**Отчет по Лабораторной работе №6**

Студент группы ИУ5-34

Тимаков Анатолий

Дата: 27.12.17

**Формулировка задачи**

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 типови возвращающий значение произвольного типа.

3. Напишите метод, соответствующий данному делегату.

4. Напишите метод, принимающий разработанный Вами делегат, в качествеодного из входным параметров. Осуществите вызов метода, передавая вкачестве 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.

**Код программы**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Lab6

{

class Program

{

delegate double PlusOrMinus(double a, int b);

static void PlusOrMinusMethod(string str, double a, int b, PlusOrMinus ProsCons)

{

double result = ProsCons(a, b);

Console.WriteLine(str + result);

}

static double Plus(double a, int b) { return a + b; }

static double Minus(double a, int b) { return a - b; }

static void PlusOrMinusMethod2(string str, double a, int b, Func<double, int, double> ProsCons) //generalized delegate Func - announce in method

{

double result = ProsCons(a, b);

Console.WriteLine(str + result);

}

public static bool GetAttributeProperty(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

//Search for attributes with exact type

var isAttribute = checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

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

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

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("Description for property1")]

public string property1 { get; set; }

public int property2 { get; set; }

[NewAttribute(Description = "Description for property3")]

public double property3 { get; private set; }

public int field1;

public float field2;

public int CompareTo(object obj)

{

return 0;

}

}

static void Main(string[] args)

{

double a = 3.14;

int b = 5;

PlusOrMinusMethod("Plus: ", a, b, Plus);

PlusOrMinusMethod("Minus: ", a, b, Minus);

Console.WriteLine("\n");

PlusOrMinusMethod("Plus with long lambda expression: ", a, b, (double c, int d) =>

{

double res = c + d;

return res;

}

);

PlusOrMinusMethod("Minus with short lambda expression: ", a, b, (c, d) => c - d);

Console.WriteLine("\n");

PlusOrMinusMethod2("Plus. Delegate Func<> (Method Plus): ", a, b, Plus);

PlusOrMinusMethod2("Minus. Delegate Func<> (lambda expression): ", a, b, (c, d) => c - d);

Console.WriteLine("\n");

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;

Console.Write("a1(5, 3): ");

a1(5, 3);

Console.WriteLine("group(5, 3): ");

group(5, 3);

Console.WriteLine("\n");

Action<int, int> group2 = a1;

Console.WriteLine("Add method call to group2: ");

group2 += a2;

group2(2, 3);

Console.WriteLine("Delete method call from group2: ");

group2 -= a1;

group2(2, 3);

Console.ReadKey();

Console.Clear();

Type t = typeof(ForInspection);

Console.WriteLine("Constructors: ");

foreach (var x in t.GetConstructors()) Console.WriteLine(x);

Console.WriteLine("\nMethods: ");

foreach (var x in t.GetMethods()) Console.WriteLine(x);

Console.WriteLine("\nProperties: ");

foreach (var x in t.GetProperties()) Console.WriteLine(x);

Console.WriteLine("\nPublic fields: ");

foreach (var x in t.GetFields()) Console.WriteLine(x);

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

Console.WriteLine("\nProperties with attributes: ");

foreach (var x in t.GetProperties())

{

object attrObj;

if (GetAttributeProperty(x, typeof(NewAttribute), out attrObj))

{

NewAttribute attr = attrObj as NewAttribute; //type conversion

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

}

}

Console.WriteLine("\nMethod call:");

//ForInspection fi = new ForInspection();

//Or make obj with reflection

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

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

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

//BindingFlags - choose operation: make obj, call method etc

Console.WriteLine("Plus(3,2)=" + Result);

}

}

}

**Результаты работы**



