

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)



Факультет Информатика и системы управления

Кафедра Системы обработки информации и управления (ИУ-5)

Отчет по лабораторной работе № 6

Студент Яковицкий Станислав Владиславович
(фамилия, имя, отчество)

Группа ИУ5-35

Название предмета Базовые компоненты интернет-технологий

Руководитель

Гапанюк Ю.Е.
ФИО

дата

подпись

Студент

Яковицкий С.В.
ФИО

дата

подпись

Задание:

Задание:

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.

Диаграмма классов:



Текст программы:

ReflectionObserver.cs

```
using System;
```

```
namespace lab06
```

```
{
```

```
    public class ReflectionObserver
```

```
    {
```

```
        public ReflectionObserver() { } public
```

```
        ReflectionObserver(int i) { } public
```

```
        ReflectionObserver(string str) { }
```

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

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

```
        [CustomAttribute("публик стринг проперти4 гет и сет")]
```

```
        public string property1 { get { return _property1; } set { _property1
```

```
= value; } }
```

```
        private string _property1;
```

```

[CustomAttribute("публик инт проперти2 гет и сет")]
public int property2 { get; set; }

public double property3 { get; set; }

[CustomAttribute("публик флоат проперти4 гет и сет")]
public float property4 { get; set; }

public short property5 { get; set; }

public int field1; public float field2;
}
}

```

CustomAttribute.cs

```

using System;

namespace lab06
{
    [AttributeUsage(AttributeTargets.Property,
        AllowMultiple = false, Inherited = false)
    ]
    public class CustomAttribute : Attribute
    {
        public CustomAttribute() { }
        public CustomAttribute(string mDescription)
        {
            Description = mDescription;
        }
        public string Description { get; set; }
    }
}

```

```
}  
}
```

Program.cs

```
using System;  
using System.Reflection;  
  
namespace lab06  
{  
  
    delegate int PlusOrMinus(int p1, int p2);  
  
    class Program  
    {  
  
        static int Plus(int p1, int p2)  
        {  
            return p1 + p2;  
        }  
        static int Minus(int p1, int p2)  
        {  
            return p1 - p2;  
        }  
        static void PlusOrMinusMethodFunc(string str, int i1, int  
i2, Func<int, int, int> PlusOrMinusParam)  
        {  
            int result = PlusOrMinusParam(i1, i2);  
            Console.WriteLine(str + result.ToString());  
        }  
        static void PlusOrMinusMethod(string str, int i1, int i2,  
PlusOrMinus PlusOrMinusParam)  
        {
```

```

        int result = PlusOrMinusParam(i1, i2);
        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;
    }
    static void Main(string[] args)
    {

        Console.WriteLine("ДЕЛЕГАТЫ\n");

        int i1 = 10, i2 = 5;
        PlusOrMinusMethod("Плюс ", i1, i2, Plus);
        PlusOrMinusMethod("Минус ", i1, i2, Minus);

        PlusOrMinus useMethod = new PlusOrMinus(Plus);
        PlusOrMinusMethod("Используем метод ", i1, i2, useMethod);

        PlusOrMinus useIdea = Plus;
        PlusOrMinusMethod("Используем \"предположение\"
компилятора ", i1, i2, useIdea);
    }
}

```

```
PlusOrMinus anonMethod = delegate (int param1, int param2)
{
    return param1 + param2;
};
PlusOrMinusMethod("Используем анонимный метод ", i1, i2,
anonMethod);
PlusOrMinusMethod("Используем лямбду 0 ", i1, i2,
(int x, int y) =>
{
    int z = x + y;
    return z;
});
PlusOrMinusMethod("Используем лямбду 1 ", i1, i2, (x, y) => {
return x + y; });
PlusOrMinusMethod("Используем лямбду 2 ", i1, i2, (x, y) => x
+ y);
```

```
Console.WriteLine("\nИспользуем делегат Func<>");
```

```
PlusOrMinusMethodFunc("Используем метод Plus ", i1, i2,
Plus);
```

```
string strOutside = "sample text";
```

```
PlusOrMinusMethodFunc("Используем лямбду 0\n", i1, i2,
(int x, int y) =>
{
    Console.WriteLine("\nПеременная вне лямбды " +
strOutside);
    int z = x + y;
    return z;
});
PlusOrMinusMethodFunc("\nИспользуем лямбду 2\n", i1, i2,
(x, y) =>
{
```



```

        return x + y;
    });
    PlusOrMinusMethodFunc("\nИспользуем лямбду 3\n", i1, i2,
(x, y) => x + y);

    Console.WriteLine("\nИспользуем групповой делегат");
    Action<int, int> a1 = (x, y) => { Console.WriteLine("{0} + {1} =
{2}", x, y, x + y); };
    Action<int, int> a2 = (x, y) => { Console.WriteLine("{0} - {1} =
{2}", x, y, x - y); };
    Action<int, int> group = a1 + a2;
    group(10, 5);

    Action<int, int> group2 = a1;
    Console.WriteLine("Добавление вызова метода к групповому
делегату");
    group2 += a2; group2(10, 5);
    Console.WriteLine("Удаление вызова метода из
группового делегата");
    if ((group2 != null) && (a1 != null))
    {
        group2 -= a1;
    }
    group2(20, 10);

    Console.WriteLine("\nРЕФЛЕКСИЯ\n");

    Type t = typeof(ReflectionObserver);

    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("\nПубличные методы");
foreach (var x in t.GetFields()) {

    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("\nСвойства с
атрибутами"); foreach (var x in
t.GetProperties()) {
    object attrObj;
    if (GetPropertyAttribute(x, typeof(CustomAttribute), out
attrObj))
    {
        CustomAttribute attr = attrObj as CustomAttribute;
        Console.WriteLine(x.Name + " - " + attr.Description);
    }
}
```

