Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6 «Вычисление расстояния Левенштейна»

Выполнил:
студент группы ИУ5-33
Сергеев МЮ
Подпись и дата:
29.12.20

Проверил:
Подпись и дата:

Москва, 2020 г.

Задание:

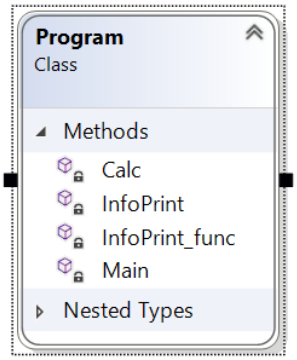
Часть 1. Разработать программу, использующую делегаты.

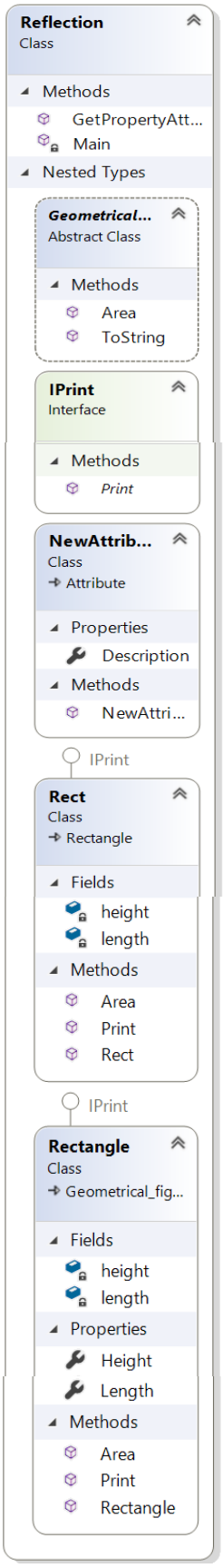
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

Диаграмма классов:




Reflection
Class
Methods
GetPropertyAtt...
Main
Nested Types
Geometrical...
Abstract Class
Methods
Area
ToString
IPrint
Interface
Methods
Print
NewAttrib...
Class
Attribute
Properties
Description
Methods
NewAttri...
IPrint
Rect
Class
Rectangle
Fields
height
length
Methods
Area
Print
Rect
IPrint
Rectangle
Class
Geometrical_fig...
Fields
height
length
Properties
Height
Length
Methods
Area
Print
Rectangle

Текст программы
Program.cs

```
using System;
namespace Lab6_1
{
   class Program
   {
      delegate double Acc(string billnum, int bill, double percent);
      static double Calc(string billnum, int bill, double percent)
      {
         Console.WriteLine("Account number: {0}", billnum);
         Console.Write("Account amount: ");
         return bill * (1 + percent);
      }

      static void InfoPrint(string name, string sur, string mid, string billnum, int
bill, double percentage, Acc billinfo)
      {
         Console.WriteLine("Client information");
         Console.WriteLine("Name {0}", name);
         Console.WriteLine("Surname {0}", sur);
         Console.WriteLine("Middle name {0}", mid);
         Console.WriteLine("{0}", billinfo(billnum, bill, percentage));
      }

      static void InfoPrint_func(string name, string sur, string mid, string
billnum, int bill, double percentage, Func<string, int, double, double> billinfo)
      {
         Console.WriteLine("Client information");
         Console.WriteLine("Name {0}", name);
         Console.WriteLine("Surname {0}", sur);
         Console.WriteLine("Middle name {0}", mid);
         Console.WriteLine("{0}", billinfo(billnum, bill, percentage));
      }
      static void Main(string[] args)
      {
         Console.WriteLine("Sergeev Maxim IU5-33b\n");
         // вызов метода для определения суммы на счету после
определения процентов
         // 2способа
```

```
        Console.WriteLine("Calling a method passing the Calc method as a
parameter");
        InfoPrint("Sergeev", "Maxim", "Yurevich", "048931572", 21325, 0.15,
Calc);
        Console.WriteLine("\n");

        Console.WriteLine("Method call passing lambda expression as
parameter");
        InfoPrint("Sergeev", "Maxim", "Yurevich", "048931572", 21325, 0.15,
                  (string billnum, int bill, double percentage) =>
                  {
                     Console.WriteLine("Account number: {0}", billnum);
                     Console.Write("Account amount: ");
                     return bill * (1 + percentage);
                  });
        Console.WriteLine("\n");
        // объявление лямбда-выражения
        Func<string, int, double, double> NewBill_3;
        NewBill_3 = (string billnum, int bill, double percentage) =>
        {
           Console.WriteLine("Account number: {0}", billnum);
           Console.Write("Account amount: ");
           return bill * (1 + percentage);
        };
        // вызов метода для определения суммы на счету после
определения процентов
        //с сипользованием лямбда-выражения
        Console.WriteLine("Method call passing Func lambda expression as
parameter");
        InfoPrint_func("Sergeev", "Maxim", "Yurevich", "048931572", 21325,
0.15, Calc);
      }
   }
}
Program.cs
using System;
using System.Linq;
using System.Reflection;
namespace Lab6_2
{
```

```
class Reflection
{
    static void Main(string[] args)
    {
        Console.WriteLine("Sergeev Maxim IU5-33B\n");
        //  создадим объект типа Rect
        Console.WriteLine("Quadrate");
        Rect MyQuadrate = new Rect(5, 8);
        Console.WriteLine("Method");
        MyQuadrate.Area();
        Console.WriteLine("Interface");
        MyQuadrate.Print();
        Console.WriteLine("\n");
        //  вывод информации о классе при помощи рефлексии
конструкторах, свойствах, методах.
        Type t = MyQuadrate.GetType();
        Console.WriteLine("\nType information:");
        Console.WriteLine("Type " + t.FullName + " inherited from " +
t.BaseType.FullName);
        Console.WriteLine("Namespace " + t.Namespace);
        Console.WriteLine("Found in assembly " + t.AssemblyQualifiedName);
        Console.WriteLine("\nConstructors:");
        foreach (var x in t.GetConstructors())
        {
            Console.WriteLine(x);
        }
        Console.WriteLine("\nMethods:");
        foreach (var x in t.GetMethods())
        {
            Console.WriteLine(x);
        }
        Console.WriteLine("\nProperties:");
        foreach (var x in t.GetProperties())
        {
            Console.WriteLine(x);
        }
        Console.WriteLine("\nData fields (public):");
        foreach (var x in t.GetFields())
        {
            Console.WriteLine(x);
```

```
        }
        Console.WriteLine("\nForInspection implements IComparable -> " +
t.GetInterfaces().Contains(typeof(IComparable)));
        // вывод свойств, которым назначен атрибут
        Type t1 = typeof(Rect);
        Console.WriteLine("\nAttribute-marked properties:");
        foreach (var x in t1.GetProperties())
        {
           object attrObj;
           if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))
           {
              NewAttribute attr = attrObj as NewAttribute;  // Для приведения
полученного значения типа object к требуемому типу NewAttribute
              Console.WriteLine(x.Name + " - " + attr.Description);
           }
        }

        Type t2 = typeof(Rect);
        Console.WriteLine("\nMethod call:");
        //Создание объекта
        //Rect rec = new Rect(5, 8);
        //Можно создать объект через рефлексию
        Rect Rec = (Rect)t.InvokeMember(null, BindingFlags.CreateInstance,
null, null, new object[] { 5, 8 });
        //Параметры вызова метода
        object[] parameters = new object[] { 3, 3 };
        //Вызов метода
        object Result = t2.InvokeMember("Area", BindingFlags.InvokeMethod,
null, Rec, new object[] { });
        //Console.WriteLine("Area(3,3)={0}", (int)Result);
        // Метод InvokeMember принимает различные
     }
     [AttributeUsage(AttributeTargets.Property, AllowMultiple = false,
Inherited = false)]
     public class NewAttribute : Attribute
     {
        public NewAttribute() { }
        public NewAttribute(string DescriptionParam) { Description =
DescriptionParam; }
        public string Description { get; set; }
```

```
        }
        // Проверка, что у свойства есть атрибут заданного типа
        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;
        }
        interface IPrint
        {
            void Print();
        }
        abstract class Geometrical_figure
        {
            public virtual void Area() { }
            public virtual void ToString() { }
        }
        class Rectangle : Geometrical_figure, IPrint
        {
            public Rectangle(double length, double height)
            {
                this.length = length;
                this.height = height;
            }
            private double length;
            private double height;
            override public void Area()
            {
                Console.WriteLine("Height: {0} Length: {1} Square area = {2}",
height.ToString(), length.ToString(), (length * height).ToString());
            }
            public void Print()
            {
```

```
            Area();
        }
        [New("Description for Length")]
        public double Length { get; set; }
        [New("Description for Heigth")]
        public double Height { get; set; }
    }
    class Rect : Rectangle, IPrint
    {
        public Rect(double length, double height) : base(length, height)
        {
            this.length = length;
            this.height = height;
        }
        private double length;
        private double height;
        override public void Area()
        {
            Console.WriteLine("Height: {0} Length: {1} Square area = {2}",
height.ToString(), length.ToString(), (length * height).ToString());
        }
        new public void Print()
        {
            Area();
        }
    }
  }
}
```

Экранные формы с примерами выполнения программы: