***Отчет по лабораторной работе № 6 по курсу С#***

7

(количество листов)

Студент группы ИУ5-34:

Тюлькина Нина

Подпись Дата

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Руководитель: Гапанюк Ю.Е.

Подпись Дата

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Москва, 2017

*Условие лабораторной работы:*

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

(В качестве примера можно использовать проект «Delegates»).

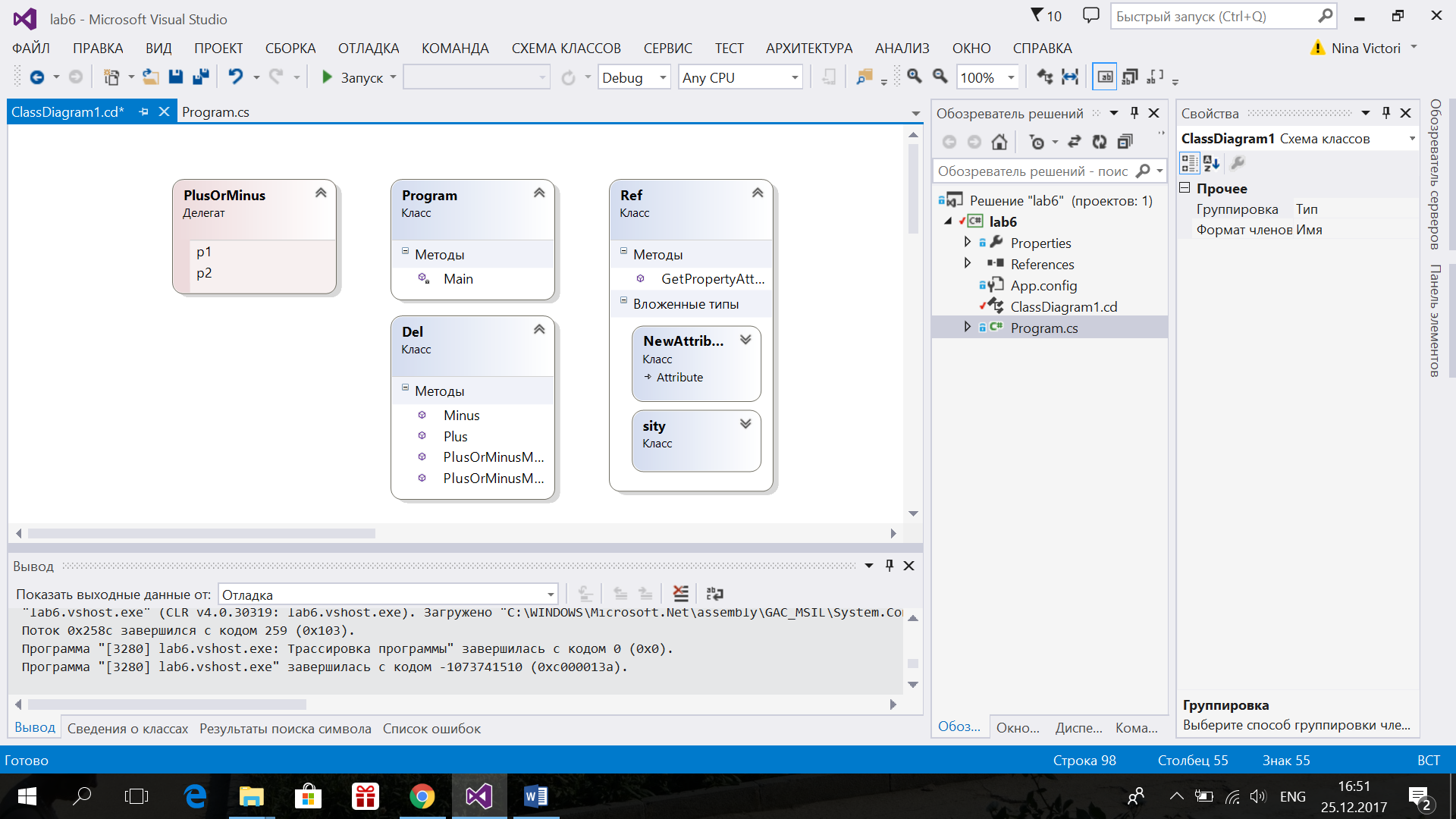
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, 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.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace lab6