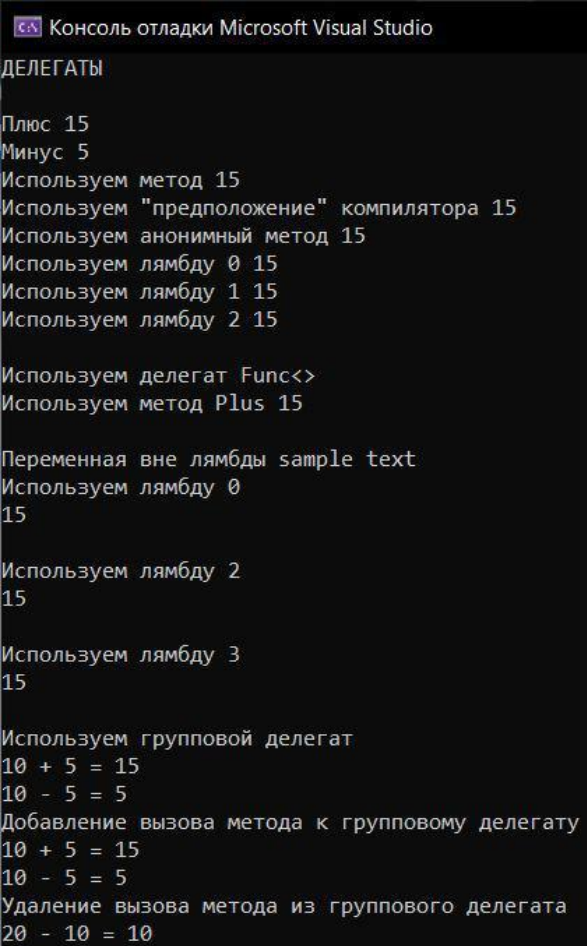
```

        Console.WriteLine("InvokeMember");
        ReflectionObserver fi =
(ReflectionObserver)t.InvokeMember(null, BindingFlags.CreateInstance,
null, null, new object[] { });
        Console.WriteLine("InvokeMethod");
        object[] parameters = { 10, 5 };
        object Result = t.InvokeMember("Plus",
BindingFlags.InvokeMethod, null, fi, parameters);
        Console.WriteLine("Plus(10,5)={0}", Result);

        Console.ReadLine();
    }
}
}

```

Экранные формы с примерами выполнения программы:



```

Консоль отладки Microsoft Visual Studio
ДЕЛЕГАТЫ
Плюс 15
Минус 5
Используем метод 15
Используем "предположение" компилятора 15
Используем анонимный метод 15
Используем лямбду 0 15
Используем лямбду 1 15
Используем лямбду 2 15
Используем делегат Func<>
Используем метод Plus 15
Переменная вне лямбды sample text
Используем лямбду 0
15
Используем лямбду 2
15
Используем лямбду 3
15
Используем групповой делегат
10 + 5 = 15
10 - 5 = 5
Добавление вызова метода к групповому делегату
10 + 5 = 15
10 - 5 = 5
Удаление вызова метода из группового делегата
20 - 10 = 10

```

```
Консоль отладки Microsoft Visual Studio
РЕФЛЕКСИЯ

Тип lab06.ReflectionObserver унаследован от System.Object
Пространство имен lab06
Находится в сборке lab06.ReflectionObserver, lab06, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

Конструкторы:
Void .ctor()
Void .ctor(Int32)
Void .ctor(System.String)

Публичные методы
Int32 field1
Single field2

Методы:
Int32 Plus(Int32, Int32)
Int32 Minus(Int32, Int32)
System.String get_property1()
Void set_property1(System.String)
Int32 get_property2()
Void set_property2(Int32)
Double get_property3()
Void set_property3(Double)
Single get_property4()
Void set_property4(Single)
Int16 get_property5()
Void set_property5(Int16)
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()

Свойства
System.String property1
Int32 property2
Double property3
Single property4
Int16 property5

Свойства с атрибутами
property1 - публик стринг проперти4 гет и сет
property2 - публик инт проперти2 гет и сет
property4 - публик флоат проперти4 гет и сет
InvokeMember
InvokeMethod
Plus(10,5)=15
```

Ссылка на репозиторий исходных кодов GitHub:

<https://github.com/YakovitskiySV/Labs>