МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. Н. Э. БАУМАНА

Факультет "Информатика и системы управления"

### Кафедра ИУ5 “Системы обработки информации и управления”

Группа ИУ5-32Б

Отчёт по лабораторной работе №6

«Работа с делегатами и атрибутами»

Студент: Преподаватель:

Ваганов Даниил Дмитриевич Гапанюк Юрий Евгеньевич

Дата: \_\_\_.\_\_\_.\_\_\_\_\_\_ Дата: \_\_\_.\_\_\_.\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_ Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Условие задания:**

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.

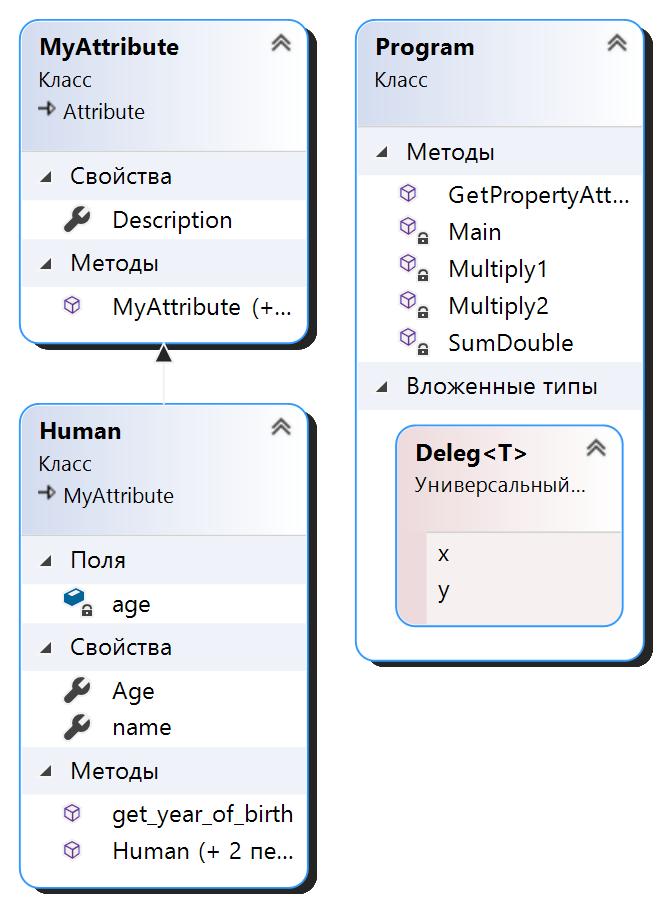
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.

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:**



**Текст программы:**

using System;

using System.Reflection;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LAB6\_BKIT

{

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

public class MyAttribute : Attribute

{

public string Description { get; set; }

public MyAttribute() { }

public MyAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

}

class Human : MyAttribute

{

[MyAttribute(Description = "Описание для имени")]

public string name { get; set; } = "Unknown";

int age;

[MyAttribute("Описание для возраста")]

public int Age

{

get { return age; }

set { age = value; }

}

public Human()

{

this.age = 0;

}

public Human(string name)

{

this.name = name;

}

public Human(string name, int age)

{

this.name = name;

this.age = age;

}

public int get\_year\_of\_birth()

{

return (int.Parse(DateTime.Now.ToString("yyyy")) - this.age);

}

}

class Program

{

public delegate T Deleg<T>(int x, double y);

static double SumDouble(int a, double b)

{

return (double)a + b;

}

static double Multiply1(double koeff\_multiply, int x, double y, Deleg<double> \_del)

{

return koeff\_multiply \* \_del(x, y);

}

static double Multiply2(double koeff\_multiply, int x, double y, Func<int, double, double> \_del)

{

return koeff\_multiply \* \_del(x, y);

}

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

var isAttribute =

checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void Main(string[] args)

{

Deleg<double> del;

del = SumDouble;

Console.WriteLine("Вывод работы метода используя делегат: " + del(2, 3));

Console.WriteLine("\nМетод, один параметр которого делегат: " + Multiply1(5, 5, 4, del));

Console.WriteLine("Лямбда-выражение: " + Multiply1(5, 5, 4.78, (x, y) => x \* y));

Func<int, double, double> retfunc = SumDouble;

Console.WriteLine("\nМетод, один параметр которого обобщенный делегат Func: " + Multiply2(10, 5, 4, retfunc));

Console.WriteLine("Лямбда-выражение: " + Multiply2(5, 5, 4, (int x, double y) => { return x \* y; }));

Human Steve = new Human("Steve", 35);

Type s = Steve.GetType();

Console.WriteLine("\n--------------------ЧАСТЬ 2--------------------\n");

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

foreach (var i in s.GetConstructors())

{

Console.WriteLine(i);

}

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

foreach (var i in s.GetProperties())

{

Console.WriteLine(i);

}

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

foreach (var i in s.GetMethods())

{

Console.WriteLine(i);

}

Type t = typeof(Human);

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

foreach (var x in t.GetProperties())

{

object attrObj;

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

{

MyAttribute attr = attrObj as MyAttribute;

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

}

}

Console.WriteLine("\nВызов метода:");

Human h = (Human)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

h.Age = 25;

object[] parameters = new object[] {};

object Result = t.InvokeMember("get\_year\_of\_birth", BindingFlags.InvokeMethod,

null, h, parameters);

Console.WriteLine("get\_year\_of\_birth = {0}", Result);

}

}

}

**Анализ результатов:**

