МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмної інженерії»

ЗВІТ

з лабораторної роботи №3

з дисципліни «Об’єктно-орієнтоване програмування»

|  |  |
| --- | --- |
| Виконав:  ст. гр. ПЗПІ-19-2  Купріянов М.О. | Прийняла:  Русакова Н.Є. |

Харків 2020

3.1 Мета роботи

Вариант 2

Розробити систему класів для геометричних фігур

1. Визначте систему класів:

«Фигура», «Точка» ( или «Отрезок»), «Треугольник», «Заштрихованный треугольник», «Правильный треугольник», «Прямоугольный треугольник», «Тетраэдр»

2. Визначте в цих класах методи, які:

• переміщують фігуру по площині (на задану відстань або в потрібну позицію);

• масштабують фігуру;

• обчислюють і повертають площа фігури, периметр;

• повертають рядок символів, що відображає ім'я класу і стан об'єкта (його основні характеристики);

• малюють фігуру в консолі або на формі (для об'ємних фігур досить проекції, наприклад, ізометричної або діметричної);

3. Визначте в ваших класах властивості і індексатори (хоча б в деяких класах).

4. Класи повинні бути пов'язані відношенням успадкування (там, де це має сенс)

Деякі класи необхідно зробити абстрактними, а деякі методи - віртуальними

Повинен бути хоча б один поліморфний клас.

Максимально виключіть дублювання коду (поля, методи, реалізація).

5. Оголосіть клас «Зображення», який має своє становище на площині і розмір, і володіє колекцією різних фігур.

Елементами колекції можна буде зробити базовий поліморфний клас (таким чином, в одній колекції можна буде зберегти всі фігури зображення)

6. У класі «Зображення» визначте методи:

• повертають сумарну площу фігур, сумарний периметр;

• зсувний всі фігури всередині зображення;

• переміщуючий зображення;

• масштабуючий зображення (що змінює його розмір разом з фігурами зі збереженням пропорцій);

• повертає повне стан зображення у вигляді рядка;

• який малює все фігури в консолі або на формі;

• зберігає і завантажує зображення з файлу;

3.2 Виконання роботи

У ході виконання лабораторної роботи було написано програму мовою С#. Код класу Image та код його запуску в класі Program приведений на рис. 3.1 – 3.8. Результат інтерпретації коду приведений на рис. 1.9. Код допоміжних класів (класів фігур) міститься в додатку А.

Рис. 3.8 – Код програми.

