**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №6

«Трек курса «Функциональное программирование»»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35 |  | преподаватель каф. ИУ5 |
| Воронцова А.В. |  | Ю.Е. Гапанюк. |
| 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.

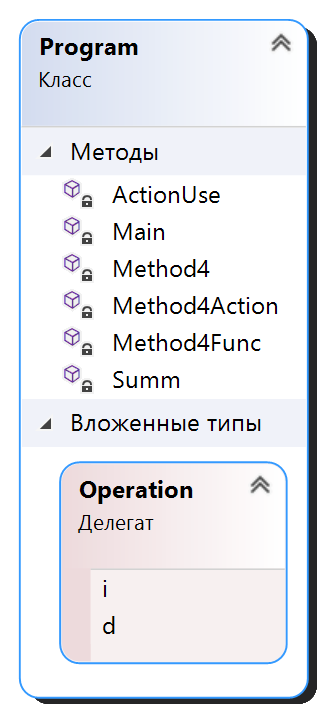
**Часть 2. Разработать программу, реализующую работу с рефлексией.**

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

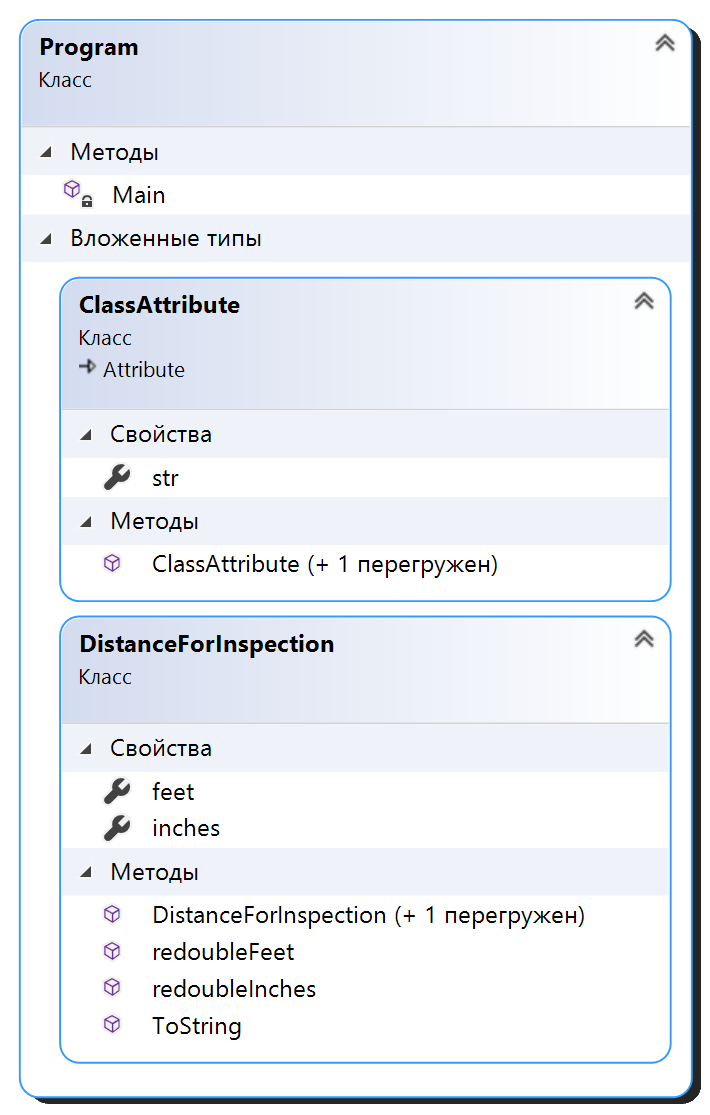
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
6. Вызовите один из методов класса с использованием рефлексии.

