**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Дисциплина «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы  ИУ5-34Б |  | преподаватель каф. ИУ5 |
| Зонова Анна |  | Гапанюк Ю.Е. |

**Часть 1**

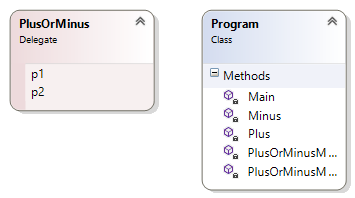
**Описание условия:**

**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 входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   * метод, разработанный в пункте 3;
   * лямбда-выражение.

Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

**Диаграмма классов**



**Текст программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegates

{

//Делегаты - аналог процедурного типа в Паскале.

//Делегат - это не тип класса, а тип метода.

//Делегат определяет сигнатуру метода (типы параметров и возвращаемого значения).

//Если создается метод типа делегата, то у него должна быть сигнатура как у делегата.

//Метод типа делегата можно передать как параметр другому методу.

//Название делегата при объявлении указывается "вместо" названия метода

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; }

/// <summary>

/// Использование обощенного делегата Func<>

/// </summary>

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

{

int Result = PlusOrMinusParam(i1, i2);

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

// Func<int, string, bool> - делегат принимает параметры типа int и string и возвращает bool

// Если метод должен возвращать void, то используется делегат Action

// Action<int, string> - делегат принимает параметры типа int и string и возвращает void

// Action как правило используется для разработки групповых делегатов, которые используются в событиях

}

/// <summary>

/// Использование делегата

/// </summary>

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

//Лямбда-выражение в виде переменной

PlusOrMinus pm4 = (int x, int y) =>

{

int z = x + y;

return z;

};

int test = pm4(1, 2);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения в виде переменной: ", i1, i2, pm4);

//Пример использования внешней переменной

int outer = 100;

PlusOrMinus pm5 = (int x, int y) =>

{

int z = x + y + outer;

return z;

};

int test2 = pm5(1, 2);

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

(int x, int y) =>

{

int 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\nИспользование обощенного делегата Func<>");

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

string OuterString = "ВНЕШНЯЯ ПЕРЕМЕННАЯ";

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

(int x, int y) =>

{

Console.WriteLine("Эта переменная объявлена вне лямбда-выражения: " + OuterString);

int z = x + y;

return z;

}

);

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

(x, y) =>

{

return x + y;

}

);

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

//////////////////////////////////////////////////////////////

//Групповой делегат всегда возвращает значение типа void

Console.WriteLine("Пример группового делегата");

Action<int, int> a1 = (x, y) => { Console.WriteLine("{0} + {1} = {2}", x, y, x + y); };

Action<int, int> a2 = (x, y) => { Console.WriteLine("{0} - {1} = {2}", x, y, x - y); };

Action<int, int> group = a1 + a2;

group(7, 10);

Action<int, int> group2 = a1;

Console.WriteLine("Добавление вызова метода к групповому делегату");

group2 += a2;

group2(8, 4);

Console.WriteLine("Удаление вызова метода из группового делегата");

group2 -= a1;

group2(30, 15);

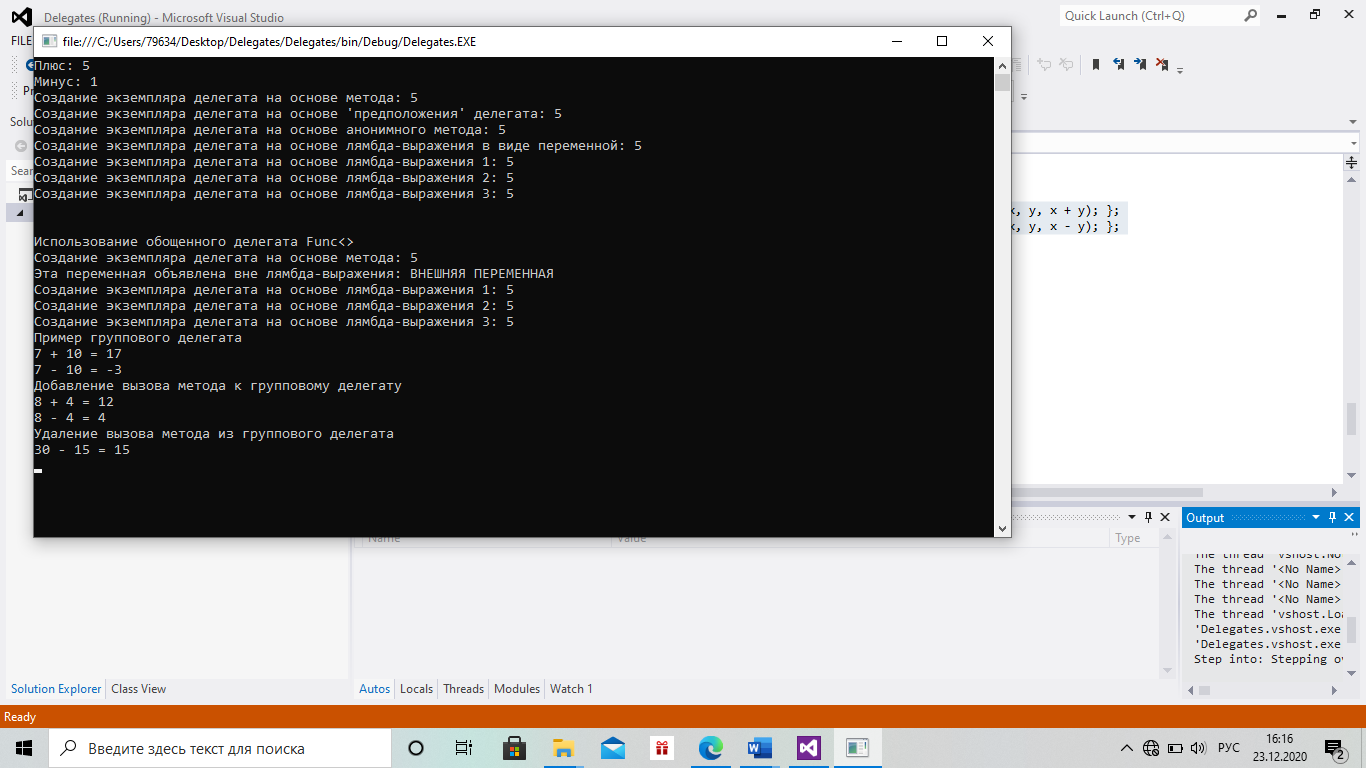
Console.ReadLine();

}

}

}

**Пример экранных форм**



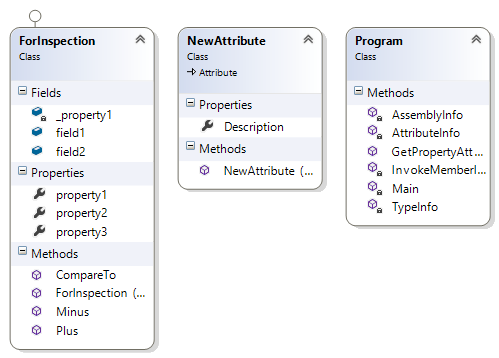
**Часть 2**

**Описание условия:**

**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.
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**

Forlnspection.cs:

using System;

namespace Reflection

{

/// <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;

}

}

}

NewAttribute.cs:

using System;

namespace Reflection

{

/// <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; }

}

}

Program.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Reflection

{

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;

}

/// <summary>

/// Получение информации о текущей сборке

/// </summary>

static void AssemblyInfo()

{

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

Assembly i = Assembly.GetExecutingAssembly();

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

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

}

/// <summary>

/// Получение информации о типе

/// </summary>

static void TypeInfo()

{

ForInspection obj = new ForInspection();

Type t = obj.GetType();

//другой способ

//Type t = typeof(ForInspection);

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))

);

}

/// <summary>

/// Пример использования метода InvokeMember

/// </summary>

static void InvokeMemberInfo()

{

Type t = typeof(ForInspection);

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);

}

/// <summary>

/// Работа с атрибутами

/// </summary>

static void AttributeInfo()

{

Type t = typeof(ForInspection);

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);

}

}

}

static void Main(string[] args)

{

AssemblyInfo();

TypeInfo();

InvokeMemberInfo();

AttributeInfo();

Console.ReadLine();

}

}

}

**Пример экранных форм**