{

class Program

{

static void Main(string[] args)

{

//---------------------------------РЕАЛИЗАЦИЯ ДЕЛЕГАТОВ---------------------------------

int a = 3;

int b = 2;

Console.WriteLine("a = " + a.ToString());

Console.WriteLine("b = " + b.ToString());

Console.WriteLine("Вычисление a - b различными способами ");

Console.WriteLine("\nИспользуемый далее метод принимает делегат в качестве одного из входных параметров");

Delegates.Del.PlusOrMinusMethod("Параметр - метод, соответствующий делегату: ", a, b, Delegates.Del.Minus);

Delegates.Del.PlusOrMinusMethod("Параметр - лямбда-выражение 1: ", a, b,

(int x, int y) =>

{

int z = x - y;

return z;

}

);

Delegates.Del.PlusOrMinusMethod("Параметр - лямбда-выражение 2: ", a, b,

(x, y) =>

{

return x - y;

}

);

Delegates.Del.PlusOrMinusMethod("Параметр - лямбда-выражение 3: ", a, b, (x, y) => x - y);

Console.WriteLine("\nИспользуемый далее метод принимает обобщенный детегат Func<> в качестве одного из входных параметров");

Delegates.Del.PlusOrMinusMethodFunc("Параметр - метод, соответствующий делегату: ", a, b, Delegates.Del.Minus);

Delegates.Del.PlusOrMinusMethodFunc("Параметр - лямбда-выражение 1: ", a, b,

(int x, int y) =>

{

int z = x - y;

return z;

}

);

Delegates.Del.PlusOrMinusMethodFunc("Параметр - лямбда-выражение 2: ", a, b,

(x, y) =>

{

return x - y;

}

);

Delegates.Del.PlusOrMinusMethodFunc("Параметр - лямбда-выражение 3: ", a, b, (x, y) => x - y);

//---------------------------------РЕАЛИЗАЦИЯ РЕФЛЕКСИИ---------------------------------

for (int i = 0; i < 100; i+=1)

Console.Write("-");

Console.WriteLine("\n");

Type t = typeof(Reflection.Ref.sity);

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())

{

object attrObj;

if (Reflection.Ref.GetPropertyAttribute(x, typeof(Reflection.Ref.NewAttribute), out attrObj))

{

Reflection.Ref.NewAttribute attr = attrObj as Reflection.Ref.NewAttribute;

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

}

}

Console.WriteLine("\nВызов метода:");

//Создание объекта

//Можно создать объект через рефлексию

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

//Параметры вызова метода

object[] parameters = new object[] { 3, 2 };

object Result = t.InvokeMember("integration", BindingFlags.InvokeMethod, null, fi, parameters); //Вызов метода

Console.WriteLine("integration(30000,20000)={0}", Result);

Console.ReadLine();

}

}

}

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

public delegate int PlusOrMinus(int p1, int p2);

class Del

{

//Методы, реализующие делегат (методы "типа" делегата)

public static int Plus(int p1, int p2) { return p1 + p2; }

public static int Minus(int p1, int p2) { return p1 - p2; }

static public void PlusOrMinusMethodFunc(string str, int i1, int i2, Func<int, int, int> PlusOrMinusParam) // Использование обощенного делегата Func<>

{

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

}

static public void PlusOrMinusMethod(string str, int i1, int i2, PlusOrMinus PlusOrMinusParam) /// Использование делегата

{

int Result = PlusOrMinusParam(i1, i2);

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

}

}

}

namespace Reflection

{

class Ref

{

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute) // Проверка, что у свойства есть атрибут заданного типа

{

bool Result = false;

attribute = null;

var isAttribute = checkType.GetCustomAttributes(attributeType, false); //Поиск атрибутов с заданным типом

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

// Класс атрибута

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

public class NewAttribute:Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

public class sity

{

private int \_numbers;

private string \_title;

public sity() { }

public sity(int c, string s) {

numbers = c;

title = s;

}

public sity(string s, int c)

{

numbers = c;

title = s;

}

public int numbers

{

get { return \_numbers; }

private set { \_numbers = value; }

}

public string title

{

get { return \_title; }

private set { \_title = value; }

}

public int integration(int x, int y) { return x + y; }

public int area;

public int established\_in;

public string mayor;

}

}

}

*Вывод*