**Диаграмма классов**

**Для файла Program.cs**

****

**Для файла Program1.cs**

****

**Текст программы**

* **Для файла Program.cs – работа с делегатами**

using System;

namespace Lab.\_5

{

class Program

{

delegate object Operation(int i, double d);

/// <summary>

/// Функция суммирования. Преобразует double в int и возвращает сумму двух

/// </summary>

/// <param int-переменная="i"></param>

/// <param double-переменная="d"></param>

/// <returns></returns>

static object Summ(int i, double d)

{

object result = i + (int)d;

return result;

}

/// <summary>

/// Метод, использующий делегат

/// </summary>

static object Method4(int i, double d, Operation op)

{

object result = op(i, d);

return result;

}

/// <summary>

/// Метод для использования Action

/// </summary>

static void ActionUse(int i, double d)

{

double result = (double)i + d;

Console.WriteLine(result.ToString());

}

/// <summary>

/// Метод, использующий Func

/// </summary>

static object Method4Func(int i, double d, Func<int, double, object> func)

{

object result = func(i, d);

return result;

}

/// <summary>

/// Метод, использующий Action

/// </summary>

static void Method4Action(int i, double d, Action<int, double> act)

{

act(i, d);

}

static void Main(string[] args)

{

Console.WriteLine("Работа с делегатами");

Console.WriteLine("Обычный делегат");

object Result = Method4(4, 5.4, Summ);

Console.WriteLine(Result.ToString());

Console.WriteLine("Использование лямбда-функции");

Result = Method4

(

8,

7.89,

(int i, double d) =>

{

object result = (double)i + d;

return result;

}

);

Console.WriteLine(Result.ToString());

Console.WriteLine("Использование обобщённого делегата Func");

Result = Method4Func(9, 5.44, (x, y) => x + (int)y);

Console.WriteLine(Result.ToString());

Console.WriteLine("Использование обобщённого делегата Action");

Method4Action(77, 88.785, ActionUse);

Console.ReadKey();

}

}

}

**Для файла Program1.cs – работа с рефлексией**

using System;

using System.Reflection;

namespace Lab.\_51

{

class Program

{

/// <summary>

/// Класс - атрибут

/// </summary>

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

public class ClassAttribute : Attribute

{

public ClassAttribute() { }

public ClassAttribute(string s)

{

str = s;

}

public string str { get; set; }

}

/// <summary>

/// Класс для проверки

/// </summary>

class DistanceForInspection

{

[ClassAttribute("Футы")]

public int feet { get; set; }

public double inches { get; set; }

public DistanceForInspection() { feet = 0; inches = 0; }

public DistanceForInspection(int i, double d) { feet = i; inches = d; }

public int redoubleFeet(int f)

{

this.feet = f;

return this.feet \*= 2;

}

public double redoubleInches() => this.inches \*= 2;

public override string ToString()

{

return "Футы: " + feet + "дюймы: " + inches;

}

}

static void Main(string[] args)

{

Type t = typeof(DistanceForInspection);

Console.WriteLine("Конструкторы:");

foreach (var i in t.GetConstructors())

{

Console.WriteLine(i);

}

Console.WriteLine("\nСвойства");

foreach (var i in t.GetProperties())

{

Console.WriteLine(i);

}

Console.WriteLine("\nМетоды");

foreach (var i in t.GetMethods())

{

Console.WriteLine(i);

}

Console.WriteLine("\nСвойства с атрибутом");

foreach (var i in t.GetProperties())

{

//true для поиска атрибутов в цепочке наследования этого элемента, иначе false

var PropertyWithAttributes = i.GetCustomAttributes(typeof(ClassAttribute), false);

if (PropertyWithAttributes.Length > 0)

{

Console.WriteLine(i);

}

}

Console.WriteLine("\nВызов метода через рефлексию:");

DistanceForInspection dfi = (DistanceForInspection)t.InvokeMember(

null, BindingFlags.CreateInstance, null, null, new object[] { });

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

object Result = t.InvokeMember("redoubleFeet", BindingFlags.InvokeMethod, null, dfi,

parameters);

