Московский государственный технический университет

им. Н. Э. Баумана

**Отчет**

**по лабораторной работе №6**

Выполнил:

Студент группы ИУ5-35Б

Богданова В.В.

Проверил:

Преподаватель

Гапанюк Ю.Е.

2020г.

**1). Описание задания**

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

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

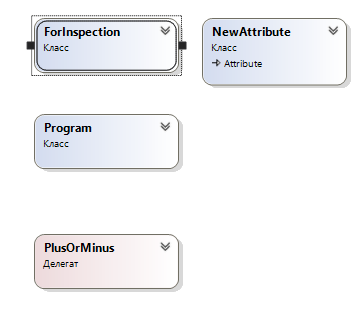
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

**2). Диаграмма классов**



**3). Текст программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace лаб6\_сишарп

{

/// <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; }

}

/// <summary>

/// Класс для исследования с помощью рефлексии

/// </summary>

public class ForInspection

{

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; }

[New("Ошибка")]

public string property1 { get; set; }

public int property2 { get; set; }

public bool property3 { get; set; }

public int field1;

public float field2;

}

delegate double PlusOrMinus(double p1, int p2);

class Program

{

//Методы, реализующие делегат (методы "типа" делегата)

static double Plus(double p1, int p2) { return p1 + p2; }

static double Minus(double p1, int p2) { return p1 - p2; }

/// <summary>

/// Использование обощенного делегата Func<>

/// </summary>

static void PlusOrMinusMethodFunc(string str, double i1, int i2, Func<double, int, double> PlusOrMinusParam)

{

double Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

}

/// <summary>

/// Использование делегата

/// </summary>

static void PlusOrMinusMethod(string str, double i1, int i2, PlusOrMinus PlusOrMinusParam)

{

double Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

}

static void Main(string[] args)

{

double i1 = 3.25;

int i2 = 2;

Console.WriteLine("Первый операнд:"+ i1.ToString());

Console.WriteLine("Второй операнд:" + i2.ToString()+'\n');

PlusOrMinusMethod("Сумма: ", i1, i2, Plus);

PlusOrMinusMethod("Разность: ", i1, i2, Minus);

Console.WriteLine('\n');

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 1: ", i1, i2,

(double x, int y) =>

{

double z = x + y;

return z;

}

);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 2: ", i1, i2,

(x, y) =>

{

return x + y;

}

);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 3: ", i1, i2, (x, y) => x + y);

Console.WriteLine('\n');

Console.WriteLine("\n\nИспользование обощенного делегата Func<>");

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе метода: ", i1, i2, Plus);

Console.WriteLine('\n');

Console.WriteLine("Использования рефлексии для класса"+'\n');

ForInspection f = new ForInspection();

Type t = f.GetType();

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('\n');

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine('\n');

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine('\n');

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine('\n');

Console.WriteLine("\nПоля данных (public):");

foreach (var x in t.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine('\n');

Console.WriteLine("\nСвойства, помеченные атрибутом:");

//Поиск атрибутов с заданным типом

foreach (var x in t.GetProperties())

{

var isAttribute = x.GetCustomAttributes(typeof(NewAttribute), false);

if (isAttribute.Length != 0)

{

NewAttribute attr = isAttribute[0] as NewAttribute;

Console.WriteLine(x.Name + " - " + attr.Description);

}

}

Console.WriteLine('\n'+"Вызов метода с помощью рефлексии:");

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);

}

}

}

**4). Пример выполнения**

