МГТУ им. Н.Э. Баумана

Факультет «Информатика и системы управления»

ДИСЦИПЛИНА:

«Электротехника»

Отчет по лабораторной работе №6

Выполнила:

Студент 2 курса

Факультет ИУ

Группа ИУ5-31Б

Кемерова А.М.

Преподаватель:

Белодедов М.В.

**Цель лабораторной работы**

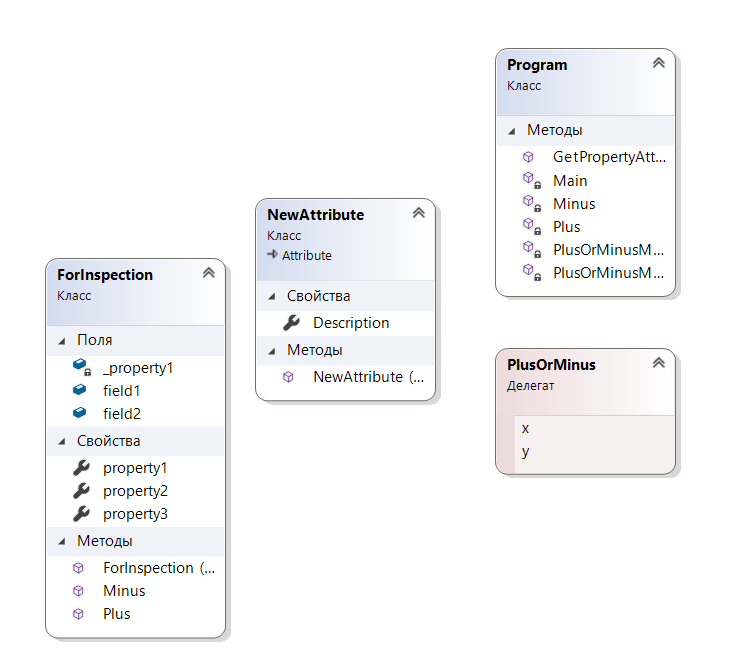
**Часть 1. Разработать программу, использующую делегаты.**

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 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.**

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



**Код программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Lab6.\_1

{

delegate double PlusOrMinus(double x, double y);

class Program

{

static double Plus(double x, double y) { return x + y; }

static double Minus(double x, double y) { return x - y; }

static void PlusOrMinusMethodFunc(string str, double a, double b, Func<double, double, double> PlusOrMinusParam)

{

double Result = PlusOrMinusParam(a, b);

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

}

static void PlusOrMinusMethod(string str, double i1, double i2, PlusOrMinus PlusOrMinusParam)

{

double Result = PlusOrMinusParam(i1, i2);

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

}

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)

{

Console.WriteLine("ДЕЛЕГАТЫ\n");

double i1 = 6, i2 = 8;

PlusOrMinusMethod("6 + 8 = ", i1, i2, Plus);

PlusOrMinusMethod("6 - 8 = ", i1, i2, Minus);

Console.WriteLine("\n");

// Создание экземпляра делегата на основе метода

PlusOrMinus pm1 = new PlusOrMinus(Plus);

PlusOrMinusMethod("Создание экземпляра делегата на основе метода: 6 + 8 = ", 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: 6 + 8 = ", i1, i2, pm2);

// Создание анонимного метода

PlusOrMinus pm3 = delegate (double param1, double param2)

{

return param1 + param2;

};

PlusOrMinusMethod("Создание экземпляра делегата на основе анонимного метода: 6 + 8 = ", i1, i2, pm2);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбдавыражения 1: 6 + 8 = ", i1, i2,

(double x, double y) =>

{

return x + y;

});

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбдавыражения 2: 6 + 8 = ", i1, i2, (x, y) => { return x + y; });

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбдавыражения 3: 6 + 8 = ", i1, i2, (x, y) => x + y);

Console.WriteLine("\nИспользование обощенного делегата Func<>");

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе метода: 6 + 8 = ", i1, i2, Plus);

string OuterString = "ВНЕШНЯЯ ПЕРЕМЕННАЯ";

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбдавыражения 1: 6 + 8 = ", i1, i2,

(double x, double y) =>

{

Console.WriteLine("Эта переменная объявлена вне лямбдавыражения: " + OuterString);

return x + y;

});

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбдавыражения 2: 6 + 8 = ", i1, i2,

(x, y) =>

{

return x + y;

});

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбдавыражения 3: 6 + 8 = ", i1, i2, (x, y) => x + y);

// Групповой делегат всегда возвращает значение типа void

Console.WriteLine("\nПример 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, 30);

Action<int, int> group2 = a1;

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

group2 += a2;

group2(43, 12);

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

group2 -= a1;

group2(7, 85);

// ЧАСТЬ 2

Console.WriteLine("\nРЕФЛЕКСИЯ\n");

Type t = typeof(ForInspection);

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("\nСвойства, помеченные атрибутом:");

foreach (var x in t.GetProperties())

{

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

{

NewAttribute attr = attrObj as NewAttribute;

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

}

}

Console.WriteLine("\nВызов метода:");

// Создание объекта через рефлексию

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

// Параметры вызова метода

object[] parameters = new object[] { 77, 55 };

// Вызов метода

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

Console.WriteLine("Plus(77,55)={0}", Result);

Console.ReadLine();

}

}

}

**ForInspection.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6.\_1

{

/// <summary>

/// Класс для исследования с помощью рефлексии

/// </summary>

public class ForInspection

{

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;

}

}

**NewAttribute.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6.\_1

{

/// <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; }

}

}

**Результаты работы программы**

