**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33Б |  | преподаватель каф. ИУ5 |
| Кузнецов Григорий |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Описание задания**

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

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

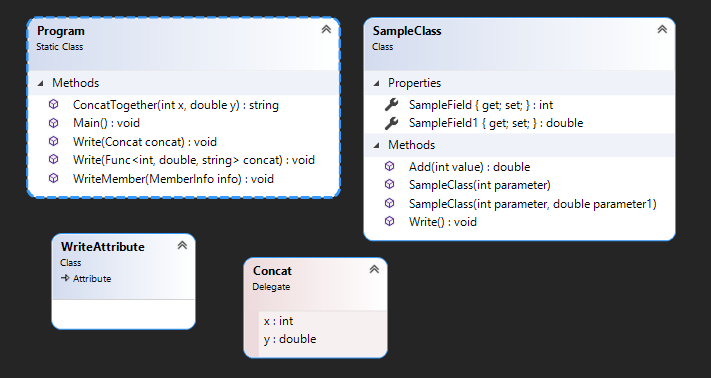
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

using System.Linq;

namespace Lab6

{

[AttributeUsage(AttributeTargets.Method | AttributeTargets.Property | AttributeTargets.Constructor)]

public class WriteAttribute : Attribute { }

public delegate string Concat(int x, double y);

public class SampleClass

{

[Write]

public int SampleField { get; set; }

public double SampleField1 { get; set; }

public SampleClass(int parameter, double parameter1) =>

(SampleField, SampleField1) = (parameter, parameter1);

[Write]

public SampleClass(int parameter) : this(parameter, parameter) { }

public void Write() =>

Console.WriteLine($"{SampleField}; {SampleField1}");

[Write]

public double Add(int value) => SampleField + SampleField1 + value;

}

public static class Program

{

public static string ConcatTogether(int x, double y) =>

$"{x}{y}";

public static void Write(Concat concat) =>

Console.WriteLine(concat(15, 3.45d));

public static void Write(Func<int, double, string> concat) =>

Console.WriteLine(concat(10, 2.71d));

public static void WriteMember(MemberInfo info)

{

if (info.GetCustomAttribute(typeof(WriteAttribute)) != null)

Console.WriteLine(info);

}

public static void Main()

{

Console.WriteLine(nameof(Concat));

Write(new Concat(ConcatTogether));

Write(new Concat((x, y) => $"{x}; {y}"));

Console.WriteLine(nameof(Func<int, double, string>));

Write(new Func<int, double, string>(ConcatTogether));

Write(new Func<int, double, string>((x, y) => $"{x} {y}"));

Console.WriteLine("Reflection");

var type = typeof(SampleClass);

foreach (var constructor in type.GetConstructors())

WriteMember(constructor);

foreach (var property in type.GetProperties())

WriteMember(property);

foreach (var method in type.GetMethods())

WriteMember(method);

var sampleClass = new SampleClass(4);

Console.WriteLine(typeof(SampleClass).GetMethod("Add").Invoke(sampleClass, new object[] { 3 }));

}

}

}

**Пример выполнения программы**

