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:
студент группы РТ5-31Б

Ходосов Михаил
Подпись и дата:

Проверил:
доцент каф. ИУ5

Гапанюк Ю.Е.
Подпись и дата:

Москва, 2020 г.

Описание задания

Часть 1. Разработать программу, использующую делегаты.

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

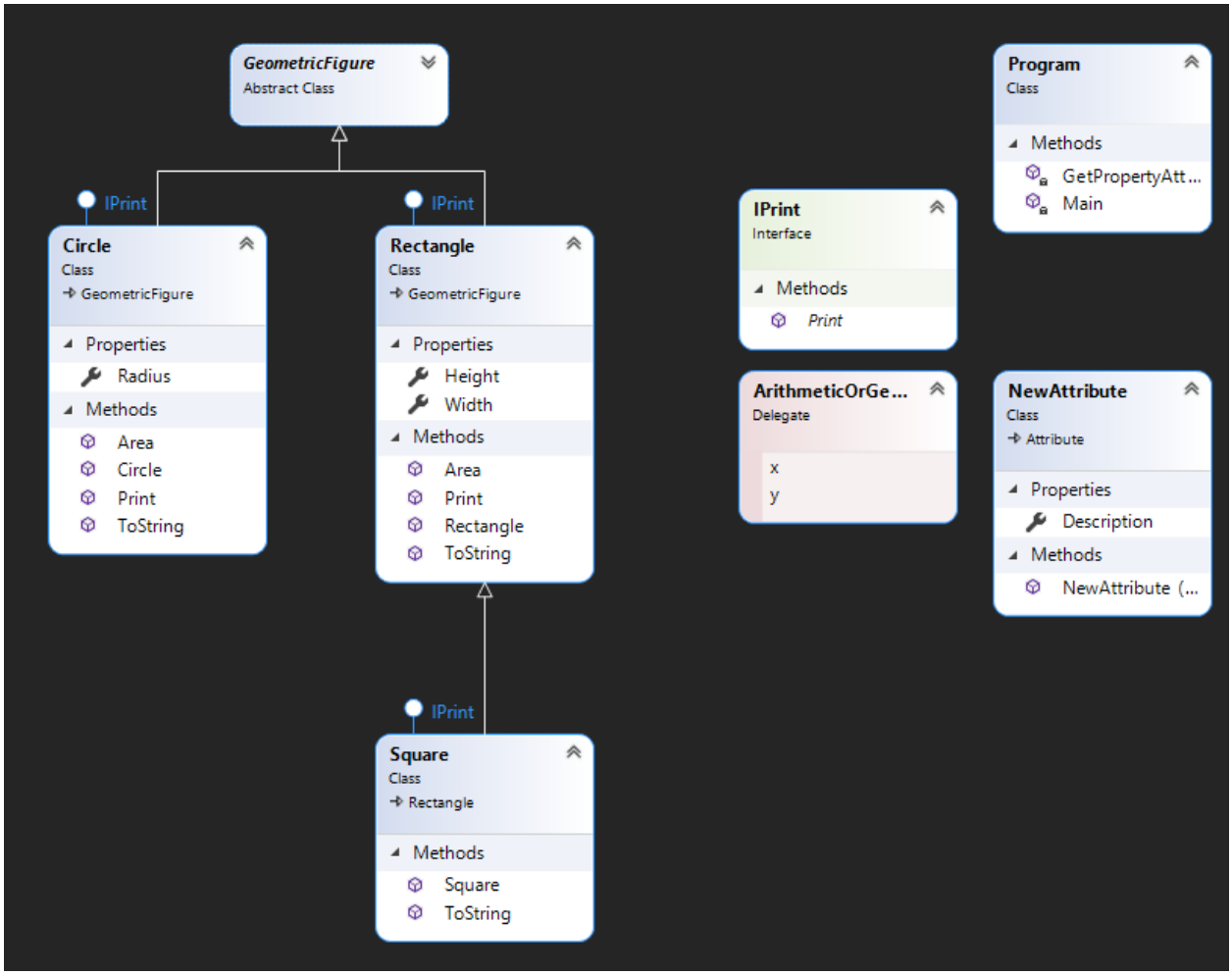
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат,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

