**Московский государственный технический**

**университет им. Н. Э. Баумана**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

«Работа с делегатами и рефлексией»

Группа: ИУ5-32Б

Студент: Селедкина А. С.

Преподаватель: Гапанюк Ю. Е.

Москва, 2018 г.

**Описание задания**

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

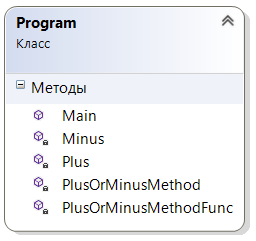
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

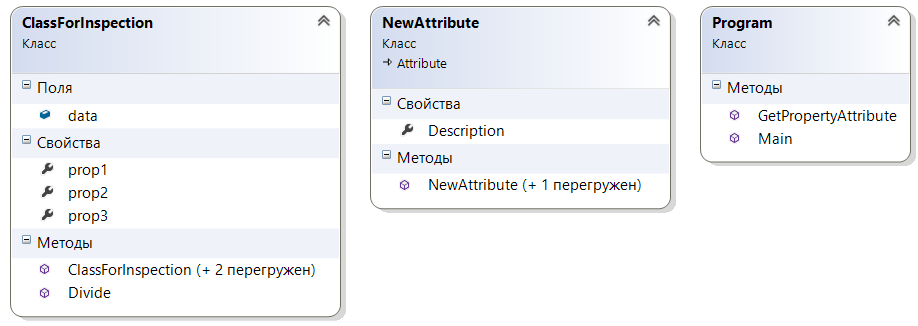
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5.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

**Диаграмма классов**

**Часть 1**



**Часть 2**



**Текст программы**

**Часть 1**

Файл **Program.cs**

using System;

using System.Security.Cryptography.X509Certificates;

namespace Lab6\_1

{

delegate int PlusOrMinus(int p1, int p2);

class Program

{

static int Plus(int p1, int p2)

{

return p1 + p2;

}

static int Minus(int p1, int p2)

{

return p1 - p2;

}

static void PlusOrMinusMethod(

string str,

int i1,

int i2,

PlusOrMinus PlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

}

static void PlusOrMinusMethodFunc(

string str,

int i1,

int i2,

Func<int, int, int> PlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

}

public static void Main(string[] args)

{

int i1 = 3;

int i2 = 2;

Console.WriteLine("Передача в качестве параметра разработанного делегата:");

PlusOrMinusMethod("Плюс: ", i1, i2, Plus);

PlusOrMinusMethod("Минус: ", i1, i2, Minus);

Console.WriteLine("\nПередача в качестве параметра лямбда-выражения:");

PlusOrMinusMethod(

"Плюс: ",

i1,

i2,

(int x, int y) =>

{

int z = x + y;

return z;

}

);

PlusOrMinusMethod(

"Минус: ",

i1,

i2,

(int x, int y) =>

{

int z = x - y;

return z;

}

);

Console.WriteLine("\nИспользование обобщенного делегата Func< >:");

PlusOrMinusMethodFunc("Плюс: ", i1, i2, Plus);

PlusOrMinusMethodFunc("Минус: ", i1, i2, Minus);

Console.ReadLine();

}

}

}

**Часть 2**

Файл **Program.cs**

using System;

using System.Reflection;

namespace Lab6\_2

{

internal class Program

{

public static bool GetPropertyAttribute(PropertyInfo checkType,

Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

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

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

public static void Main(string[] args)

{

Type t = typeof(ClassForInspection);

Console.WriteLine("\nИнформация о типе:");

Console.WriteLine("Тип " + t.FullName + " унаследован от " + t.BaseType.FullName);

Console.WriteLine("Пространство имен " + t.Namespace);

Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("\nПоля данных (public):");

foreach (var x in t.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства, помеченные атрибутом:");

foreach (var x in t.GetProperties())

{

object attrObj;

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

{

NewAttribute attr = attrObj as NewAttribute;

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

}

}

Console.WriteLine("\nВызов метода с помощью рефлексии:");

ClassForInspection cfi = new ClassForInspection();

object[] parameters = new object[] { 3.0, 2.0 };

object result = t.InvokeMember("Divide", BindingFlags.InvokeMethod, null, cfi, parameters);

Console.WriteLine("Divide(3.0, 2.0) = {0}", result);

Console.ReadLine();

}

}

}

Файл **ClassForInspection.cs**

namespace Lab6\_2

{

public class ClassForInspection

{

public ClassForInspection() { }

public ClassForInspection(string str) { }

public ClassForInspection(double d) { }

public double Divide(double a, double b)

{

return a / b;

}

[NewAttribute("Описание для prop1")]

public int prop1 { get; set; }

[NewAttribute("Описание для prop2")]

public int prop2 { get; set; }

public int prop3 { get; set; }

public int[] data;

}

}

Файл **NewAttribute.cs**

using System;

namespace Lab6\_2

{

[AttributeUsage(AttributeTargets.Property,

AllowMultiple = false,

Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

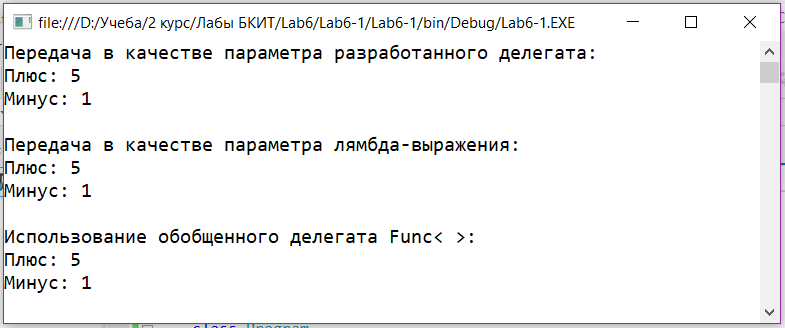
public string Description { get; set; }

}

}

**Примеры выполнения программы**

**Часть 1**



**Часть 2**

