Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Московский государственный технический университет имени Н.Э.Баумана

(МГТУ им. Н.Э.Баумана)

**Отчет по лабораторной работе №6**

**по курсу БКИТ**

Выполнил: Труфанов Вадим, ИУ 5-33

Преподаватель: Гапанюк Ю.Е.

Москва, 2018

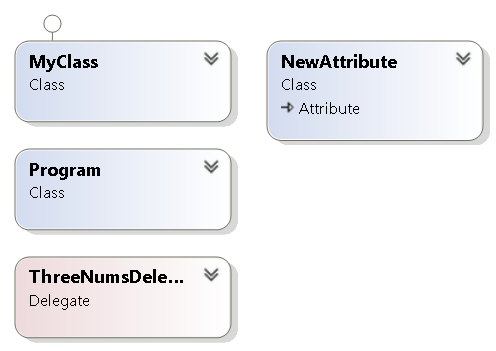
1. **Описание задания**

**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.
7. **Диаграмма классов**



1. **Текст программы**

*Класс Program*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_6

{

delegate int ThreeNumsDelegate(int num1, int num2, int num3);

class Program

{

static void Main(string[] args)

{

Console.WriteLine("------Work with Reflection-------");

int num1 = 7, num2 = 4, num3 = 15;

Console.WriteLine("Values of numbers: {0}, {1}, {2}", num1, num2, num3);

Console.WriteLine("Using method without Func< >");

//Передача метода-делегата

ThreeNumsMethod("Max value with delegate: ", num1, num2, num3, Max);

//Передача лямюда-выражения

ThreeNumsMethod("Max value with lambda: ", num1, num2, num3,

(int x, int y, int z) =>

{

return Math.Max(Math.Max(x, y), z);

}

);

Console.WriteLine("Using method with Func< >");

//Передача метода-делегата

ThreeNumsMethod("Min value with delegate: ", num1, num2, num3, Min);

//Передача лямюда-выражения

ThreeNumsMethod("Min value with lambda: ", num1, num2, num3,

(int x, int y, int z) =>

{

return Math.Min(Math.Min(x, y), z);

}

);

Console.WriteLine("\n\n------Work with Reflection-------");

TypeInfo();

AttributeInfo();

InvokeMemberInfo();

Console.ReadLine();

}

static int Min(int p1, int p2, int p3)

{

return Math.Min(Math.Min(p1, p2), p3);

}

static int Max(int p1, int p2, int p3)

{

return Math.Max(Math.Max(p1, p2), p3);

}

static void ThreeNumsMethod(string str, int i1, int i2, int i3, Func<int, int, int, int> threeNumsParam)

{

int Result = threeNumsParam(i1, i2, i3);

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

}

static void TypeInfo()

{

MyClass obj = new MyClass();

Type t = obj.GetType();

Console.WriteLine("\nКонструкторы:");

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("\nПоля данных (public):");

foreach (var x in t.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine("\nForInspection реализует IComparable -> " +

t.GetInterfaces().Contains(typeof(IComparable))

);

}

static void AttributeInfo()

{

Type t = typeof(MyClass);

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);

}

}

}

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

//Поиск атрибутов с заданным типом

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

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void InvokeMemberInfo()

{

Type t = typeof(MyClass);

Console.WriteLine("\nВызов метода:");

//MyClass fi = new MyClass();

MyClass fi = (MyClass) t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] {"MyStringggg", 999 });

//Параметры вызова метода

object[] parameters = new object[] { 678, fi.MyProperty };

//Вызов метода

object Result = t.InvokeMember("MyMethod", BindingFlags.InvokeMethod, null, fi, parameters);

Console.WriteLine("Max = {0}", Result);

}

}

}

*Класс MyClass*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_6

{

public class MyClass : IComparable

{

public string myField = string.Empty;

public MyClass()

{

}

public MyClass(string str, int num) {

MyProperty = num;

MyString = str;

}

public MyClass(string str)

{

myField = str;

}

public MyClass(int num) {

MyAutoImplementedProperty = num;

}

public int MyMethod(int parameter1, int parameter2)

{

Console.WriteLine("First Parameter {0}, second parameter {1}", parameter1, parameter2);

return Math.Max(parameter1, parameter2);

}

public void PrintNumber() {

Console.WriteLine(MyProperty);

}

public void PrintString() {

Console.WriteLine(MyString);

}

[NewAttribute()]

public int MyAutoImplementedProperty { get; set; }

private int myPropertyVar;

[NewAttribute(Description = "This is my and only my Property")]

public int MyProperty

{

get { return myPropertyVar; }

set { myPropertyVar = value; }

}

private string myStringVar;

[NewAttribute("This is my and only my String")]

public string MyString

{

get { return myStringVar; }

set { myStringVar = value; }

}

public int CompareTo(object obj)

{

return 0;

}

}

}

*Класс NewAttribute*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_6

{

//только для свойств класса, нельзя применять несколько NewAttribute,

//не наследуется от класса с атрибутами

[AttributeUsage(AttributeTargets.Property,

AllowMultiple = false,

Inherited = false)]

class NewAttribute : Attribute

{

public NewAttribute() {

AutoImplement();

}

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

private void AutoImplement() {

Description = "Property is auto implemented";

}

public string Description { get; set; }

}

}

1. **Экранные формы с примерами выполнения программы**