Figures.cs

```
using System;

namespace Figures
{
    abstract class GeometricFigure
    {
        public string Type { get; protected set; }

        public abstract double Area();

        public override string ToString()
        {
            return this.Type + " площадью " + this.Area().ToString();
        }
    }

    interface IPrint
    {
        void Print();
    }

    class Rectangle : GeometricFigure, IPrint
    {
        [NewAttribute("Высота")]
        public double Height { get; protected set; }

        public double Width { get; protected set; }

        public Rectangle(double ph, double pw)
        {
            this.Height = ph;
            this.Width = pw;
            this.Type = "Прямоугольник";
        }

        public override double Area()
        {
            return (this.Height * this.Width);
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }

        public override string ToString()
        {
            return this.Type + " шириной " + this.Width.ToString() + ", высотой " +
this.Height.ToString() + ", площадью " + this.Area().ToString();
        }
    }

    class Square: Rectangle, IPrint
    {
        public Square(double size) : base(size, size)
        {
            this.Type = "Квадрат";
        }

        public override string ToString()
```

```
        {
            return this.Type + " со стороной " + this.Width.ToString() + ", площадью " +
this.Area().ToString();
        }
    }

    class Circle : GeometricFigure, IPrint
    {
        public double Radius { get; protected set; }

        public Circle(double pr)
        {
            this.Radius = pr;
            this.Type = "Круг";
        }

        public override double Area()
        {
            return this.Radius * this.Radius * Math.PI;
        }

        public void Print()
        {
            Console.WriteLine(this.ToString());
        }

        public override string ToString()
        {
            return this.Type + " радиусом " + this.Radius.ToString() + ", площадью " +
this.Area().ToString();
        }
    }

    public class NewAttribute : Attribute
    {
        public NewAttribute() { }
        public NewAttribute(string description)
        {
            Description = description;
        }

        public string Description { get; set; }
    }
}
```

Program.cs

```
using System;
using System.Reflection;
using Figures;

namespace lab6
{
    delegate double ArithmeticOrGeometricMean(double x, double y);
    class Program
    {
        static void Main(string[] args)
        {
            double ArithmeticMean(double x, double y)
            {
                return 0.5 * (x + y);
            }

            double GeometricMean(double x, double y)
```

```
            {
                return Math.Sqrt(x * x + y * y);
            }

            void ArithmeticOrGeometricMeanMethod(double x, double y,
ArithmeticOrGeometricMean function)
            {
                Console.WriteLine(function(x, y));
            }

            //Вызов метода с использованием метода соответствующего делегату
            Console.Write("Среднее арифметическое чисел 4 и 6 равно ");
            ArithmeticOrGeometricMeanMethod(4, 6, ArithmeticMean);

            //Вызов метода с использованием лямбда-выражения
            Console.Write("Среднее геометрическое чисел 12 и 9 равно ");
            ArithmeticOrGeometricMeanMethod(12, 9, (double x, double y) => { return
GeometricMean(x, y); });

            void ArithmeticOrGeometricMeanFunc(double x, double y, Func<double, double,
double> func)
            {
                Console.WriteLine(func(x, y));
            }

            Console.WriteLine("\nИспользование обобщенного делегата");
            Console.Write("Среднее арифметическое чисел 11 и 13 равно ");
            ArithmeticOrGeometricMeanFunc(11, 13, ArithmeticMean);

            Console.Write("Среднее геометрическое чисел 33 и 44 равно ");
            ArithmeticOrGeometricMeanFunc(33, 44, (x, y) => { return GeometricMean(x, y);
});


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

            Rectangle obj = new Rectangle(9, 12);
            Type t = obj.GetType();

            Console.WriteLine("Информация о прямоугольнике:\n");
            Console.WriteLine("Тип " + t.FullName + " унаследован от " +
t.BaseType.FullName);
            Console.WriteLine("Пространство имён: " + t.Namespace);
            Console.WriteLine("В сборке " + t.AssemblyQualifiedName);

            Console.WriteLine("\nКонструкторы\n");
            foreach (var constructor in t.GetConstructors())
            {
                Console.WriteLine(constructor);
            }

            Console.WriteLine("\nМетоды\n");
            foreach (var method in t.GetMethods())
            {
                Console.WriteLine(method);
            }

            Console.WriteLine("\nСвойства\n");
            foreach (var property in t.GetProperties())
            {
                Console.WriteLine(property);
            }

            Console.WriteLine("\nВывод свойств с атрибутами:\n");
            foreach (var prop in t.GetProperties())
```

```
            {
                object attrObj;
                if (GetPropertyAttribute(prop, typeof(NewAttribute), out attrObj))
                {
                    NewAttribute attr = attrObj as NewAttribute;
                    Console.WriteLine(prop);
                }
            }

            Console.WriteLine("\nВызов метода класса с помощью рефлексии\n");
            object[] parameters = new object[] { };
            object Result = t.InvokeMember("Area", BindingFlags.InvokeMethod, null, obj,
parameters);
            Console.WriteLine("Площадь прямоугольника = {0}", Result);

        }

        private 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;
        }
    }


}
```

Результат работы программы