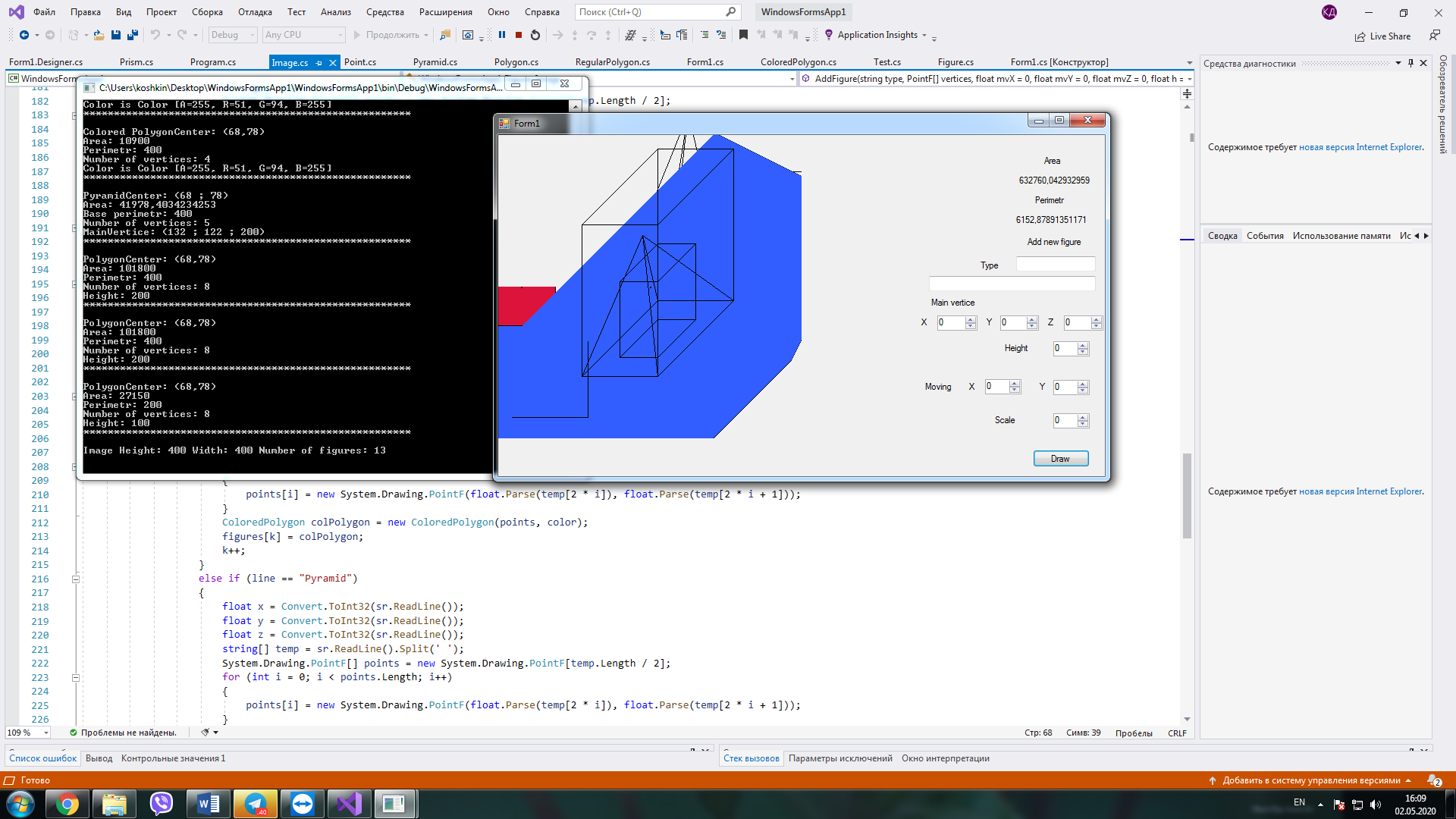


Рис. 3.1 – Результат інтерпретації програми

ВИСНОВКИ

В ході виконання третьої лабораторної роботи я вдосконалив навички написання програми мовою C#. Я розібрався с основними принципами об’єктно-орієнтованого програмування, навчився створювати методи та працювати зі классами. Засвоїв знання, отримані з курсу «Основи програмування» у новому спектрі.

ДОДАТОК А

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

abstract class Figure

{

public float CenterX { protected set; get; }

public float CenterY { protected set; get; }

public abstract void Drawing(System.Drawing.Graphics graphics);

public abstract void MoveTo(float x, float y);

public abstract void MoveBy(float dx, float dy);

public abstract void Scale(float n);

public abstract string Print();

public abstract double Area();

public abstract double Perimetr();

public abstract void SaveToFile(StreamWriter sr);

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class Point : Figure

{

public Point(float x, float y)

{

this.CenterX = x;

this.CenterY = y;

}

public override double Area()

{

return 0;

}

public override void Drawing(System.Drawing.Graphics graphics)

{

float tempY = Convert.ToInt32(graphics.VisibleClipBounds.Height) - CenterY;

float tempX = CenterX;

System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Brushes.Black);

graphics.DrawRectangle(pen, tempX, tempY, 1, 1);

}

public override void MoveTo(float x, float y)

{

CenterX = x;

CenterY = y;

}

public override void MoveBy(float dx, float dy)

{

CenterX += dx;

CenterY += dy;

}

public override double Perimetr()

{

return 0;

}

public override string Print()

{

return "Point" + "Center: (" + CenterX + "," + CenterY + ") " + "Area: " + this.Area() + " " + "Perimetr: " + this.Perimetr();

}

public override void Scale(float x)

{

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Point");

sr.Write(CenterX);

sr.Write(" ");

sr.Write(CenterY);

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class Polygon : Figure

{

public System.Drawing.PointF[] Vertices { set; get; }

public Polygon (System.Drawing.PointF[] vertices)

{

Vertices = vertices;

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void Drawing(System.Drawing.Graphics graphics)

{

System.Drawing.PointF[] tempVertices = new System.Drawing.PointF[Vertices.Length];

for (int i = 0; i < Vertices.Length; i++)

{

tempVertices[i].Y = Convert.ToInt32(graphics.VisibleClipBounds.Height) - Vertices[i].Y;

tempVertices[i].X = Vertices[i].X;

}

System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.Black);

graphics.DrawPolygon(pen, tempVertices);

}

public override void MoveTo(float x, float y)

{

float dx = CenterX - x;

float dy = CenterY - y;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X -= dx;

Vertices[i].Y -= dy;

}

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void MoveBy(float dx, float dy)

{

CenterX += dx;

CenterY += dy;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X += dx;

Vertices[i].Y += dy;

}

}

public override void Scale(float n)

{

float tempX = CenterX;

float tempY = CenterY;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X \*= n;

Vertices[i].Y \*= n;

}

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

this.MoveTo(tempX, tempY);

}

public override string Print()

{

return "Polygon" + "Center: (" + CenterX + "," + CenterY + ") " + "\n" + "Area: " + this.Area() + "\n" + "Perimetr: " + this.Perimetr() + "\n" + "Number of vertices: " + Vertices.Length;

}

public override double Area()

{

float sum1 = 0;

float sum2 = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

sum1 += Vertices[i].X \* Vertices[i + 1].Y;

sum2 += Vertices[i].Y \* Vertices[i + 1].X;

}

return Math.Abs((sum1 - sum2) / 2);

}

public override double Perimetr()

{

double sum = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

double x = Vertices[i].X - Vertices[i + 1].X;

double y = Vertices[i].Y - Vertices[i + 1].Y;

sum += Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2));

}

double x1 = Vertices[0].X - Vertices[Vertices.Length - 1].X;

double y1 = Vertices[0].Y - Vertices[Vertices.Length - 1].Y;

sum += Math.Sqrt(Math.Pow(x1, 2) + Math.Pow(y1, 2));

return sum;

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Polygon");

foreach(System.Drawing.PointF point in Vertices)

{

sr.Write(point.X);

sr.Write(" ");

sr.Write(point.Y);

sr.Write(" ");

}

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class RegularPolygon : Polygon

{

public RegularPolygon(System.Drawing.PointF[] vertices) : base(vertices) { }

public override double Perimetr()

{

double x = Vertices[0].X - Vertices[1].X;

double y = Vertices[0].Y - Vertices[1].Y;

return (Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2))) \* Vertices.Length;

}

public override string Print()

{

return "Regular Polygon" + "Center: (" + CenterX + "," + CenterY + ") " + "\n" + "Area: " + this.Area() + " " + "\n" + "Perimetr: " + this.Perimetr() + "\n" + "Number of vertices: " + Vertices.Length;

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Regular polygon");

foreach (System.Drawing.PointF point in Vertices)

{

sr.Write(point.X);

sr.Write(" ");

sr.Write(point.Y);

sr.Write(" ");

}

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class ColoredPolygon : Polygon

{

public System.Drawing.Color Color { set; get; }

public ColoredPolygon(System.Drawing.PointF[] vertices, System.Drawing.Color color): base(vertices)

{

Color = color;

}

public override void Drawing(System.Drawing.Graphics graphics)

{

System.Drawing.PointF[] tempVertices = new System.Drawing.PointF[Vertices.Length];

//Console.WriteLine("height " + graphics.ClipBounds.Height);

for (int i = 0; i < Vertices.Length; i++)

{

//Console.WriteLine(" my X " + Vertices[i].X + ", y " + Vertices[i].Y);

tempVertices[i].Y = graphics.VisibleClipBounds.Height - Vertices[i].Y;

tempVertices[i].X = Vertices[i].X;

//Console.WriteLine("X " + tempVertices[i].X + ", y " + tempVertices[i].Y);

}

System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.Black);

graphics.DrawPolygon(pen, tempVertices);

System.Drawing.SolidBrush brush = new System.Drawing.SolidBrush(Color);

graphics.FillPolygon(brush, tempVertices);

}

public override string Print()

{

return "Colored Polygon" + "Center: (" + CenterX + "," + CenterY + ") " + "\n" + "Area: " + this.Area() + " " + "\n" + "Perimetr: " + this.Perimetr() + "\n" + "Number of vertices: " + Vertices.Length + "\n" + "Color is " + Color;

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Colored polygon");

sr.WriteLine(System.Drawing.ColorTranslator.ToHtml(System.Drawing.Color.FromArgb(Color.ToArgb())));

foreach (System.Drawing.PointF point in Vertices)

{

sr.Write(point.X);

sr.Write(" ");

sr.Write(point.Y);

sr.Write(" ");

}

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class Pyramid : Figure

{

public float MainVerticeX { get; set; }

public float MainVerticeY { get; set; }

public float MainVerticeZ { get; set; }

public System.Drawing.PointF[] Vertices { set; get; }

public Pyramid (float mainVerticeX, float mainVerticeY, float mainVerticeZ, System.Drawing.PointF[] vertices)

{

MainVerticeX = mainVerticeX;

MainVerticeY = mainVerticeY;

MainVerticeZ = mainVerticeZ;

Vertices = vertices;

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void Drawing(System.Drawing.Graphics graphics)

{

float X3D = graphics.VisibleClipBounds.Width/2;

float Y3D = graphics.VisibleClipBounds.Height / 2;

System.Drawing.Pen penn = new System.Drawing.Pen(System.Drawing.Brushes.Black);

graphics.DrawRectangle(penn, X3D, Y3D, 1, 1);

float TMX = X3D + (MainVerticeY - MainVerticeX / Convert.ToInt32(Math.Sqrt(2.0)));

float TMY = Y3D + MainVerticeX / Convert.ToInt32(Math.Sqrt(2.0)) - MainVerticeZ;

System.Drawing.PointF[] tempVertices = new System.Drawing.PointF[Vertices.Length];

for (int i = 0; i < Vertices.Length; i++)

{

tempVertices[i].Y = Y3D + Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0));

tempVertices[i].X = X3D + (Vertices[i].Y - Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0)));

}

System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.Black);

graphics.DrawLine(pen, tempVertices[0].X, tempVertices[0].Y, tempVertices[tempVertices.Length - 1].X, tempVertices[tempVertices.Length - 1].Y);

for (int i = 0; i < tempVertices.Length - 1; i++)

{

graphics.DrawLine(pen, tempVertices[i].X, tempVertices[i].Y, tempVertices[i + 1].X, tempVertices[i + 1].Y);

graphics.DrawLine(pen, tempVertices[i].X, tempVertices[i].Y, TMX, TMY);

}

graphics.DrawLine(pen, tempVertices[tempVertices.Length - 1].X, tempVertices[tempVertices.Length - 1].Y, TMX, TMY);

}

public override void MoveTo(float x, float y)

{

float dx = CenterX - x;

float dy = CenterY - y;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X -= dx;

Vertices[i].Y -= dy;

}

MainVerticeX -= dx;

MainVerticeY -= dy;

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void MoveBy(float dx, float dy)

{

CenterX += dx;

CenterY += dy;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X += dx;

Vertices[i].Y += dy;

}

MainVerticeX += dx;

MainVerticeY += dy;

}

public override void Scale(float n)

{

float tempX = CenterX;

float tempY = CenterY;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X \*= n;

Vertices[i].Y \*= n;

}

MainVerticeY \*= n;

MainVerticeX \*= n;

MainVerticeZ \*= n;

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

this.MoveTo(tempX, tempY);

}

public override string Print()

{

return "Pyramid" + "Center: (" + CenterX + " " + ";" + " " + CenterY + ") " + "\n" + "Area: " + this.Area() + "\n" + "Base perimetr: " + this.Perimetr() + "\n" + "Number of vertices: " + (Vertices.Length + 1) + "\n" + "MainVertice: (" + MainVerticeX + " " + ";" + " " + MainVerticeY + " " + ";" + " " + MainVerticeZ + ") ";

}

public override double Area()

{

double sum = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

double xa = Vertices[i].X - Vertices[i + 1].X;

double ya = Vertices[i].Y - Vertices[i + 1].Y;

double a = Math.Sqrt(Math.Pow(xa, 2) + Math.Pow(ya, 2));

double xb = MainVerticeX - Vertices[i].X;

double yb = MainVerticeY - Vertices[i].Y;

double b = Math.Sqrt(Math.Pow(xb, 2) + Math.Pow(yb, 2) + Math.Pow(MainVerticeZ, 2));

double xc = MainVerticeX - Vertices[i + 1].X;

double yc = MainVerticeY - Vertices[i + 1].Y;

double c = Math.Sqrt(Math.Pow(xc, 2) + Math.Pow(yc, 2) + Math.Pow(MainVerticeZ, 2));

double p = (a + b + c) / 2;

sum += Math.Sqrt(p\*(p - a)\*(p - b)\*(p - c));

}

float sum1 = 0;

float sum2 = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

sum1 += Vertices[i].X \* Vertices[i + 1].Y;

sum2 += Vertices[i].Y \* Vertices[i + 1].X;

}

sum += Math.Abs((sum1 - sum2) / 2);

return sum;

}

public override double Perimetr()

{

double sum = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

double x = Vertices[i].X - Vertices[i + 1].X;

double y = Vertices[i].Y - Vertices[i + 1].Y;

sum += Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2));

}

double x1 = Vertices[0].X - Vertices[Vertices.Length - 1].X;

double y1 = Vertices[0].Y - Vertices[Vertices.Length - 1].Y;

sum += Math.Sqrt(Math.Pow(x1, 2) + Math.Pow(y1, 2));

return sum;

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Pyramid");

sr.WriteLine(MainVerticeX);

sr.WriteLine(MainVerticeY);

sr.WriteLine(MainVerticeZ);

foreach (System.Drawing.PointF point in Vertices)

{

sr.Write(point.X);

sr.Write(" ");

sr.Write(point.Y);

sr.Write(" ");

}

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

class Prism : Figure

{

public float Height { set; get; }

public System.Drawing.PointF[] Vertices { set; get; }

public Prism (float height, System.Drawing.PointF[] vertices)

{

Height = height;

Vertices = vertices;

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void Drawing(System.Drawing.Graphics graphics)

{

float X3D = graphics.VisibleClipBounds.Width / 2;

float Y3D = graphics.VisibleClipBounds.Height / 2;

System.Drawing.PointF[] downVertices = new System.Drawing.PointF[Vertices.Length];

for (int i = 0; i < Vertices.Length; i++)

{

downVertices[i].Y = Y3D + Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0));

downVertices[i].X = X3D + (Vertices[i].Y - Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0)));

}

System.Drawing.PointF[] upVertices = new System.Drawing.PointF[Vertices.Length];

for (int i = 0; i < Vertices.Length; i++)

{

upVertices[i].Y = Y3D + Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0)) - Height;

upVertices[i].X = X3D + (Vertices[i].Y - Vertices[i].X / Convert.ToInt32(Math.Sqrt(2.0)));

}

System.Drawing.Pen pen = new System.Drawing.Pen(System.Drawing.Color.Black);

graphics.DrawLine(pen, downVertices[0].X, downVertices[0].Y, downVertices[Vertices.Length - 1].X, downVertices[Vertices.Length - 1].Y);

graphics.DrawLine(pen, upVertices[0].X, upVertices[0].Y, upVertices[Vertices.Length - 1].X, upVertices[Vertices.Length - 1].Y);

for (int i = 0; i < Vertices.Length - 1; i++)

{

graphics.DrawLine(pen, downVertices[i].X, downVertices[i].Y, downVertices[i + 1].X, downVertices[i + 1].Y);

graphics.DrawLine(pen, upVertices[i].X, upVertices[i].Y, upVertices[i + 1].X, upVertices[i + 1].Y);

graphics.DrawLine(pen, upVertices[i].X, upVertices[i].Y, downVertices[i].X, downVertices[i].Y);

graphics.DrawLine(pen, downVertices[i + 1].X, downVertices[i + 1].Y, upVertices[i + 1].X, upVertices[i + 1].Y);

}

}

public override void MoveTo(float x, float y)

{

float dx = CenterX - x;

float dy = CenterY - y;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X -= dx;

Vertices[i].Y -= dy;

}

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

}

public override void MoveBy(float dx, float dy)

{

CenterX += dx;

CenterY += dy;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X += dx;

Vertices[i].Y += dy;

}

}

public override void Scale(float n)

{

float tempX = CenterX;

float tempY = CenterY;

for (int i = 0; i < Vertices.Length; i++)

{

Vertices[i].X \*= n;

Vertices[i].Y \*= n;

}

float sumX = 0;

float sumY = 0;

foreach (System.Drawing.PointF vertice in Vertices)

{

sumX += vertice.X;

sumY += vertice.Y;

}

CenterX = sumX / Vertices.Length;

CenterY = sumY / Vertices.Length;

this.MoveTo(tempX, tempY);

Height \*= n;

}

public override string Print()

{

return "Polygon" + "Center: (" + CenterX + "," + CenterY + ") " + "\n" + "Area: " + this.Area() + "\n" + "Perimetr: " + this.Perimetr() + "\n" + "Number of vertices: " + Vertices.Length\*2 + "\n" + "Height: " + Height;

}

public override double Area()

{

float sum1 = 0;

float sum2 = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

sum1 += Vertices[i].X \* Vertices[i + 1].Y;

sum2 += Vertices[i].Y \* Vertices[i + 1].X;

}

return this.Perimetr() \* Height + Math.Abs(sum1 - sum2);

}

public override double Perimetr()

{

double sum = 0;

for (int i = 0; i < Vertices.Length - 1; i++)

{

double x = Vertices[i].X - Vertices[i + 1].X;

double y = Vertices[i].Y - Vertices[i + 1].Y;

sum += Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2));

}

double x1 = Vertices[0].X - Vertices[Vertices.Length - 1].X;

double y1 = Vertices[0].Y - Vertices[Vertices.Length - 1].Y;

sum += Math.Sqrt(Math.Pow(x1, 2) + Math.Pow(y1, 2));

return sum;

}

public override void SaveToFile(StreamWriter sr)

{

sr.WriteLine("Prism");

sr.WriteLine(Height);

foreach (System.Drawing.PointF point in Vertices)

{

sr.Write(point.X);

sr.Write(" ");

sr.Write(point.Y);

sr.Write(" ");

}

sr.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Figures

{

public class Image

{

private string CONSTANT\_PATH = @"Figures.txt";

public PictureBox pictureBox { get; private set; }

private System.Drawing.Bitmap bitMap { get; set; }

public float Height { set; get; }

public float Width { set; get; }

public float X { set; get; }

public float Y { set; get; }

public Figure[] Figures { set; get; }

public Image (float x, float y, float width, float height)

{

X = x;

Y = y;

Height = height;

Width = width;

Figures = this.ReadFile(CONSTANT\_PATH);

pictureBox = new PictureBox();

((System.ComponentModel.ISupportInitialize)(this.pictureBox)).BeginInit();

//

// pictureBox1

//

pictureBox.Location = new System.Drawing.Point(Convert.ToInt32(X), Convert.ToInt32(Y));

pictureBox.Name = "pictureBox1";

pictureBox.Size = new System.Drawing.Size(Convert.ToInt32(Width), Convert.ToInt32(Height));

pictureBox.TabIndex = 0;

pictureBox.TabStop = false;

bitMap = new System.Drawing.Bitmap(pictureBox.Size.Width, pictureBox.Size.Height);

pictureBox.Image = bitMap;

}

public void AddFigure(string type, System.Drawing.PointF[] vertices, float mvX = 0, float mvY = 0, float mvZ = 0, float h = 0)

{

Figure[] tempFigures = new Figure[Figures.Length + 1];

for(int i = 0; i < Figures.Length; i++)

{

tempFigures[i] = Figures[i];

}

if (type == "Polygon")

{

Polygon polygon = new Polygon(vertices);

tempFigures[tempFigures.Length - 1] = polygon;

Figures = tempFigures;

}

else if (type == "Regular polygon")

{

RegularPolygon regPolygon = new RegularPolygon(vertices);

tempFigures[tempFigures.Length - 1] = regPolygon;

Figures = tempFigures;

}

else if (type == "Colored polygon")

{

ColoredPolygon colPolygon = new ColoredPolygon(vertices, System.Drawing.ColorTranslator.FromHtml("#335EFF"));

tempFigures[tempFigures.Length - 1] = colPolygon;

Figures = tempFigures;

}

else if (type == "Pyramid")

{

float x = mvX;

float y = mvY;

float z = mvZ;

Pyramid pyramid = new Pyramid(x, y, z, vertices);

tempFigures[tempFigures.Length - 1] = pyramid;

Figures = tempFigures;

}

else

{

float height = h;

Prism prism = new Prism(height, vertices);

tempFigures[tempFigures.Length - 1] = prism;

Figures = tempFigures;

}

}

public double AreaOfFigures ()

{

double sum = 0;

foreach (Figure area in Figures)

{

sum += area.Area();

}

return sum;

}

public double PerimetrOfFigures()

{

double sum = 0;

foreach (Figure perimetr in Figures)

{

sum += perimetr.Perimetr();

}

return sum;

}

public void Move(float dx, float dy)

{

X += dx;

Y += dy;

pictureBox.Location = new System.Drawing.Point(Convert.ToInt32(X), Convert.ToInt32(Y));

}

public void MovingFigures(float dx, float dy)

{

for (int i = 0; i < Figures.Length; i++)

{

Figures[i].MoveBy(dx, dy);

}

this.Drawing();

}

public void Scale(float n)

{

if (n == 0) return;

Height \*= n;

Width \*= n;

for (int i = 0; i < Figures.Length; i++)

{

Figures[i].Scale(n);

}

}

public string Print()

{

return "Image " + "Height: " + Height + " Width: " + Width + " Number of figures: " + Figures.Length;

}

public void Conect(Image image2)

{

Figure[] tempFigures = new Figure[this.Figures.Length + image2.Figures.Length];

for (int i = 0; i < this.Figures.Length; i++)

{

tempFigures[i] = Figures[i];

}

int k = 0;

for (int j = Figures.Length; j < tempFigures.Length; j++)

{

tempFigures[j] = image2.Figures[k];

k++;

}

Figures = tempFigures;

}

public void Drawing()

{

bitMap = new System.Drawing.Bitmap(pictureBox.Size.Width, pictureBox.Size.Height);

System.Drawing.Graphics g = System.Drawing.Graphics.FromImage(bitMap);

for(int i = 0; i < Figures.Length; i++)

{

Figures[i].Drawing(g);

}

pictureBox.Image = bitMap;

}

public Figure[] ReadFile(string path)

{

Figure[] figures;

using (StreamReader sr = (new FileInfo(path)).OpenText())

{

figures = new Figure[Convert.ToInt32(sr.ReadLine())];

string line;

int k = 0;

while ((line = sr.ReadLine()) != null)

{

if(line == "Point")

{

string[] temp = sr.ReadLine().Split(' ');

Point point = new Point(float.Parse(temp[0]), float.Parse(temp[1]));

figures[k] = point;

k++;

}

else if (line == "Polygon")

{

string[] temp = sr.ReadLine().Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

Polygon polygon = new Polygon(points);

figures[k] = polygon;

k++;

}

else if (line == "Regular polygon")

{

string[] temp = sr.ReadLine().Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

RegularPolygon regPolygon = new RegularPolygon(points);

figures[k] = regPolygon;

k++;

}

else if (line == "Colored polygon")

{

System.Drawing.Color color = System.Drawing.ColorTranslator.FromHtml(sr.ReadLine());

string[] temp = sr.ReadLine().Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

ColoredPolygon colPolygon = new ColoredPolygon(points, color);

figures[k] = colPolygon;

k++;

}

else if (line == "Pyramid")

{

float x = Convert.ToInt32(sr.ReadLine());

float y = Convert.ToInt32(sr.ReadLine());

float z = Convert.ToInt32(sr.ReadLine());

string[] temp = sr.ReadLine().Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

Pyramid pyramid = new Pyramid(x, y, z, points);

figures[k] = pyramid;

k++;

}

else

{

float height = float.Parse(sr.ReadLine());

string[] temp = sr.ReadLine().Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

Prism prism = new Prism(height, points);

figures[k] = prism;

k++;

}

}

}

return figures;

}

public void SaveFigures()

{

using (StreamWriter sr = new StreamWriter(CONSTANT\_PATH, false))

{

sr.WriteLine(Figures.Length);

foreach(Figure figure in Figures)

{

figure.SaveToFile(sr);

}

}

foreach (Figure figure in Figures)

{

Console.WriteLine(figure.Print());

Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Console.WriteLine();

}

Console.WriteLine(Print());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using WindowsFormsApp1.Figures;

namespace WindowsFormsApp1

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Image image1 = new Image(0, 0, 400, 400);

image1.SaveFigures();

Form1 form = new Form1(image1);

form.Controls.Add(image1.pictureBox);

image1.Drawing();

form.Paint += new PaintEventHandler(Paint);

Application.Run(form);

}

private static void Paint(object sender, PaintEventArgs e)

{

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using WindowsFormsApp1.Figures;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

public WindowsFormsApp1.Figures.Image Image { set; get; }

public Form1(WindowsFormsApp1.Figures.Image image)

{

Image = image;

InitializeComponent();

label.Text = Convert.ToString(Image.AreaOfFigures());

labelP.Text = Convert.ToString(Image.PerimetrOfFigures());

}

private void draw\_Click(object sender, EventArgs e)

{

if (type.Text.Length == 0) return;

string[] temp = textBox1.Text.Split(' ');

System.Drawing.PointF[] points = new System.Drawing.PointF[temp.Length / 2];

for (int i = 0; i < points.Length; i++)

{

points[i] = new System.Drawing.PointF(float.Parse(temp[2 \* i]), float.Parse(temp[2 \* i + 1]));

}

if (type.Text == "Pyramid")

{

Image.AddFigure(type.Text, points, Convert.ToSingle(MNX.Value), Convert.ToSingle(MNY.Value), Convert.ToSingle(MNZ.Value));

}

else if (type.Text == "Prism")

{

Image.AddFigure(type.Text, points, 0, 0, 0, Convert.ToSingle(height.Value));

}

else Image.AddFigure(type.Text, points);

Image.SaveFigures();

Image.Drawing();

label.Text = Convert.ToString(Image.AreaOfFigures());

labelP.Text = Convert.ToString(Image.PerimetrOfFigures());

Image.MovingFigures(Convert.ToSingle(moveX.Value), Convert.ToSingle(moveY.Value));

Image.Scale(Convert.ToSingle(scaleN.Value));

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

}

}

ДОДАТОК Б

Пример фигур

Colored polygon

0 50 50 50 50 150 0 150

Regular polygon

63 53 113 53 113 103 63 103

Polygon

120 53 150 60 170 80 160 100 130 120 120 90

Pyramid

120

150

200

43 53 93 53 93 103 43 103

Prism

90

53 173 103 173 103 223 53 223