**Московский государственный технический университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

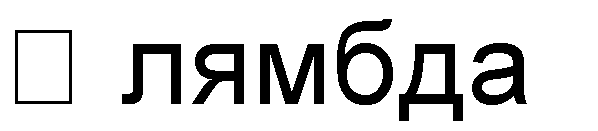
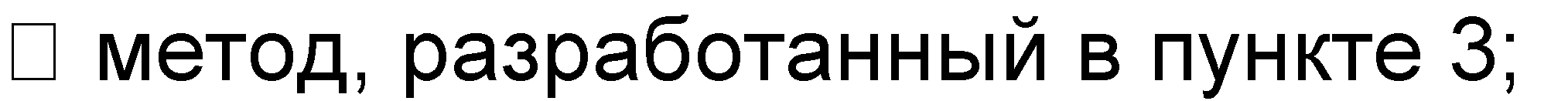
Работу выполнила: Корнеева Анна, ИУ5-34

г. Москва, 2017 г.

**Задание**

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 входных параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:



-

выражение.

1. 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.

1. Создайте класс атрибута (унаследован от класса System.Attribute).
2. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
3. Вызовите один из методов класса с использованием рефлексии. **Текст программы**

Часть 1:

using System;

using System.Collections.Generic; using System.Linq; using System.Text;

using System.Threading.Tasks;

namespace Лаб6

{

delegate int PlusOrMinus(int p1, int p2); class Program

{

//Методы, реализующие делегат (методы "типа" делегата) static int Plus(int p1, int p2) { return p1 + p2; } static int Minus(int p1, int p2) { return p1 - p2; }

//Пример объявления метода с делегатным параметром static void PlusOrMinusMethod(string str, int i1, int i2, PlusOrMinus PlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2); Console.WriteLine(str + Result.ToString());

}

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

{

int Result = PlusOrMinusParam(i1, i2); Console.WriteLine(str + Result.ToString());

}

static void Main(string[] args)

{ int i1 = 3; int i2 = 2;

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

PlusOrMinusMethod("Минус: ", i1, i2, Minus); //Создание экземпляра делегата на основе метода

PlusOrMinus pm1 = new PlusOrMinus(Plus);

PlusOrMinusMethod("Создание экземпляра делегата на основе метода: ",

i1, i2, pm1);

//Создание экземпляра делегата на основе 'предположения' делегата

//Компилятор 'пердполагает' что метод Plus типа делегата PlusOrMinus pm2 = Plus;

PlusOrMinusMethod("Создание экземпляра делегата на основе

'предположения' делегата: ", i1, i2, pm2);

//Создание анонимного метода

PlusOrMinus pm3 = delegate (int param1, int param2)

{

return param1 + param2;

};

PlusOrMinusMethod("Создание экземпляра делегата на основе анонимного метода: ", i1, i2, pm2);

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

(int x, int y) =>

{

int z = x + y; return z;

}

);

//Для обобщённого делегата PlusOrMinusMethodFunc("Создание экземпляра делегата на основе метода: ", i1, i2, Minus);

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

Console.ReadKey();

}

}

}

Часть 2:

using System;

using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;

using System.Reflection;

namespace Лаб\_6.\_2

{

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)

{

ForInspection obj = new ForInspection();

Type t = obj.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("\nКонструкторы:"); foreach (var x in t.GetConstructors()) { Console.WriteLine(x); }

Console.WriteLine("\nМетоды:"); foreach (var x in t.GetMethods())

{ Console.WriteLine(x); }

Console.WriteLine("\nСвойства:"); foreach (var x in t.GetProperties())

{ Console.WriteLine(x); }

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

foreach (var x in t.GetFields())

{ Console.WriteLine(x); }

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

Console.WriteLine("\nСвойства, помеченные 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);

}

}

Console.WriteLine("\nВызов метода:");

//Создание объекта

//ForInspection fi = new ForInspection();

//Можно создать объект через рефлексию

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);

Console.ReadLine();

}

}

/// <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("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;

/// <summary>

/// Реализация интерфейса IComparable

/// </summary>

public int CompareTo(object obj) { return 0; }

}

/// <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; }

}

}

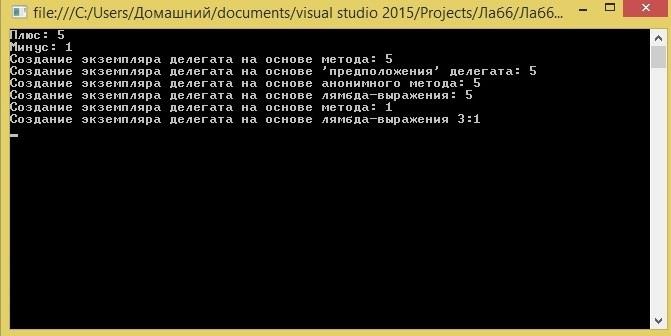
# Диаграмма классов

Часть 1:



# Результат

Часть 1:



Часть 2:

