**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

«Трек курса «Задания на основе языка C#»»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Драгун И.А. |  | Ю.Е. Гапанюк. |
| Подпись и дата:  28.12.2020 |  | Подпись и дата:  28.12.2020 |

Москва, 2020 г.

**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.

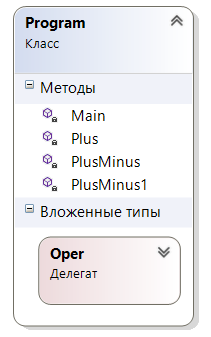
**Часть 2. Разработать программу, реализующую работу с рефлексией.**

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

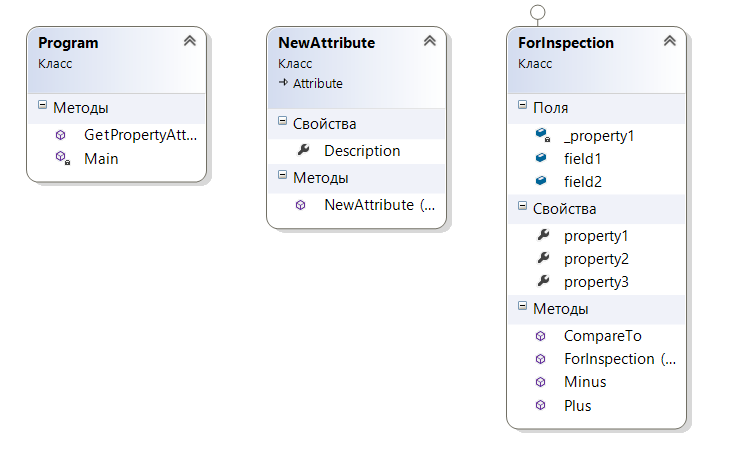
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5.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

**Диаграмма классов**

**Часть 1.**



**Часть 2.**



**Текст программы**

**Часть 1.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6\_1

{

class Program

{

delegate double Oper(double a, int b);

static void PlusMinus1(

string str, double a, int b,

Func<double, int, double> param)

{

double Result = param(a, b);

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

}

static double Plus(double a, int b)

{

return a + b;

}

static void PlusMinus(

string str,

double a,

int b,

Oper param)

{

double Result = param(a, b);

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

}

static void Main(string[] args)

{

Console.WriteLine("Вызов через метод");

//Вызов через метод

PlusMinus("Плюс", 7.2, 10, Plus);

Console.WriteLine("Вызов через лямбда-выражение");

//вызов через лямбда

PlusMinus("Плюс", 7.2, 10, (a, b) => a + b);

//Func

Console.WriteLine("Использование Func");

PlusMinus1("Плюс", 7.2, 10, Plus);

Console.Read();

}

}

}

**Часть 2.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Lab6\_1\_2

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Вывод информации о сборке:");

Assembly i = Assembly.GetExecutingAssembly();

Console.WriteLine("Полное имя:" + i.FullName);

Console.WriteLine("Исполняемый файл:" + i.Location);

ForInspection obj = new ForInspection();

Type t1 = obj.GetType();

Console.WriteLine("\nИнформация о типе:");

Console.WriteLine("Тип " + t1.FullName + " унаследован от " + t1.BaseType.FullName);

Console.WriteLine("Пространство имен " + t1.Namespace);

Console.WriteLine("Находится в сборке " + t1.AssemblyQualifiedName);

Console.WriteLine("\nКонструкторы:");

foreach (var x in t1.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t1.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t1.GetProperties())

{

Console.WriteLine(x);

}

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

foreach (var x in t1.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine("\nForInspection реализует IComparable -> " + t1.GetInterfaces().Contains(typeof(IComparable)));

Console.Read();

//---------------------------------------------------------

Console.WriteLine("---------------------------------------------------------");

Type t2 = typeof(ForInspection);

Console.WriteLine("\nВызов метода:");

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

foreach (var x in t2.GetProperties())

{

object attrObj;

if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))

{

NewAttribute attr = attrObj as NewAttribute;

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

}

}

//Создание объекта

//ForInspection fi = new ForInspection();

//Можно создать объект через рефлексию

ForInspection fi = (ForInspection)t2.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

//Параметры вызова метода

object[] parameters = new object[] { 3, 2 };

//Вызов метода

object Result = t2.InvokeMember("Plus", BindingFlags.InvokeMethod, null, fi, parameters);

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

}

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;

}

}

/// Класс атрибута

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

// Класс для исследования с помощью рефлексии

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("Описание для property1")]

public string property1

{

get { return \_property1; }

set { \_property1 = value; }

}

private string \_property1;

public int property2 { get; set; }

[NewAttribute(Description = "Описание для property3")]

public double property3 { get; private set; }

public int field1;

public float field2;

/// Реализация интерфейса IComparable

public int CompareTo(object obj)

{

return 0;

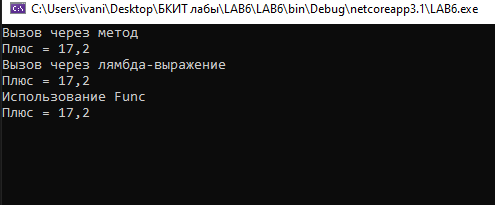
}

}

}

**Экранные формы с примерами выполнения программы**

**Часть 1.**



**Часть 2.**

