Московский государственный технический университет имени Н.Э. Баумана (МГТУ им. Н.Э. Баумана)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

ПО ДИСЦИПЛИНЕ «БАЗОВЫЕ КОМПОНЕНТЫ ИНТЕРНЕТ ТЕХНОЛОГИЙ»

Выполнил(а): Белов Ф.А.

Проверил(а): Гапанюк Ю.Е.

Москва, 2017

**Формулировка задачи:**

**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**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace \_3sem\_lab6

{

class MainClass

{

delegate double Operation(int a, double b);

static double Plus (int a, double b)

{

return a + b;

}

static double Minus(int a, double b)

{

return a - b;

}

static void OperationMethod(string str, int a, double b, Operation OperationParam)

{

double result = OperationParam(a, b);

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

}

static void OperationGeneralizedMethod(string str, int a, double b, Func <int, double, double> OperationParam)

{

double result = OperationParam(a, b);

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

}

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

{

attribute = null;

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

if (isAttribute.Length > 0)

{

attribute = isAttribute[0];

return true;

}

return false;

}

public static void Main(string[] args)

{

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

Console.WriteLine("С помощью метода:");

OperationMethod("Сумма: ", 5, 2.1, Plus);

Console.WriteLine("С помощью лямбда-выражения:");

OperationMethod("Произведение: ", 5, 2.1, (a, b) => { return a \* b; });

Console.WriteLine("С использование обобщенного делегата:");

Console.WriteLine("С помощью метода:");

OperationGeneralizedMethod("Разность: ", 5, 2.1, Minus);

Console.WriteLine("С помощью лямбда-выражения:");

OperationGeneralizedMethod("Частное: ", 5, 2.1, (a, b) => { return a / b; });

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

User person = new User("Fedor", 19);

Type t = person.GetType();

Console.WriteLine("Конструкторы: ");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

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

Console.WriteLine("Свойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

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

Console.WriteLine("Методы:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Type type = typeof(User);

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

Console.WriteLine(("Свойства помеченные аттрибутом: "));

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(("Вызов метода с использованием рефлексии: "));

Console.WriteLine(("Cоздан объект Человек с параметрами {Fedor, 19}"));

object[] parameters = new object[] { "Fedor", 19 };

User user = (User)type.InvokeMember(null, BindingFlags.CreateInstance, null, null, parameters);

object result = type.InvokeMember("getBirthYear", BindingFlags.InvokeMethod, null, user, null);

Console.WriteLine("Год рождения: " + result.ToString());

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

}

}

[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 User

{

private string \_userName;

private int \_age;

[NewAttribute("Username пользователя")]

public string userName

{

get { return \_userName; }

private set { \_userName = value; }

}

public int age

{

get { return \_age; }

private set { \_age = value; }

}

public User(string name, int number) {

userName = name + age.ToString();

age = number;

}

public int getBirthYear()

{

return 2017 - age;

}

}

}