Console.WriteLine("Метод, удваивающий футы: " + Result);

Console.ReadKey();

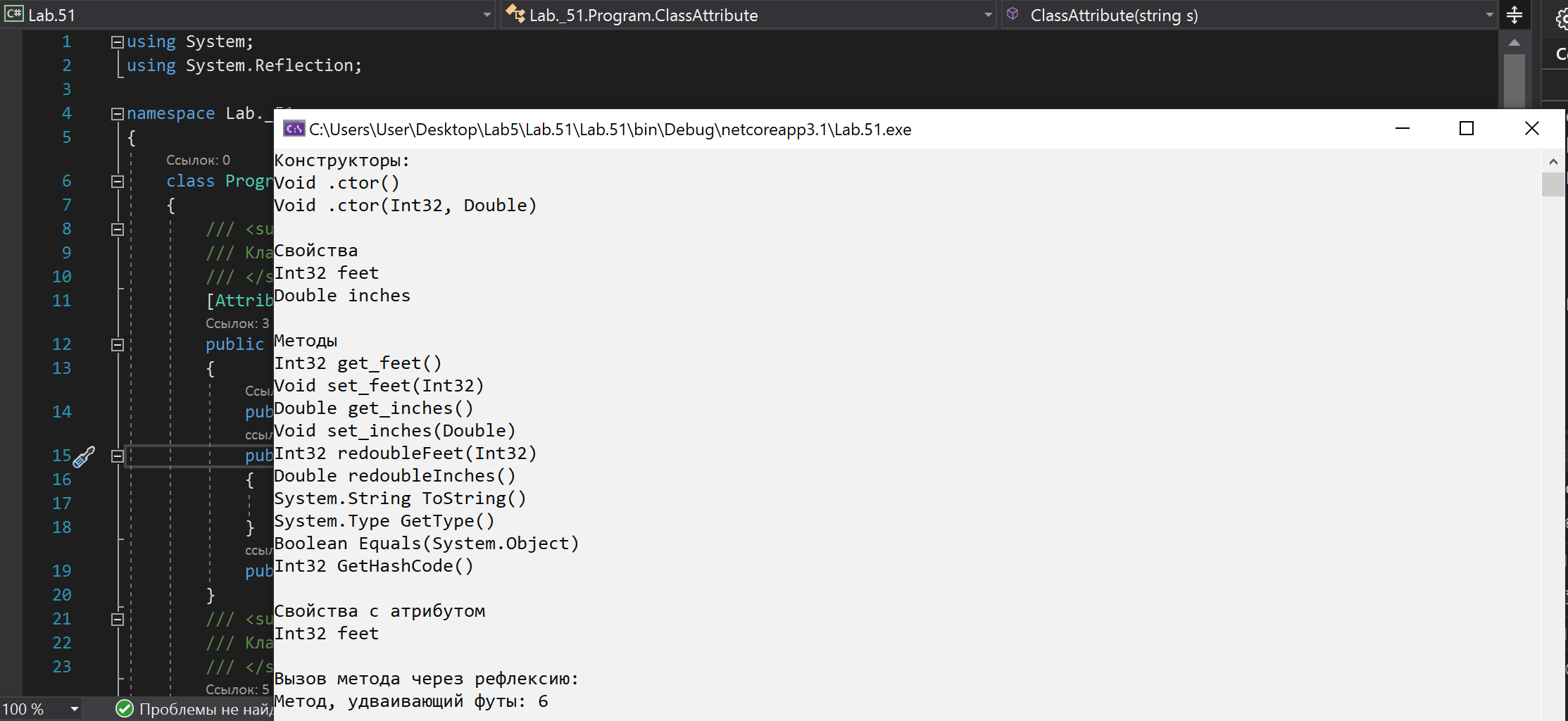
}

}

}

**Принтскрины экранных форм выполнения программ**

1. Работа с рефлексией (часть 2)



1. Работа с делегатами (часть 1)

