Московский государственный технический университет имени Н. Э. Баумана.

Факультет “Радиотехнический”
Кафедра ИУ5 “Системы обработки информации и управления”

Отчёт к лабораторной работе №6 по курсу «Парадигмы и конструкции языков программирования»

Выполнил:
Студент группы РТ5-31Б
Цыгичко А.Н.
Подпись и дата:

Проверил:
Преподаватель кафедры ИУ5
Гапанюк Ю.Е.
Подпись и дата:

Москва,2024 г.

Часть 1:

Постановка задачи.

Часть 1. Разработать программу, использующую делегаты.

(В качестве примера можно использовать проект «Delegates»).

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.

Текст программы:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab6_Part1
{
    internal class Program
    {
        public delegate Object MyDelegate(string value1, int value2);

        static void Main(string[] args)
        {
            //Пункты 2, 3
            string MultiStr(string value1, int value2)
            {
                return string.Concat(Enumerable.Repeat(value1, value2));
            }
            MyDelegate myDelegate = new MyDelegate(MultiStr);
            Console.WriteLine(myDelegate("1", 2));



            //Пункт 4
```

```
        void GetMultiIntValue(MyDelegate newDelegate, string firstWord, string delegateWord, int
delegateNum)
        {
            Console.WriteLine(firstWord + " " + newDelegate(delegateWord,
delegateNum).ToString());
        }
        GetMultiIntValue(new MyDelegate(MultiStr), "World", "Hellow", 1);
        GetMultiIntValue((string source, int increment) => source + " " + increment.ToString(),
"Andew", "got", 5);




        //Пункт5
        Func<string, int, Object> multiStr = (s, i) => string.Concat(Enumerable.Repeat(s, i));
        void NewMethod(Func<string, int, Object> func, string firstWord, string delegateWord, int
delegateNum)
        {
            Console.WriteLine(firstWord + " " + func(delegateWord, delegateNum).ToString());
        }
        NewMethod(multiStr, "Hello", "World", 3);

        Console.ReadLine();
    }
  }
}
```

Результат:

