**Московский Государственный Технический Университет имени Н. Э. Баумана**

**Базовые компоненты интернет-технологий**

**Лабораторная работа №6**

ИСПОЛНИТЕЛЬ:

ФИО Болотов Н.А.

Группа РТ5-31

ПРЕПОДАВАТЕЛЬ:

ФИО Гапанюк Ю. Е.

**Москва 2017**

**Программа лабораторной работы:**

**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.

**Код программы:**

*Делегаты:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegates

{

class Program

{

delegate float PowOrDiv(float x1, float x2);

static float Pow(float i, float j)

{

return i \* j;

}

static float Div(float i, float j)

{

return i / j;

}

static void Res(string str, float i, float j, PowOrDiv PowOrDivParam)

{

float N = PowOrDivParam(i, j);

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

}

static void Main(string[] args)

{

Console.WriteLine("x = 666");

Console.WriteLine("y = 228");

float i = 666, j = 228;

string str1 = "Умножение: ";

Res(str1, i, j, (x, y) => { return x \* y; });

string str2 = "Деление: ";

Res(str2, i, j, (x, y) => { return x / y; });

Console.WriteLine("-----------------------------------");

Console.WriteLine("Использование обобщённого делегата Action<>:");

Action<float, float> a1 = (x, y) => {

Console.WriteLine("{0} \* {1} = {2}", x, y, x \* y);

};

Action<float, float> a2 = (x, y) => {

Console.WriteLine("{0} / {1} = {2}", x, y, x / y);

};

Action<float, float> group = a1 + a2;

group(i, j);

Console.ReadKey();

}

}

}

*Рефлексия:*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Reflection

{

class Program

{

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

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

public class ForInspection

{

public ForInspection() { }

public ForInspection(int i) { }

public ForInspection(string str) { }

public int Plus(int x, int y) { return x + y; }

public int Minus(int x, int y) { return x - y; }

[NewAttribute("Описание для property1")]

public string property1 { get { return \_property1; } set { \_property1 = value; } }

private string \_property1;

public int property2 { get; set; }

[NewAttribute(Description = "Описание для property3")]

public double property3 { get; private set; }

public int field1;

public float field2;

}

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;

}

static void Main(string[] args)

{

Type t = typeof(ForInspection);

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:");

ForInspection fi = (ForInspection)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

object[] parameters = new object[] { 3, 2 };

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

Console.WriteLine("Plus(3,2)={0}", Result);

Console.ReadKey();

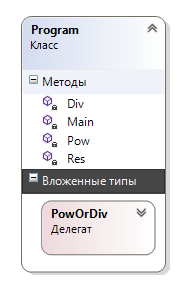
}

}

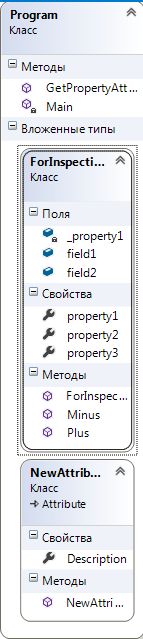
}

**Диаграмма классов:**

*Делегаты:*

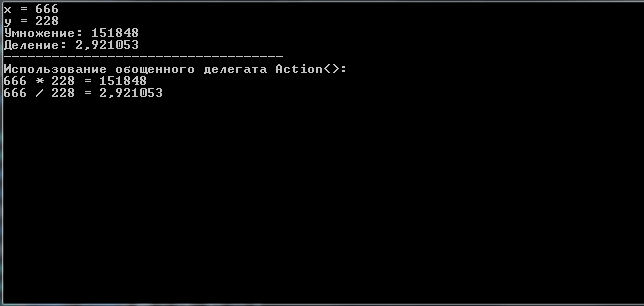


*Рефлексия:*



**Пример консольного вывода:**

*Делегаты:*



*Рефлексия:*

