**Министерство науки и высшего образования Российской** **Федерации** **Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Московский государственный технический университет** **имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Радиотехнический»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Лабораторная работа №6

по дисциплине «Базовые компоненты интернет-технологий» на тему:

«Делегаты и рефлексия в C#»

Выполнил:

студент группы РТ5-31Б

В. И. Плешаков

Проверил:

Ю. Е. Гапанюк

2020 г.

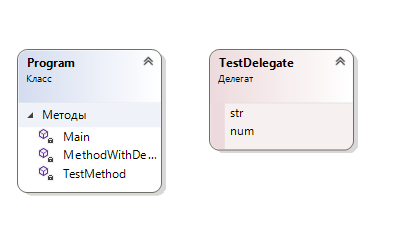
Задание:

Часть 1. Разработать программу, использующую делегаты

Часть 2. Разработать программу, реализующую работу с рефлексией

**Часть 1.**

Диаграмма классов:

****

Текст программы:

using System;

namespace Lab6\_2020

{

delegate string TestDelegate(string str, int num);// Объявление делегата

class Program

{

static string TestMethod(string str, int num)// Объявление метода, соответствующего делегату

{

str += "a";

num += 10;

return str + num.ToString();

}

static void MethodWithDelegate(int num1, int num2, TestDelegate DelegateParam)// Метод с делегатным параметром

{

string str = num1.ToString();

Console.WriteLine(DelegateParam(str, num2));

}

static void Main(string[] args)

{

MethodWithDelegate(12, 22, TestMethod);// Вызов метода с делегатным параметром в виде метода

// Вызов метода с делегатным параметром в виде лямбда-выражения

MethodWithDelegate(12, 22,

(string str, int num) =>

{

num \*= 4;

return num.ToString() + str;

}

);

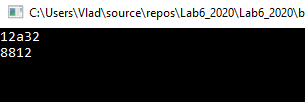
Console.ReadLine();

}

}

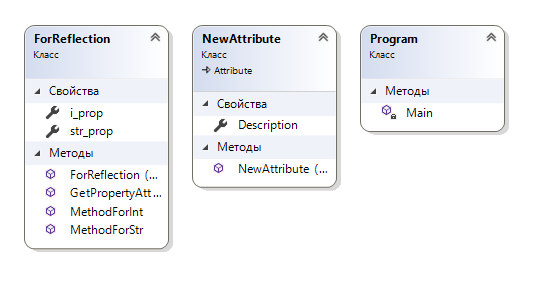
}

Пример выполнения программы:



**Часть 2.**

Диаграмма классов:



Текст программы:

using System;

using System.Reflection;

namespace Lab6\_2020\_2

{

class ForReflection

{

public ForReflection()// Конструктор без параметров

{

i\_prop = 1;

str\_prop = "a";

}

public ForReflection(int i\_param, string str\_param)// Конструктор с двумя параметрами

{

i\_prop = i\_param;

str\_prop = str\_param;

}

[NewAttribute("Целочисленное свойство")]

public int i\_prop { get; set; } // Целочисленное свойтсво

public string str\_prop { get; set; } // Строковое свойство

public void MethodForStr()

{

Console.WriteLine(str\_prop);

}

public int MethodForInt(int param)

{

return (i\_prop + param);

}

/// <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;

}

}

class Program

{

static void Main(string[] args)

{

Type t = typeof(ForReflection);

// Вывод информации о конструкторах

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

// Вывод информации о методах

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

// Вывод информации о свойствах

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("\nРабота с атрибутами:");

// Выводятся только свойства с атрибутами

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

foreach (var x in t.GetProperties())

{

object attrObj;

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

{

NewAttribute attr = attrObj as NewAttribute;

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

}

}

//Вызов метода класса с помощью рефлексии

Console.WriteLine("\nВызов метода:");

//Создание объекта

ForReflection fr = (ForReflection)t.InvokeMember( null, BindingFlags.CreateInstance, null, null, new object[] { 12, "abc" });

//Параметры вызова метода

object[] parameters = new object[] { 10 };

//Вызов метода

object Result = t.InvokeMember("MethodForInt", BindingFlags.InvokeMethod, null, fr, parameters);

Console.WriteLine("MethodForInt(10)={0}", Result);

Console.ReadLine();

}

}

/// <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; }

}

}

Пример выполнения программы:

