**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б |  | преподаватель каф. ИУ5 |
| Незаметдинов Тимур |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

# Описание задания

**Часть 1. Разработать программу, использующую делегаты.**

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

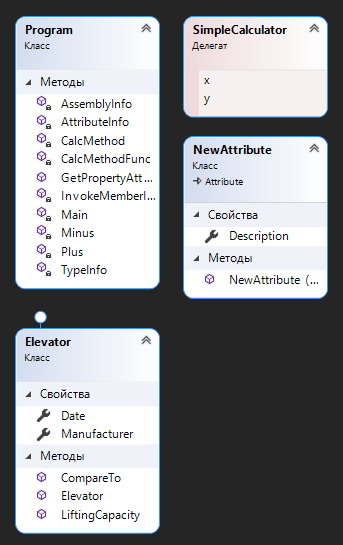
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, 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**

namespace WindowsFormsApplication1

{

|  |
| --- |
| using System; using System.Collections.Generic; using System.Linq; using System.Reflection; using System.Text; using System.Threading.Tasks;  namespace Reflection {          delegate string SimpleCalculator(double x, double y);      class Program     {          static string Plus(double x, double y)         {             return $"{x} + {y} равно" + " " + (x +y);         }          static string Minus(double x, double y)         {             return $"{x} - {y} равно" + " " + (x - y);         }          static void CalcMethod(double x, double y, SimpleCalculator parameter)         {             string result = parameter(x, y);             Console.WriteLine(result);         }          static void CalcMethodFunc(double x, double y, Func<double, double, string> parameter)         {             string result = parameter(x, y);             Console.WriteLine(result);         }           /// <summary>         /// Проверка, что у свойства есть атрибут заданного типа         /// </summary>         /// <returns>Значение атрибута</returns>         public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)         {             bool Result = false;             attribute = null;              //Поиск атрибутов с заданным типом             var isAttribute = checkType.GetCustomAttributes(attributeType, false);             if (isAttribute.Length > 0)             {                 Result = true;                 attribute = isAttribute[0];             }              return Result;         }          /// <summary>         /// Получение информации о текущей сборке         /// </summary>         static void AssemblyInfo()         {             Console.WriteLine("Вывод информации о сборке:");             Assembly i = Assembly.GetExecutingAssembly();             Console.WriteLine("Полное имя:" + i.FullName);             Console.WriteLine("Исполняемый файл:" + i.Location);         }          /// <summary>         /// Получение информации о типе         /// </summary>         static void TypeInfo()         {             Elevator obj = new Elevator();             Type t = obj.GetType();              //другой способ             //Type t = typeof(ForInspection);              Console.WriteLine("\nИнформация о типе:");             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("\nМетоды:");             foreach (var x in t.GetMethods())             {                 Console.WriteLine(x);             }              Console.WriteLine("\nСвойства:");             foreach (var x in t.GetProperties())             {                 Console.WriteLine(x);             }              Console.WriteLine("\nПоля данных (public):");             foreach (var x in t.GetFields())             {                 Console.WriteLine(x);             }              Console.WriteLine("\nElevator реализует IComparable -> " +             t.GetInterfaces().Contains(typeof(IComparable))             );         }          /// <summary>         /// Пример использования метода InvokeMember         /// </summary>         static void InvokeMemberInfo()         {             Type t = typeof(Elevator);             Console.WriteLine("\nВызов метода:");              //Создание объекта             //ForInspection fi = new ForInspection();             //Можно создать объект через рефлексию             Elevator fi = (Elevator)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });              //Параметры вызова метода             object[] parameters = new object[] { 6 };             //Вызов метода             object Result = t.InvokeMember("LiftingCapacity", BindingFlags.InvokeMethod, null, fi, parameters);             Console.WriteLine("LiftingCapacity(6)={0}", Result);         }          /// <summary>         /// Работа с атрибутами         /// </summary>         static void AttributeInfo()         {             Type t = typeof(Elevator);             Console.WriteLine("\nСвойства, помеченные атрибутом:");             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);                 }             }         }          static void Main(string[] args)         {             double x, y;             Console.WriteLine("Введите Х");             x = Convert.ToDouble(Console.ReadLine());             Console.WriteLine("Введите Y");             y = Convert.ToDouble(Console.ReadLine());             CalcMethod(x, y, Plus);             CalcMethod(x, y, Minus);             SimpleCalculator pluslambda = (double a, double b) =>             {                 return $"{x} + {y} равно" + " " + (x + y);             };             CalcMethod(x, y, pluslambda);             CalcMethodFunc(x, y, Minus);             AssemblyInfo();             TypeInfo();             InvokeMemberInfo();             AttributeInfo();              Console.ReadLine();         }     } } |

using System;  
  
namespace Reflection  
{  
    /// <summary>  
    /// Класс атрибута   
    /// </summary>  
    [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]  
    public class NewAttribute : Attribute  
    {  
        public NewAttribute() { }  
        public NewAttribute(string DescriptionParam)  
        {  
            Description = DescriptionParam;  
        }  
  
        public string Description { get; set; }  
    }  
}

using System;  
  
namespace Reflection  
{  
    /// <summary>  
    /// Класс для исследования с помощью рефлексии  
    /// </summary>  
    public class Elevator : IComparable  
    {  
        public Elevator() { }  
        //public Elevator(int i) { }  
        //public Elevator(string str) { }  
  
        public int LiftingCapacity(int capacity)  
        {  
            return capacity \* 80;  
        }  
  
        [NewAttribute("Год выпуска")]  
        public int Date  
        {  
            get { return Date; }  
            set { Date = value; }  
        }  
  
        [NewAttribute("Производитель")]  
        public string Manufacturer  
        {  
            get { return Manufacturer; }  
            set { Manufacturer = value; }  
        }  
  
       /// public int field1;  
       /// public float field2;  
  
        /// <summary>  
        /// Реализация интерфейса IComparable  
        /// </summary>  
        public int CompareTo(object obj)  
        {  
            return 0;  
        }  
    }  
}

# Результат работы программы

