Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

****

Факультет ИУ

Кафедра ИУ5

**Отчет по лабораторной работе № 6**

Студент Гаджиев Кирилл Кириллович

Группа ИУ5-32

Название предмета БКИТ

Руководитель

Гапанюк Ю.Е. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата подпись

Студент

Гаджиев К.К. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата подпись

Задание

**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.

Листинг программы

using System;  
  
  
namespace Lab\_06  
{  
   
 public class MyAttribute : System.Attribute  
 {  
 public string Name { get; set; }  
  
 public MyAttribute(string name)  
 {  
 Name = name;  
 }  
 }  
  
 class Tree\_calculator  
 {  
 private double x;  
 private string creator;  
  
 public double Value  
 {  
 get { return x; }  
 }  
  
 [MyAttribute("some")]  
 public string Creator  
 {  
 get { return creator; }  
 }  
  
 [MyAttribute("ForDelegates7576")]  
 public calculate\_all foo;  
  
 [MyAttribute("ForDelegateso98767")]  
 public delegate double calculate\_all(double y, string user\_name);  
  
 [MyAttribute("ForDelegates74")]  
 public double Wrap(calculate\_all del, double y, string user\_name)  
 {  
 double temp;  
 Console.WriteLine("Wrap\n{");  
 temp = del.Invoke(y, user\_name);  
 Console.WriteLine("}");  
 return temp;  
 }  
  
 [MyAttribute("ForDelegates12")]  
 public double WrapFunc(Func<double, string, double> del, double y, string user\_name)  
 {  
 double temp;  
 Console.WriteLine("WrapFunc\n{");  
 temp = del.Invoke(y, user\_name);  
 Console.WriteLine("}");  
 return temp;  
 }  
  
 public Tree\_calculator(double x, string user\_name)  
 {  
 Console.WriteLine("Created with {0} by {1}", x, user\_name);  
 creator = user\_name;  
 this.x = x;  
 foo = Addition;  
 foo += Division;  
 foo += Subtraction;  
 foo += Multiplication;  
 }  
  
 public void Edit(double y, string user\_name)  
 {  
 Console.WriteLine("Edited from {0} to {1} by {2}", this.x, y, user\_name);  
 this.x = y;  
 }  
  
 public double Multiplication(double y, string user\_name)  
 {  
 Console.WriteLine("x = {0} \* {1} = {2} by {3}", x, y, x \* y, user\_name);  
 return x \* y;  
 }  
  
 public double Addition(double y, string user\_name)  
 {  
 Console.WriteLine("x = {0} + {1} = {2} by {3}", this.x, y, this.x + y, user\_name);  
 return x + y;  
 }  
  
 public double Subtraction(double y, string user\_name)  
 {  
 Console.WriteLine("x = {0} - {1} = {2} by {3}", this.x, y, this.x - y, user\_name);  
 return x - y;  
 }  
  
 public double Division(double y, string user\_name)  
 {  
 if (y.Equals(0.0))  
 {  
 Console.WriteLine("x = {0} / {1} = ZeroDivison error by {2}", this.x, y, user\_name);  
 return x;  
 }  
 else  
 {  
 Console.WriteLine("x = {0} / {1} = {2} by {3}", this.x, y, this.x / y, user\_name);  
 return x / y;  
 }  
 }  
  
 public static bool GetAttributeNames(System.Reflection.MemberInfo check, Type attribute)  
 {  
 bool result = false;  
  
 var isAttribute = check.GetCustomAttributes(attribute, false);  
 if (isAttribute.Length > 0)  
 {  
 result = true;  
 }  
 return result;  
 }  
  
 public static void Main(string[] args)  
 {  
 Console.ForegroundColor = ConsoleColor.Magenta;  
 Console.WriteLine("Delegates");  
   
 Console.ForegroundColor = ConsoleColor.Green;  
 Console.Write("Enter your name:");  
 Console.ResetColor();  
 string name = "";  
 name = Console.ReadLine();  
  
 Tree\_calculator calc = new Tree\_calculator(10, name);  
 //calc.foo(0, name);  
 calc.Wrap(calc.foo, 15, name);  
 calc.Wrap(calc.Addition, 10, name);  
 Console.ForegroundColor = ConsoleColor.Green;  
 calc.Wrap((y, usr\_name) => { Console.Write("Lambda addition: "); return calc.Addition(y, usr\_name); }, 10, name);  
 calc.WrapFunc(calc.Addition, 15, name);  
  
 Console.ForegroundColor = ConsoleColor.Magenta;  
 Console.WriteLine("\nReflection\n");  
 Console.ForegroundColor = ConsoleColor.Yellow;  
   
 var instance = calc;  
 var type = instance.GetType();  
 var props = type.GetProperties();  
  
 Console.ForegroundColor = ConsoleColor.Blue;  
 Console.WriteLine("Properties:");  
 foreach (var prop in props)  
 {  
 Console.WriteLine("{0}: {1}", prop.Name, prop.GetValue(instance));  
 }  
  
 Console.ForegroundColor = ConsoleColor.Yellow;  
 Console.WriteLine("\nMembers:");  
 var members = type.GetMembers();  
 foreach (var member in members)  
 {  
 Console.WriteLine("{0}: {1}", member.Name, member.GetCustomAttributes(true));  
 }  
  
 Console.ForegroundColor = ConsoleColor.Cyan;  
 Console.WriteLine("\nWith custom attribute \"MyAttribute\" ");  
 var attrs = typeof(Tree\_calculator).GetMembers();  
 foreach (var attr in attrs)  
 {  
 if (GetAttributeNames(attr, typeof(MyAttribute)))  
 {  
 Console.WriteLine(attr.Name);  
 }  
 }  
  
 Console.ForegroundColor = ConsoleColor.Blue;  
 Console.BackgroundColor = ConsoleColor.DarkYellow;  
 object[] parametres = new object[] { 4, "Reflection"} ;  
 object result = type.InvokeMember("Addition", System.Reflection.BindingFlags.InvokeMethod, null, calc, parametres);  
 Console.ResetColor();  
 }  
 }  
}

Результаты работы программы

