**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»

отчёт

по лабораторной работе №6.

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы ИУ5-32 |  | преподаватель каф. ИУ5 |
| Федосеева Е.Ю. |  | Гапанюк Ю. Е. |
|  |  |  |

Москва, 2020 г.

Описание задания

Часть 1. Разработать программу, использующую делегаты.

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

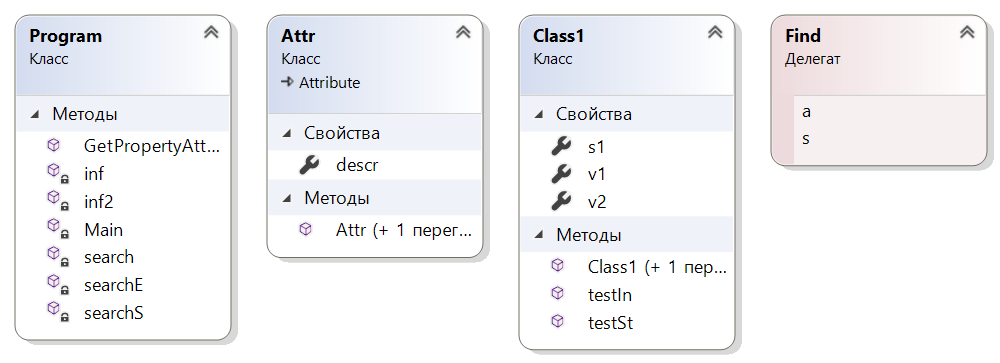
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

namespace Lab6

{

class Program

{

#region Part1

public delegate double SomeDelegate(double a, int b);

public static double? SomeMethod1(SomeDelegate del, double a, int b) => del?.Invoke(a, b);

public static double? SomeMethod2(Func<double, int, double> func, double a, int b) => func?.Invoke(a, b);

public static void SomeMethod3(Action action) => action?.Invoke();

public static double Product(double a, int b)

{

Console.WriteLine(a \* b);

return a \* b;

}

#endregion

static void Main()

{

#region Part1

var c1 = SomeMethod1(Product, 5, 5);

var c2 = SomeMethod1((a, b) =>

{

Console.WriteLine(a + b);

return a + b;

}, 5, 5);

var d1 = SomeMethod2(Product, 5, 5);

var d2 = SomeMethod2((a, b) =>

{

Console.WriteLine(a + b);

return a + b;

}, 5, 5);

SomeMethod3(() => Product(5, 5));

SomeMethod3(() =>

{

Console.WriteLine(5 + 5);

});

#endregion

#region Part2

var someClass = new SomeClass(11, "some text");

Console.WriteLine("\n" +

$"{someClass.ConstructorInfo()} \n" +

$"{someClass.MethodInfo()} \n" +

$"{someClass.PropertyInfo()} \n");

Console.WriteLine($"Свойства класса {nameof(someClass)} с атрибутом {nameof(UInfoAttribute)}:\n" +

$"{someClass.OutputProperties()}");

someClass.СallSomeMethod();

#endregion

Console.ReadKey();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

namespace Lab6

{

public static class ReflectInfoExt

{

public static string MethodInfo<T>(this T obj) where T : class

{

var result = new StringBuilder();

result.Append($"\*\*\* Список методов класса {obj} \*\*\*\n");

foreach (var m in typeof(T).GetMethods())

{

result.Append(" --> " + m.ReturnType.Name + " \t" + m.Name + "(");

var p = m.GetParameters();

for (var i = 0; i < p.Length; i++)

{

result.Append(p[i].ParameterType.Name + " " + p[i].Name);

if (i + 1 < p.Length) result.Append(", ");

}

result.Append(")\n");

}

return result.ToString();

}

public static string ConstructorInfo<T>(this T obj) where T : class

{

var result = new StringBuilder();

result.Append($"\*\*\* Список конструкторов класса {obj} \*\*\*\n");

foreach (var c in typeof(T).GetConstructors())

{

result.Append("--> " + c.Name + "(");

var p = c.GetParameters();

for (var i = 0; i < p.Length; i++)

{

result.Append(p[i].ParameterType.Name + " " + p[i].Name);

if (i + 1 < p.Length) result.Append(", ");

}

result.Append(")\n");

}

return result.ToString();

}

public static string PropertyInfo<T>(this T obj) where T : class

{

var result = new StringBuilder();

result.Append($"\*\*\* Список свойств класса {obj} \*\*\*\n");

foreach (var prop in typeof(T).GetProperties())

{

result.Append($"--> {prop.PropertyType.Name} \t{prop.Name}\n");

}

return result.ToString();

}

public static string OutputProperties<T>(this T obj) where T : class

{

var result = new StringBuilder();

foreach (var prop in typeof(T).GetProperties())

{

if (((UInfoAttribute[])prop.GetCustomAttributes(typeof(UInfoAttribute), false)).Length != 0)

{

result.Append($"--> {prop.PropertyType.Name} \t{prop.Name}\n");

}

}

return result.ToString();

}

public static void СallSomeMethod<T>(this T obj) where T : class

{

var method = obj.GetType().GetMethod("SomeMethod");

method?.Invoke(obj, null);

}

}

}

using System;

namespace Lab6

{

#region Attribute

[AttributeUsage(AttributeTargets.Property)]

public sealed class UInfoAttribute : Attribute

{

public string Desc;

public UInfoAttribute() { }

public UInfoAttribute(string str)

{

Desc = str;

}

}

#endregion

public class SomeClass

{

public int SomeProperty1 { get; }

[UInfo(Desc = "Некоторое свойство")]

public string SomeProperty2 { get; set; }

public SomeClass()

{

}

public SomeClass(int a)

{

SomeProperty1 = a;

}

public SomeClass(string b)

{

SomeProperty2 = b;

}

public SomeClass(int a, string b)

{

SomeProperty1 = a;

SomeProperty2 = b;

}

public void SomeMethod()

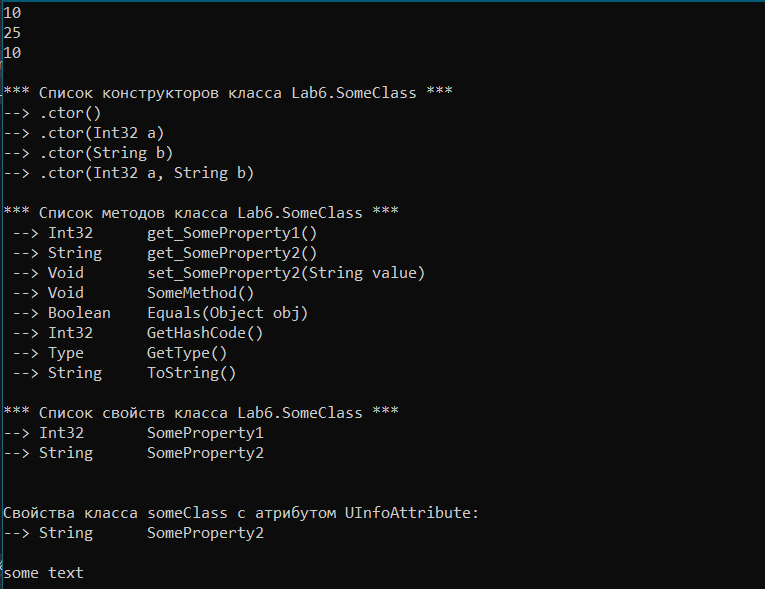
{

Console.WriteLine(SomeProperty2 ?? "Некоторый текст");

}

}

}

Анализ результатов