```
11
World Hellow
Andew got 5
Hello WorldWorldWorld
```

Часть 2:

Постановка задачи.

Часть 2. Разработать программу, реализующую работу с рефлексией.

(В качестве примера можно использовать проект «Reflection»).

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.

Текст программы:

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.NetworkInformation;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;

namespace Lab6_Part2
{
    [AttributeUsage((AttributeTargets.Method))]
    public class OpenMethod : System.Attribute
    {

    }

    [AttributeUsage((AttributeTargets.Property))]
    public class OpenProperty : System.Attribute
    {

    }
```

```
    [AttributeUsage((AttributeTargets.Constructor))]
    public class OpenConstructor : System.Attribute
    {

    }
    public class ReflectClass
    {
        [OpenConstructor]
        private ReflectClass()
        {

        }

        public ReflectClass(string name)
        {
            Name = name;
        }

        [OpenProperty]
        private string Name { get; }
        public int count { get; set; }

        private int Sum(int a, int b)
        {
            return a + b;
        }

        [OpenMethod]
        public void PrintMessage(string message)
        {
            Console.WriteLine(message);
        }

        public static void MethodReflectInfo<T>(T obj) where T : class
        {
            Type type = typeof(T);

            #region Constructor info
            Console.WriteLine("Список конструкторов класса {0}\n", obj.ToString());
            foreach (ConstructorInfo ctor in type.GetConstructors(BindingFlags.Instance |
BindingFlags.NonPublic | BindingFlags.Public))
            {
                string modificator = "";

                if (ctor.IsPublic)
                    modificator += "public";
                else if (ctor.IsPrivate)
                    modificator += "private";
                else if (ctor.IsAssembly)
                    modificator += "internal";
                else if (ctor.IsFamily)
                    modificator += "protected";
                else if (ctor.IsFamilyAndAssembly)
                    modificator += "private protected";
                else if (ctor.IsFamilyOrAssembly)
                    modificator += "protected internal";
```

```
            Console.Write($"{modificator} {type.Name}(");

            ParameterInfo[] parameters = ctor.GetParameters();
            for (int i = 0; i < parameters.Length; i++)
            {
                var param = parameters[i];
                Console.Write($"{param.ParameterType.Name} {param.Name}");
                if (i < parameters.Length - 1) Console.Write(", ");
            }
            Console.WriteLine(")");
        }
        Console.WriteLine("\n");
        #endregion

        #region Methods info
        Console.WriteLine("Список методов класса {0}\n", obj.ToString());
        foreach(MethodInfo mth in type.GetMethods(BindingFlags.DeclaredOnly |
BindingFlags.Instance | BindingFlags.NonPublic
            | BindingFlags.Public | BindingFlags.Static).Where(m => !m.Name.StartsWith("get_") &&
!m.Name.StartsWith("set_")))
        {
            string modificator = "";

            if (mth.IsPublic)
                modificator += "public";
            else if (mth.IsPrivate)
                modificator += "private";
            else if (mth.IsFamily)
                modificator += "protected";

            if (mth.IsStatic)
                modificator += " static";
            else if (mth.IsAbstract)
                modificator += " abstract";
            else if (mth.IsVirtual)
                modificator += " virtual";

            Console.Write($"{modificator} {mth.ReturnType.Name} {mth.Name}(");

            ParameterInfo[] parameters = mth.GetParameters();
            for (int i = 0; i < parameters.Length; i++)
            {
                var param = parameters[i];
                Console.Write($"{param.ParameterType.Name} {param.Name}");
                if (i < parameters.Length - 1) Console.Write(", ");
            }
            Console.WriteLine(")");
        }
        Console.WriteLine("\n");
        #endregion

        #region Properties info
        Console.WriteLine("Список свойств класса {0}\n", obj.ToString());
        foreach (PropertyInfo prop in type.GetProperties(BindingFlags.Public |
BindingFlags.NonPublic | BindingFlags.Instance))
        {
            Console.Write($"{prop.Name} Can read: {prop.CanRead} Can write: {prop.CanWrite}");
```

```
            Console.Write("\n");
        }
        #endregion
    }

    public static void MethodReflectInfoToOpenAttributes<T>(T obj) where T : class
    {
        Type type = typeof(T);

        #region Constructor info
        Console.WriteLine("Список конструкторов класса {0}\n", obj.ToString());
        foreach (ConstructorInfo ctor in type.GetConstructors(BindingFlags.Instance |
BindingFlags.NonPublic
            | BindingFlags.Public).Where(p => p.GetCustomAttributes<OpenConstructor>().Any()))
        {
            string modificator = "";

            if (ctor.IsPublic)
                modificator += "public";
            else if (ctor.IsPrivate)
                modificator += "private";
            else if (ctor.IsAssembly)
                modificator += "internal";
            else if (ctor.IsFamily)
                modificator += "protected";
            else if (ctor.IsFamilyAndAssembly)
                modificator += "private protected";
            else if (ctor.IsFamilyOrAssembly)
                modificator += "protected internal";
            Console.Write($"{modificator} {type.Name}(");

            ParameterInfo[] parameters = ctor.GetParameters();
            for (int i = 0; i < parameters.Length; i++)
            {
                var param = parameters[i];
                Console.Write($"{param.ParameterType.Name} {param.Name}");
                if (i < parameters.Length - 1) Console.Write(", ");
            }
            Console.WriteLine(")");
        }
        Console.WriteLine("\n");
        #endregion

        #region Methods info
        Console.WriteLine("Список методов класса {0}\n", obj.ToString());
        foreach (MethodInfo mth in type.GetMethods(BindingFlags.DeclaredOnly |
BindingFlags.Instance | BindingFlags.NonPublic
            | BindingFlags.Public | BindingFlags.Static).Where(m => !m.Name.StartsWith("get_") &&
!m.Name.StartsWith("set_")
            && m.GetCustomAttributes<OpenMethod>().Any()))
        {
            string modificator = "";

            if (mth.IsPublic)
                modificator += "public";
            else if (mth.IsPrivate)
                modificator += "private";
```

```
            else if (mth.IsFamily)
                modificator += "protected";

            if (mth.IsStatic)
                modificator += " static";
            else if (mth.IsAbstract)
                modificator += " abstract";
            else if (mth.IsVirtual)
                modificator += " virtual";

            Console.Write($"{modificator} {mth.ReturnType.Name} {mth.Name}(");

            ParameterInfo[] parameters = mth.GetParameters();
            for (int i = 0; i < parameters.Length; i++)
            {
                var param = parameters[i];
                Console.Write($"{param.ParameterType.Name} {param.Name}");
                if (i < parameters.Length - 1) Console.Write(", ");
            }
            Console.WriteLine(")");
        }
        Console.WriteLine("\n");
        #endregion

        #region Properties info
        Console.WriteLine("Список свойств класса {0}\n", obj.ToString());
        foreach (PropertyInfo prop in type.GetProperties(BindingFlags.Public |
BindingFlags.NonPublic
            | BindingFlags.Instance).Where(p => p.GetCustomAttributes<OpenProperty>().Any()))
        {
            Console.Write($"{prop.Name} Can read: {prop.CanRead} Can write: {prop.CanWrite}");
            Console.Write("\n");
        }
        #endregion
    }
}

internal class Program
{
    static void Main(string[] args)
    {
        ReflectClass reflectClass = new ReflectClass("TestClass");

        ReflectClass.MethodReflectInfo<ReflectClass>(reflectClass);
        Console.WriteLine("\n\n\n");

        ReflectClass.MethodReflectInfoToOpenAttributes<ReflectClass>(reflectClass);
        Console.WriteLine("\n\n\n");

        Console.WriteLine("Результат вызова метода PrintMessage с помощью рефлексии:\n");
        MethodInfo classMethod = typeof(ReflectClass).GetMethod("PrintMessage");
        object classValue = classMethod.Invoke(reflectClass, new object[] { "Hellow World!" });

        Console.ReadLine();
    }
}
}
```

Результат:

```
Список конструкторов класса Lab6_Part2.ReflectClass

private ReflectClass()
public ReflectClass(String name)


Список методов класса Lab6_Part2.ReflectClass

private Int32 Sum(Int32 a, Int32 b)
public Void PrintMessage(String message)
public static Void MethodReflectInfo(T obj)
public static Void MethodReflectInfoToOpenAttributes(T obj)


Список свойств класса Lab6_Part2.ReflectClass

Name Can read: True Can write: False
count Can read: True Can write: True




Список конструкторов класса Lab6_Part2.ReflectClass

private ReflectClass()


Список методов класса Lab6_Part2.ReflectClass

public Void PrintMessage(String message)


Список свойств класса Lab6_Part2.ReflectClass

Name Can read: True Can write: False




Результат вызова метода PrintMessage с помощью рефлексии:

Hellow World!
```