|  |  |  |
| --- | --- | --- |
| Защищено:  Ю.Е. Гапанюк    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |  | Демонстрация:  Ю.Е. Гапанюк    "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

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

**Базовые компоненты интернет-технологий**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент гр. ИУ5Ц-52Б  (учится с гр.ИУ5-32Б) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Железов А.С. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

**Москва, МГТУ - 2017**

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

# Задание к Лабораторной роботе № 6:

**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.

**Текст программы:**

**Часть 1:**

using System;

using System.Collections.Generic;

namespace ConsoleApp1

{

class Program

{

public delegate int d1(int x1, string x2);

public static List<int> SumArrays(List<int> x, List<string> str, d1 func)

{

if (x.Count != str.Count)

{

List<int> t = new List<int>(-1);

return t;

}

else

{

List<int> t = new List<int>();

for (int i = 0; i < x.Count; i++)

{

t.Add(func(x[i], str[i]));

}

return t;

}

}

public static List<int> SumArraysFunc(List<int> x, List<string> str, Func<int, string, int> func)

{

if (x.Count != str.Count)

{

List<int> t = new List<int>(-1);

return t;

}

else

{

List<int> t = new List<int>();

for (int i = 0; i < x.Count; i++)

{

t.Add(func(x[i], str[i]));

}

return t;

}

}

public static int Sum(int x, string str)

{

int result = Int32.Parse(str);

return x + result;

}

static void Main(string[] args)

{

List<int> x = new List<int>();

x.Add(1);

x.Add(2);

x.Add(3);

List<string> str = new List<string>();

str.Add("2");

str.Add("3");

str.Add("4");

d1 sum = Sum;

List<int> l1 = SumArrays(x, str, sum);

foreach(var i in l1)

{

Console.WriteLine(i);

}

Console.WriteLine();

List<int> l2 = SumArrays(x, str,

(x\_1, str\_1) => x\_1 \* Int32.Parse(str\_1));

foreach (var i in l2)

{

Console.WriteLine(i);

}

Console.WriteLine();

Func<int, string, int> sum1 = Sum;

List<int> l3 = SumArraysFunc(x, str, sum1);

foreach (var i in l3)

{

Console.WriteLine(i);

}

Console.WriteLine();

Console.ReadKey();

}

}

}

**Часть 2:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp2

{

class Worker

{

[Attr("Имя")]

private int age { get; set; }

public string name { get; set; }

[Attr("Времяпровождение")]

public string timespending { get; set; }

public Worker(string name, int age)

{

this.age = age;

this.name = name;

}

public Worker(string name)

{

this.age = 30;

this.name = name;

}

public Worker(int age)

{

this.name = "Вася";

this.age = age;

}

public void Work(int type)

{

switch (type)

{

case 1:

Console.WriteLine("Сижу в офисе");

break;

case 2:

Console.WriteLine("Иду в бар");

break;

case 3:

Console.WriteLine("Получаю зарплату");

break;

default:

Console.WriteLine("Лечу на гаваи");

break;

}

}

}

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

class Attr : Attribute{

public string description { get; set; }

public Attr(string str\_description)

{

description = str\_description;

}

}

class Program

{

static void Main(string[] args)

{

Type t = typeof(Worker);

Console.WriteLine("Конструкторы:");

foreach(var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nАтрибуты свойств:");

foreach (var x in t.GetProperties())

{

var isAttr = x.GetCustomAttributes(typeof(Attr), false);

if (isAttr.Length > 0)

{

Console.WriteLine(isAttr[0]);

}

}

Worker Worker1 = new Worker(18);

object[] param = new object[] { 3 };

Console.WriteLine();

t.InvokeMember("Work", System.Reflection.BindingFlags.InvokeMethod, null, Worker1, param);

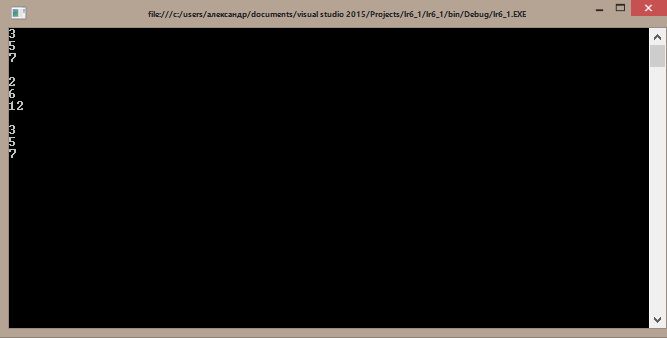
Console.ReadKey();

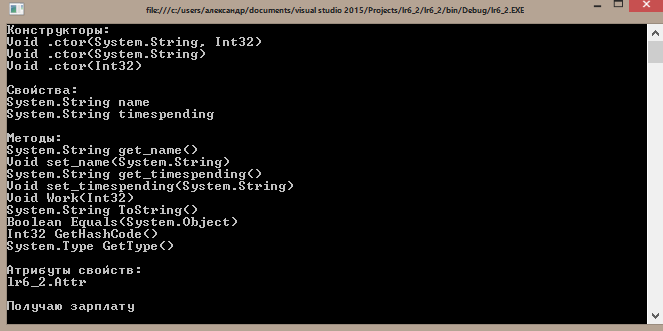
}

}

}

**Результаты выполнения программы:**

****

****

**Диаграмма классов:**

