



**Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)**

**Базовые компоненты интернет технологий  
Отчет по лабораторной работе №6**

Студент: Мурзин В.В.  
Группа: ИУ5Ц-51Б

Преподаватель: Гапанюк Ю. Е.

2019 г.

## **Лабораторная работа №6**

### **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.

### **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.

## Приложение

### DelegatesLab:

#### Program.cs:

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

namespace DelegatesLab
{
    class Program
    {
        delegate String Operation(String name, double number);
        private static void RegisterDelegate(Operation op, String name, double number)
        {
            Console.WriteLine(op(name, number));
        }
        private static void RegisterFunc(Func<String, double, String> func, String name,
double number)
        {
            Console.WriteLine(func(name, number));
        }
        static void Main(string[] args)
        {
            Operation op = SqrtExp;
            Console.WriteLine(op("sqrt", 34.4));
            RegisterDelegate(op, "exp", -9); // делегат как параметр
            RegisterDelegate((name, number) => name.Equals("sqrt")?
                "Корень из " + number + " = " + Math.Sqrt(number) :
                (name.Equals("exp") ? "е в степени " + number + " = " + Math.Exp(number)
: ""), "sqrt", 6.66); //лямбда-выражение как параметр
            Func<String, double, String> func = SqrtExp;
            RegisterFunc(func, "pow", 9.999); // использование Func<>

            Console.ReadKey();
        }
        private static String SqrtExp(String name, double number)
        {
            if (name.Equals("sqrt"))
            {
                return "Корень из " + number + " = " + Math.Sqrt(number);
            }
            else if(name.Equals("exp"))
            {
                return "е в степени " + number + " = " + Math.Exp(number);
            }
            else
            {
                return "Нет такой операции.";
            }
        }
    }
}
```

### ReflectionLab:

#### Program.cs:

```

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

namespace ReflectionLab
{
    class Program
    {
        public class User
        {
            public string Name { get; set; }
            [NumValidation(18)]
            public int Age { get; set; }
            [NumValidation(5)]
            public int ChildCount { get; set; }
            public User(string n, int a)
            {
                Name = n;
                Age = a;
            }
            public User()
            {
                Name = "Игорь";
                Age = 22;
            }
            public void Display()
            {
                Console.WriteLine("Имя: " + Name + ", Возраст:" + Age);
            }
            public int Payment(int hours, int perhour)
            {
                return hours * perhour;
            }
        }

        public static void FieldPropertiesConstructorsInfo<T>(T obj) where T : class
        {
            Type t = typeof(T);
            Console.WriteLine("\n*** Конструкторы ***\n");
            ConstructorInfo[] constructors = t.GetConstructors();
            foreach (ConstructorInfo info in constructors)
            {
                Console.WriteLine("--> Количество параметров: " +
info.GetParameters().Count());
                // Вывести параметры конструкторов
                ParameterInfo[] p = info.GetParameters();
                for (int i = 0; i < p.Length; i++)
                {
                    Console.Write(p[i].ParameterType.Name + " " + p[i].Name);
                    if (i + 1 < p.Length) Console.Write(", ");
                }
                Console.WriteLine();
            }
            Console.WriteLine("\n*** Поля ***\n");
            FieldInfo[] fieldNames = t.GetFields();
            foreach (FieldInfo fil in fieldNames)
            {
                Console.WriteLine("--> " + fil.FieldType + " " + fil.Name + "\n");
            }
            Console.WriteLine("\n*** Свойства ***\n");
            PropertyInfo[] propertyNames = t.GetProperties();
            foreach (PropertyInfo property in propertyNames)
            {
                Console.WriteLine("--> " + property.PropertyType + " " + property.Name +
"\n");
            }
        }
    }
}

```

```

// Данный метод выводит информацию о содержащихся в классе методах
public static void MethodReflectInfo<T>(T obj) where T : class
{
    Type t = typeof(T);
    // Получаем коллекцию методов
    MethodInfo[] MArr = t.GetMethods(BindingFlags.DeclaredOnly |
BindingFlags.Instance | BindingFlags.Public);
    Console.WriteLine("*** Список методов класса {0} ***\n", obj.ToString());

    // Вывести методы
    foreach (MethodInfo m in MArr)
    {
        Console.Write(" --> " + m.ReturnType.Name + " \t" + m.Name + "(");
        // Вывести параметры методов
        ParameterInfo[] p = m.GetParameters();
        for (int i = 0; i < p.Length; i++)
        {
            Console.Write(p[i].ParameterType.Name + " " + p[i].Name);
            if (i + 1 < p.Length) Console.Write(", ");
        }
        Console.Write(")\n");
    }
}

[AttributeUsage(AttributeTargets.Property)]
public class NumValidationAttribute : System.Attribute
{
    public int Age { get; set; }

    public NumValidationAttribute()
    { }

    public NumValidationAttribute(int age)
    {
        Age = age;
    }
}

static void Main(string[] args)
{
    User user = new User();
    MethodReflectInfo<User>(user);
    FieldPropertiesConstructorsInfo<User>(user);
    User oleg = new User("Олег", 35);
    Console.WriteLine();
    oleg.Display();
    ValidateUser(oleg);

    //----- рефлексия вызов метода
    Type t = typeof(User);
    MethodInfo methodInfo = t.GetMethod("Payment");
    object[] parametersArray = new object[] { 30, 400 };
    Console.WriteLine("Результат вызова метода Payment с параметрами 30 и 400 = "
+ methodInfo.Invoke(oleg, parametersArray));
    Console.ReadKey();
}

static void ValidateUser(User user)
{
    Type t = typeof(User);
    object[] p = t.GetProperties();
    Console.WriteLine("-----");
    Console.WriteLine("Значения атрибутов:");
    foreach (PropertyInfo i in p)
    {

```

```
        object[] attrs = i.GetCustomAttributes(false);  
        foreach (NumValidationAttribute attr in attrs)  
        {  
            Console.WriteLine(attr.Age);  
        }  
    }  
}  
}
```

## Ссылка на репозиторий исходных кодов GitHub

<https://github.com/SalyVo/Lab6>