Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації та управління

ЗВІТ

до лабораторної роботи № 2

з дисципліни “Основи Web-програмування”

Виконав студент	*ІП-61 Каджая Володимир Миколайович*	
	(№ групи, прізвище, ім’я, по батькові)	
Прийняв	*Ліщук К. І.*	
	(посада, прізвище, ім’я, по батькові)	

Київ 2018

ЗМІСТ

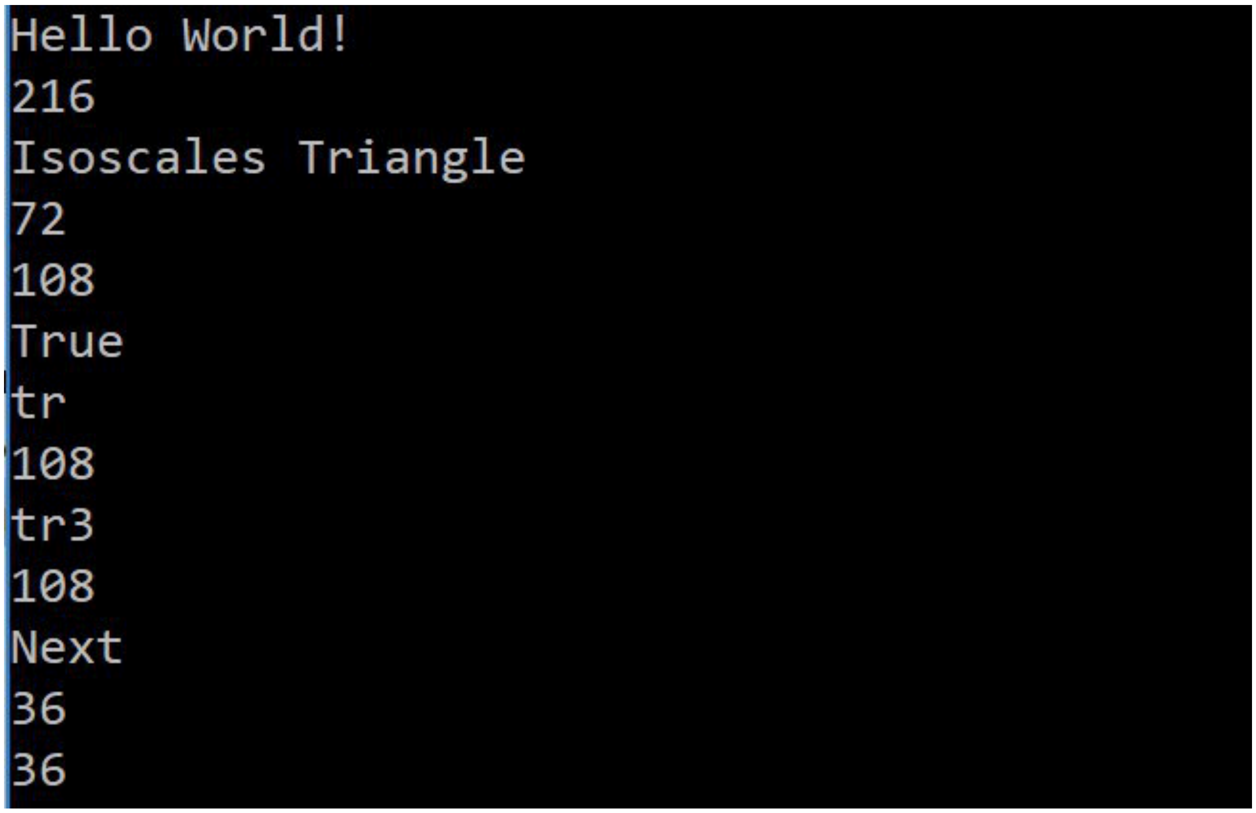
1. ПОСТАНОВКА ЗАДАЧІ

Создать абстрактный класс Triangle (треугольник), задав в нем длину двух сторон, угол между ними, методы вычисления площади и периметра. На его основе создать классы, описывающие равносторонний, равнобедренный и прямоугольный треугольники со своими методами вычисления площади и периметра.

Создать класс Picture, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти.

Предусмотреть возможность вывода характеристик объектов списка и получения суммарной площади.

2. РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ



```
Hello World!
216
Isoscales Triangle
72
108
True
tr
108
tr3
108
Next
36
36
```

3. КОД ПРОГРАМИUSING SYSTEM;

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Printing;
using System.Collections;
using System.Printing.IndexedProperties;
using System.Reflection;
using System.Collections.Generic;

namespace task2
{
class MainClass
{
	public static void Main (string[] args)
	{
		Console.WriteLine ("Hello World!");
		Triangle tr = new Triangle ("Triangle", 18, 18, 36, 12, 23);
		Triangle tr1 = new Triangle ("Triangle", 18, 18, 36, 12, 23);
		Triangle tr2 = new Triangle ("Triangle", 18, 18, 36, 12, 23);

		Triangle tr3 = new Triangle ("Triangle22", 23, 22, 36, 12, 20);

		Triangle[] arr = { tr, tr1, tr2 };
		Picture pic = new Picture (arr);
		pic.SumAllAreas ();
		//Console.WriteLine (pic.SumAllAreas);
```

```
                tr.CallTriangles ();
                tr.Perimeter ();
                tr.GetArea ();
                Console.WriteLine (tr.Equals (tr1));
                //Console.WriteLine(tr.ToString ());
                Console.WriteLine ("tr");
                Console.WriteLine(tr.GetArea ());
                tr3 = tr1.DeepCopy ();
                Console.WriteLine ("tr3");
                Console.WriteLine(tr3.GetArea ());
                Console.WriteLine ("Next");
                Console.WriteLine (tr1.GetHashCode ());
                Console.WriteLine (tr3.GetHashCode ());
        }
}

abstract class IShape
{
        abstract public string GetName(string Name);
        abstract public double GetArea();
}

[Serializable]
class Triangle : IShape
{
        protected double side;
        protected double side2;
        protected double side3;
        protected double height;
        protected double radian;
        protected string name;
        protected double Area;

        public Triangle() {
        }

        public Triangle(string name, double side, double side2, double side3,
double height, double radian)
        {
            if (side >= 0 && side2 >= 0 && side3 >= 0 && height >= 0)
            {
                this.side = side;
                this.side2 = side2;
                this.side3 = side3;
                this.height = height;
                if (radian >= 0 && radian < 180) {
                    this.radian = radian;
                }
                else
                {
                    throw new Exception ("Critical error: Triangle is
not exist");
                }
            }
            else
            {
                throw new Exception ("Critical error: Value can not be
negative");
            }
        }
        public void CallTriangles()
        {
            if (side == side2 || side2 == side3 || side3 == side) {
                Console.WriteLine ("Isoscales Triangle");
                IsoscalesTriangle ist = new IsoscalesTriangle (name,
side, side2, side3, height, radian);
                ist.Perimeters ();
                ist.GetAreas ();
                Console.WriteLine (ist.Perimeters ());
                Console.WriteLine (ist.GetAreas ());
            }
            else if (side == side2 && side2 == side3 && side3 == side) {
                Console.WriteLine ("Equilateral Triangle");
                EquilateralTriangle eqt = new EquilateralTriangle (name,
side, side2, side3, height, radian);
                eqt.Perimeters ();
                eqt.GetAreas ();
                Console.WriteLine (eqt.Perimeters ());
                Console.WriteLine (eqt.GetAreas ());
            }
```

```
                else if(side3 == Math.Pow(side, 2.0) + Math.Pow(side2, 2.0)) {
                        Console.WriteLine ("Right triangle");
                        RightTriangle rgh = new RightTriangle (name, side,
side2, side3, height, radian);
                        rgh.Perimeters ();
                        rgh.GetAreas ();
                        Console.WriteLine (rgh.Perimeters ());
                        Console.WriteLine (rgh.GetAreas ());
                }
                else
                {
                        Perimeter ();
                        GetArea ();
                        Console.WriteLine (Perimeter ());
                        Console.WriteLine (GetArea ());
                }
        }
        public override string GetName(string Name)
        {
                return "Shape: " + Name;
        }
        public override double GetArea()
        {
                double area = (side * height) / 2;
                return area;
        }
        public double Perimeter()
        {
                double P = side + side2 + side3;
                return P;
        }
        public double CountAreas()
        {
                Area += GetArea ();
                return Area;
        }

        public override bool Equals (object obj)
        {
                if (obj == null || GetType () != obj.GetType ()) {
                        return false;
                }
                Triangle tr = (Triangle)obj;
                return (side == tr.side) && (side2 == tr.side2) && (side3 ==
tr.side3);
        }

        public override int GetHashCode ()
        {
                int res = 0;
                if (this.GetType ().Name == "Triangle") {
                        res = ((int)side ^ (int)side2 ^ (int)side3);
                }
                return res;
```

```
            //return base.GetHashCode ();
        }

        public override string ToString ()
        {
            Type type = typeof(Triangle);
            FieldInfo[] fields = type.GetFields (BindingFlags.Public);
            Console.WriteLine ("Displaying the values of the fields of
{0}:", type);
            Triangle tr = new Triangle ();
            String res = "";
            for (int i = 0; i < fields.Length; i++) {
                //Console.WriteLine("{0}:\t'{1}'", fields[i].Name,
fields[i].GetValue(tr));
                res = "{0}:\t'{1}'" + fields[i].Name +
fields[i].GetValue(tr);
            }

            MethodInfo[] methodinfo = type.GetMethods ();
            String mm = "";
            foreach (MethodInfo temp in methodinfo)
            {
                mm = temp.Name;
            }
            String r = "Class is " + type.Name +
                "\n" + "Methods are " + mm;
            return res + "\n" + r;
        }

        public Triangle DeepCopy()
        {
            return (Triangle)this.MemberwiseClone ();
        }

        public static bool operator==(Triangle tr1, Triangle tr2)
        {
            if (tr1.Equals (tr2))
                return true;
            return false;
        }

        public static bool operator!=(Triangle tr1, Triangle tr2)
        {
            if (tr1.Equals (tr2))
                return false;
            return true;
        }
    }

    class IsoscalesTriangle : Triangle
    {
        public IsoscalesTriangle(string name, double side, double side2,
double side3, double height, double radian) :
        base(name, side, side2, side3, height, radian)
```

```
		{}
		public double GetAreas()
		{
			return base.GetArea ();
		}
		public double Perimeters()
		{
			return base.Perimeter ();
		}
		public double CountAreas()
		{
			return base.CountAreas ();
		}
	}

	class EquilateralTriangle : Triangle
	{
		public EquilateralTriangle(string name, double side, double side2,
double side3, double height, double radian) :
		base(name, side, side2, side3, height, radian)
		{}
		public double GetAreas()
		{
			return base.GetArea ();
		}
		public double Perimeters()
		{
```

```
                return base.Perimeter ();
            }
            public double CountAreas()
            {
                return base.CountAreas ();
            }
        }

        class RightTriangle : Triangle
        {
            public RightTriangle(string name, double side, double side2, double
    side3, double height, double radian) :
            base(name, side, side2, side3, height, radian)
            {}
            public double GetAreas()
            {
                return base.GetArea ();
            }
            public double Perimeters()
            {
                return base.Perimeter ();
            }
            public double CountAreas()
            {
                return base.CountAreas ();
            }
        }
```

```
class Picture
{
    private Triangle[] tring;
    private double tr;

    public Picture(Triangle[] tring)
    {
        if (tring.Length < 1) {
            throw new Exception ("Object of arrays are too less");
        }
        else
        {
            this.tring = tring;
        }
    }
    public void SumAllAreas()
    {
        foreach (Triangle triang in tring)
        {
            triang.CountAreas ();
            tr = triang.CountAreas ();
        }
        Console.WriteLine (tr);
    }
    public void AddElement(Triangle newTr)
    {
```

```
                Triangle[] tr = new Triangle[tring.Length + 1];
                Array.Copy (tring, tr, tring.Length);
                tr [tring.Length] = newTr;
                tring = tr;
                Console.WriteLine ("Added new element -> ", newTr.GetType
().Name);
            }
            public void RemoveAt(int indexer)
            {
                if (indexer < 0 || indexer > tring.Length - 1)
                {
                    throw new Exception ("Out of rangre!");
                }
                string elemName = tring [tring.Length - 1].GetType ().Name;
                Triangle[] tr = new Triangle[tring.Length - 1];
                if (indexer > 0)
                {
                    Array.Copy (tring, 0, tr, 0, indexer);
                }
                if (indexer < tring.Length - 1)
                {
                    Array.Copy (tring, indexer + 1, tr, indexer, tring.Length -
indexer - 1);
                }
                tring = tr;
                Console.WriteLine ("Delete element -> ", elemName);
            }
```

```
        }
        }

        public class StackOverflowException : Exception
        {
        public StackOverflowException() : base("Your stack is overflow!") {}
        public StackOverflowException(string message, Exception inner) :
base(message, inner) {}
        protected
StackOverflowException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) {}
        }

        public class DivideByZeroException : Exception
        {
        public DivideByZeroException() : base("Division by zero!") {}

        public DivideByZeroException(string message)
            : base(message)
        {
        }

        public DivideByZeroException(string message, Exception inner) :
base(message, inner) {}

        protected
DivideByZeroException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) {}
```

```
    }

    public class ArrayTypeMismatchException : Exception
    {
    public ArrayTypeMismatchException() : base("The array has another
type!") {}

    public ArrayTypeMismatchException(string message) : base(message) {}

    public ArrayTypeMismatchException(string message, Exception inner) :
base(message, inner) {}

    protected
ArrayTypeMismatchException(System.Runtime.Serialization.SerializationInfo
info,
        System.Runtime.Serialization.StreamingContext context) {}
    }

    public class IndexOutOfRangeException : Exception
    {
    public IndexOutOfRangeException() : base("Out of range!") {}

    public IndexOutOfRangeException(string message) : base(message) {}

    public IndexOutOfRangeException(string message, Exception inner) :
base(message, inner) {}
```

```
        protected IndexOutOfRangeException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) {}
        }

        public class InvalidCastException : Exception
        {
        public InvalidCastException() : base("Invalid cast!") {}

        public InvalidCastException(string message) : base(message) {}

        public InvalidCastException(string message, Exception inner) : base(message, inner) {}

        protected InvalidCastException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) {}
        }


        public class OutOfMemoryException : Exception
        {
        public OutOfMemoryException() : base("Out of memmory!") {}

        public OutOfMemoryException(string message) : base(message) {}
        public OutOfMemoryException(string message, Exception inner) :
base(message, inner) {}

        protected
OutOfMemoryException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) { }
        }

        public class OverflowException : Exception
        {
        public OverflowException() : base("Overflow!") {}

        public OverflowException(string message) : base(message) {}

        public OverflowException(string message, Exception inner) : base(message,
inner) {}

        protected
OverflowException(System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext context) {}
        }
```