6'));

}

//5.0

static void FourthSolve(

int a, Func<int , double, char, string> now\_ans)

{

if (a == 13)

Console.WriteLine(now\_ans(13, 13.13, '3'));

else

Console.WriteLine(now\_ans(444, 44.4444, '4'));

}

static void Main(string[] args)

{

//4.1

ThirdSolve(42, SecondSolve);

//4.2

ThirdSolve(1, ( a, b, c) =>

{

return (a.ToString() + ' ' + c + ' ' + b.ToString());

});

//5.1

FourthSolve(13, SecondSolve);

//5.2

FourthSolve(2, (int a, double b, char c) =>

{

return (a.ToString() + ' ' + c + ' ' + b.ToString());

});

}

}

}

Часть 2:

**Program.cs:**

using System;

using System.Reflection;

namespace Reflection

{

class Program

{

/// <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)

{

//3

ForInspection obj = new ForInspection();

Type t = typeof(ForInspection);

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties()) {

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

//5

Console.WriteLine("\nСвойства c 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);

}

}

//6

Console.WriteLine("\nВызов метода:");

//Создание объекта

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("Minus", BindingFlags.InvokeMethod,

null, fi, parameters);

Console.WriteLine("Minus(3,2)={0}", Result);

}

}

}

**NewAttribute.cs:**

using System;

namespace Reflection

{

/// <summary>

/// Класс атрибута

/// </summary>

[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:**

using System;

namespace Reflection

{

/// <summary>

/// Класс для исследования с помощью рефлексии

/// </summary>

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("property1 дан атрибут")]

public string property1

{

get { return \_property1; }

set { \_property1 = value; }

}

private string \_property1;

public int property2 { get; set; }

[NewAttribute(Description = "property3 дан атрибут")]

public double property3 { get; private set; }

public int field1;

public float field2;

/// <summary>

/// Реализация интерфейса IComparable

/// </summary>

public int CompareTo(object obj)

{

return 0;

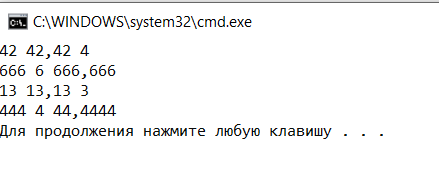
}

}

}

**Примеры:**

Часть 1:



Часть 2:

