**Лабораторна робота №5**

**Ієрархія об'єктів і групи. Агрегація. Композиція. Асоціація**

**Мета:** Одержати практичні навички створення об'єктів-груп (агрегація, композиція, асоціація).

**Хід роботи**

**Завдання 1**

Розробити ієрархію класів яка включає агрегацію та композицію взявши тему з Лабораторної роботи №3

**Код програми**

**Detail.h**

#ifndef DETAIL\_HPP

#define DETAIL\_HPP

#include <string>

#include <iostream>

class Detail

{

private:

std::string \_d\_name;

public:

Detail();

Detail(const std::string& name);

std::string getName() const;

void setName(const std::string& newName);

void Show() const;

};

#endif // DETAIL\_HPP

**Detail.cpp**

#include "Detail.hpp"

Detail::Detail() : \_d\_name("Unnamed detail")

{

std::cout << "Detail::Detail() called\n";

}

Detail::Detail(const std::string& name) : \_d\_name(name)

{

std::cout << "Detail::Detail(name) called\n";

}

std::string Detail::getName() const { return \_d\_name; }

void Detail::setName(const std::string& newName) { \_d\_name = newName; }

void Detail::Show() const

{

std::cout << "Detail: " << \_d\_name << std::endl;;

}

**Node.h**

#ifndef NODE\_HPP

#define NODE\_HPP

#include <vector>

#include "Detail.hpp"

class Node

{

private:

// Composition

std::vector<Detail>\* details\_;

public:

Node();

void AddDetail(const Detail& newEl);

void RemoveDetails();

const std::vector<Detail> getDetails() const;

void Show() const;

};

#endif // NODE\_HPP

**Node.cpp**

#include "Node.hpp"

#include <iostream>

Node::Node() {

std::cout << "Node::Node() called\n";

details\_ = new std::vector<Detail>();

}

void Node::AddDetail(const Detail& newEl) {

details\_->push\_back(newEl);

}

void Node::RemoveDetails() {

details\_->clear();

}

const std::vector<Detail> Node::getDetails() const {

return (\*details\_);

}

void Node::Show() const {

std::cout << "Node has " << details\_->size() << " details:" << std::endl;;

std::cout << "Details: "<<std::endl;

for (const auto& e : (\*details\_))

e.Show();

}

**Mechanism.h**

#ifndef MECHANISM\_HPP

#define MECHANISM\_HPP

#include <vector>

#include "Node.hpp"

class Mechanism

{

private:

// Agregation

std::vector<Node> nodes\_;

public:

Mechanism(std::vector<Node> nodes);

void AddNode(Node node);

const std::vector<Node>& getNodes() const;

void Show() const;

};

#endif // MECHANISM\_HPP

**Mechanism.cpp**

#include "Mechanism.hpp"

#include <iostream>

Mechanism::Mechanism(std::vector<Node> nodes) {

std::cout << "Mechanism::Mechanism() called\n";

nodes\_ = nodes;

}

void Mechanism::AddNode(Node node) {

nodes\_.push\_back(node);

}

const std::vector<Node>& Mechanism::getNodes() const {

return nodes\_;

}

void Mechanism::Show() const {

std::cout << "Mechanism contains " << nodes\_.size() << " nodes." << std::endl;

for (const auto node : nodes\_)

node.Show();

}

**Product.h**

#ifndef PRODUCT\_HPP

#define PRODUCT\_HPP

#include <vector>

#include "Mechanism.hpp"

class Product

{

private:

std::vector<Mechanism> mechanisms\_;

public:

Product();

void AddMechanism(const Mechanism& mech);

void Show() const;

};

#endif // PRODUCT\_HPP

**Product.cpp**

#include "Product.hpp"

#include <iostream>

Product::Product() {

std::cout << "Product::Product() called\n";

}

void Product::AddMechanism(const Mechanism& mech) {

mechanisms\_.push\_back(mech);

}

void Product::Show() const {

std::cout << "Product has " << mechanisms\_.size() << " mechanisms:\n";

for (const auto& m : mechanisms\_)

m.Show();

}

**Main.cpp**

#include "Product.hpp"

int main() {

Detail d1("Screw");

Detail d2("Bolt");

Detail d3("Gear");

std::cout << "----------------------------------------------------------------------" << std::endl;

Node n1;

n1.AddDetail(d1);

n1.AddDetail(d2);

Node n2;

n2.AddDetail(d3);

std::cout << "----------------------------------------------------------------------" << std::endl;

std::vector<Node> nl{ n1,n2 };

Mechanism mech1(nl);

Product product;

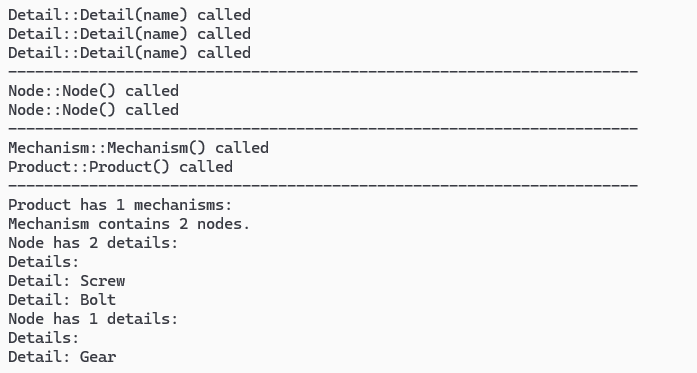
product.AddMechanism(mech1);

std::cout << "----------------------------------------------------------------------" << std::endl;

product.Show();

}

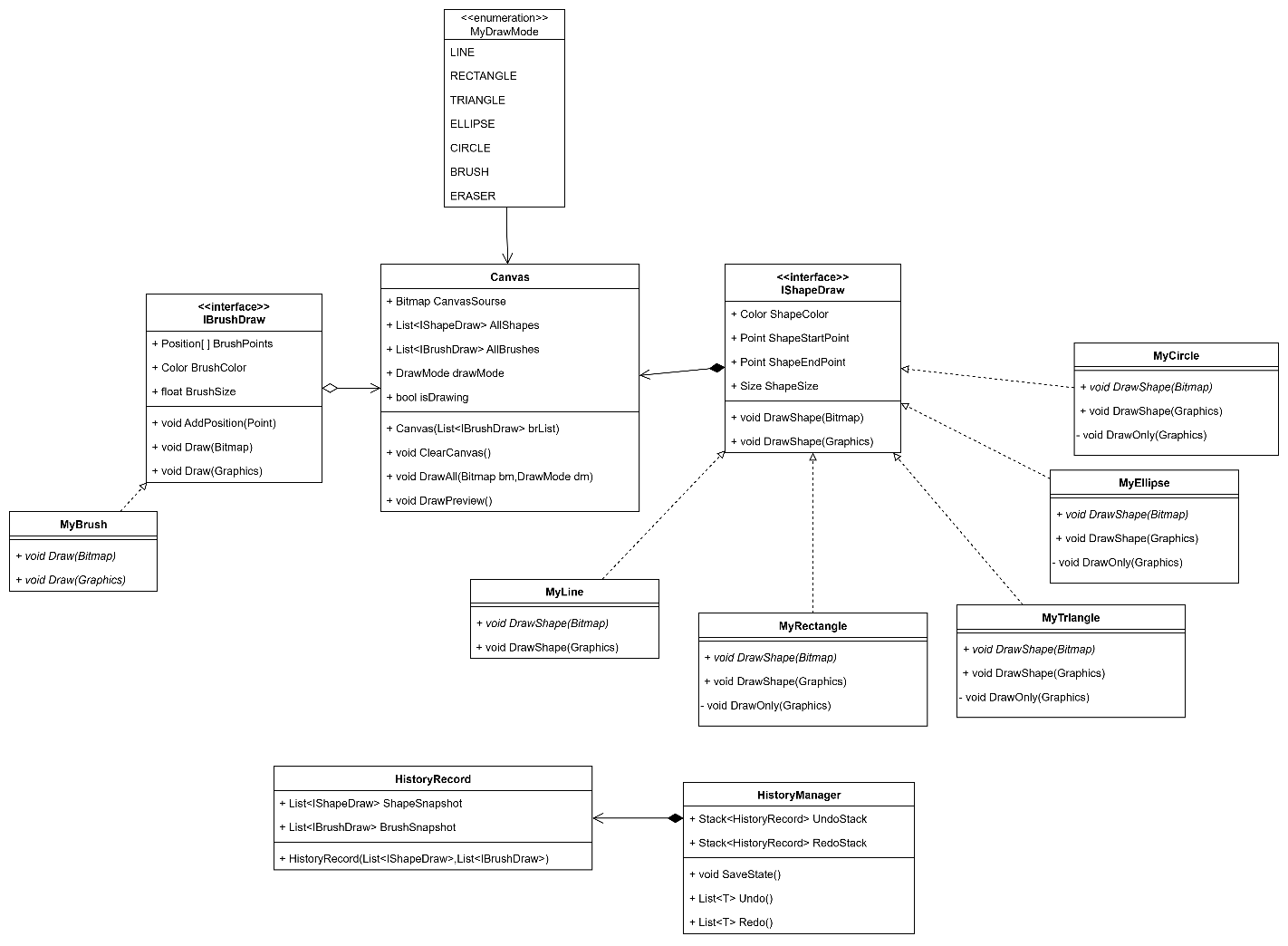
**Результат**

****

**Завдання 2**

В проекті Windows Form створити зв’язки класів за своєю темою використовуючи композицію агрегацію та асоціацію

**Діаграма класів**

****

**Код програми**

**FileManager.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.File

{

public static class FileManager

{

public static Bitmap? LoadFromFile()

{

using OpenFileDialog ofd = new OpenFileDialog();

ofd.Filter = "PNG Image|\*.png|JPEG Image|\*.jpg;\*.jpeg|All files|\*.\*";

if (ofd.ShowDialog() == DialogResult.OK)

{

try

{

return new Bitmap(ofd.FileName);

}

catch (Exception)

{

MessageBox.Show("Cannot open this file as a image.", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);

return null;

}

}

return null;

}

public static void SaveAsToFile(Bitmap bmp)

{

if (bmp == null)

throw new ArgumentNullException(nameof(bmp));

using SaveFileDialog sfd = new SaveFileDialog();

sfd.Filter = "PNG Image|\*.png|JPEG Image|\*.jpg;\*.jpeg";

if (sfd.ShowDialog() == DialogResult.OK)

{

try

{

bmp.Save(sfd.FileName);

}

catch (Exception ex)

{

MessageBox.Show($"Save file error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

**NormaliseMachine.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Nomalisation

{

public static class NormaliseMashine

{

/// <summary>

/// Function to denormalise a point from [0, 1] range to actual pixel coordinates based on the form's client size.

/// </summary>

/// <param name="form"></param>

/// <param name="x"></param>

/// <param name="y"></param>

/// <returns></returns>

/// <exception cref="ArgumentNullException"></exception>

/// <exception cref="ArgumentOutOfRangeException"></exception>

public static Point DenormalisePoint(Form form, float x, float y)

{

if (form == null)

throw new ArgumentNullException(nameof(form));

if (x < 0 || x > 1 || y < 0 || y > 1)

throw new ArgumentOutOfRangeException(nameof(x), "x and y must be in the range [0, 1]");

int denormX = (int)(x \* form.ClientSize.Width);

int denormY = (int)(y \* form.ClientSize.Height);

return new Point(denormX, denormY);

}

public static Point DenormalisePoint(Form form, PointF pointF)

{

if (form == null)

throw new ArgumentNullException(nameof(form));

if (pointF.X < 0 || pointF.X > 1 || pointF.Y < 0 || pointF.Y > 1)

throw new ArgumentOutOfRangeException(nameof(pointF), "x and y must be in the range [0, 1]");

int denormX = (int)(pointF.X \* form.ClientSize.Width);

int denormY = (int)(pointF.Y \* form.ClientSize.Height);

return new Point(denormX, denormY);

}

public static Size DenormaliseSize(Form form, float width, float height)

{

if (form == null)

throw new ArgumentNullException(nameof(form));

if (width < 0 || width > 1 || height < 0 || height > 1)

throw new ArgumentOutOfRangeException(nameof(width), "width and height must be in the range [0, 1]");

int denormWidth = (int)(width \* form.ClientSize.Width);

int denormHeight = (int)(height \* form.ClientSize.Height);

return new Size(denormWidth, denormHeight);

}

public static Size DenormaliseSize(Form form, SizeF sizeF)

{

if (form == null)

throw new ArgumentNullException(nameof(form));

if (sizeF.Width < 0 || sizeF.Width > 1 || sizeF.Height < 0 || sizeF.Height > 1)

throw new ArgumentOutOfRangeException(nameof(sizeF), "width and height must be in the range [0, 1]");

int denormWidth = (int)(sizeF.Width \* form.ClientSize.Width);

int denormHeight = (int)(sizeF.Height \* form.ClientSize.Height);

return new Size(denormWidth, denormHeight);

}

}

}

**MainForm.cs**

using LW\_Daryev\_WinForm\_NewEdition.Nomalisation;

using LW\_Daryev\_WinForm\_NewEdition.File;

using LW\_Daryev\_WinForm\_NewEdition.Draw;

using LW\_Daryev\_WinForm\_NewEdition.Shape;

using LW\_Daryev\_WinForm\_NewEdition.Brush;

using LW\_Daryev\_WinForm\_NewEdition.HisotyManagment;

namespace LW\_Daryev\_WinForm\_NewEdition

{

public delegate void AddShapeToListDelegate(IShapeDraw shape);

public delegate void ShowDrModeOnStatusStripDelegate(string mode);

public partial class MainForm : Form

{

HistoryManager HistoryOnMainForm = new HistoryManager();

public UserCanvas CanvasOnMainForm;

AddShapeToListDelegate AddShapeToList;

public ShowDrModeOnStatusStripDelegate ShowDrModeOnStatusStrip;

public MainForm()

{

InitializeComponent();

InitializeControlsLS();

mainPicture.Image = new Bitmap(mainPicture.Width, mainPicture.Height);

CanvasOnMainForm = new UserCanvas((mainPicture.Image as Bitmap), new List<IBrushDraw>());

SizeToolStripComboBox.Items.AddRange(new object[] { "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "12", "14", "16", "18", "20", "24", "28", "32" });

SizeToolStripComboBox.SelectedIndex = 4;

AddShapeToList += (IShapeDraw obj) => { CanvasOnMainForm.ShapesList.Add(obj); };

ShowDrModeOnStatusStrip += (string mode) => { StatusModeStripValue.Text = $"{mode}"; };

}

public void InitializeControlsLS()

{

menuBarStrip.Location = NormaliseMashine.DenormalisePoint(this, 0.0f, 0.0f);

menuBarStrip.Size = NormaliseMashine.DenormaliseSize(this, 1.0f, 0.05f);

toolsToolStrip.Location = NormaliseMashine.DenormalisePoint(this, 0.0f, 0.1f);

toolsToolStrip.Size = NormaliseMashine.DenormaliseSize(this, 0.11f, 1.0f);

propToolStrip.Location = NormaliseMashine.DenormalisePoint(this, 0.11f, 0.05f);

propToolStrip.Size = NormaliseMashine.DenormaliseSize(this, 0.89f, 0.05f);

mainPicture.Location = NormaliseMashine.DenormalisePoint(this, 0.13f, 0.13f);

mainPicture.Size = NormaliseMashine.DenormaliseSize(this, 0.85f, 0.85f);

}

public void SetControlsLS()

{

mainPicture.Location = NormaliseMashine.DenormalisePoint(this, 0.13f, 0.08f);

mainPicture.Size = NormaliseMashine.DenormaliseSize(this, 0.85f, 0.85f);

}

private void mainPicture\_MouseDown(object sender, MouseEventArgs e)

{

if (CanvasOnMainForm.IsDrawing) return;

CanvasOnMainForm.IsDrawing = true;

float SizeValue;

Color ColorValue = ColorInfoAndChangeToolStripButton.BackColor;

try

{

SizeValue = float.Parse(SizeToolStripComboBox.SelectedItem.ToString());

}

catch

{

SizeToolStripComboBox.SelectedIndex = 4;

SizeValue = float.Parse(SizeToolStripComboBox.SelectedItem.ToString());

}

switch (CanvasOnMainForm.DrMode)

{

case MyDrawMode.NONE: return;

case MyDrawMode.LINE:

AddShapeToList(new MyLine(e.Location, e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.RECTANGLE:

AddShapeToList(new MyRectangle(e.Location, e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.TRIANGLE:

AddShapeToList(new MyTriangle(e.Location, e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.ELLIPSE:

AddShapeToList(new MyEllipse(e.Location, e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.CIRCLE:

AddShapeToList(new MyCircle(e.Location, e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.BRUSH:

CanvasOnMainForm.BrushList.Add(new MyBrush(e.Location, ColorValue, SizeValue));

break;

case MyDrawMode.ERRASER:

CanvasOnMainForm.BrushList.Add(new MyBrush(e.Location, Color.White, SizeValue));

break;

default: return;

}

}

private void mainPicture\_MouseMove(object sender, MouseEventArgs e)

{

if (CanvasOnMainForm.DrMode == MyDrawMode.NONE) return;

if (CanvasOnMainForm.IsDrawing)

{

if (CanvasOnMainForm.DrMode == MyDrawMode.BRUSH || CanvasOnMainForm.DrMode == MyDrawMode.ERRASER)

{

if (CanvasOnMainForm.BrushList.Any())

CanvasOnMainForm.BrushList.Last().AddPosition(e.Location);

}

else

{

CanvasOnMainForm.ShapesList.Last().EndPoint = e.Location;

}

mainPicture.Invalidate();

}

}

private void mainPicture\_MouseUp(object sender, MouseEventArgs e)

{

if (CanvasOnMainForm.DrMode == MyDrawMode.NONE) return;

if (CanvasOnMainForm.IsDrawing)

{

if (CanvasOnMainForm.DrMode == MyDrawMode.BRUSH)

{

if (CanvasOnMainForm.BrushList.Any())

CanvasOnMainForm.BrushList.Last().AddPosition(e.Location);

}

else { CanvasOnMainForm.ShapesList.Last().EndPoint = e.Location; }

CanvasOnMainForm.DrawAll();

mainPicture.Invalidate();

CanvasOnMainForm.IsDrawing = false;

HistoryOnMainForm.SaveState(CanvasOnMainForm.ShapesList, CanvasOnMainForm.BrushList);

}

}

private void mainPicture\_Paint(object sender, PaintEventArgs e)

{

CanvasOnMainForm.DrawPreview(e.Graphics);

}

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

{

ColorDialog colorDialog = new ColorDialog();

if (colorDialog.ShowDialog() == DialogResult.OK)

{

ColorInfoAndChangeToolStripButton.BackColor = colorDialog.Color;

}

}

private void toolStripLabel3\_Click(object sender, EventArgs e) //Undo

{

var state = HistoryOnMainForm.Undo();

if (state == null) return;

CanvasOnMainForm.ShapesList = new List<IShapeDraw>(state.ShapesSnapshot);

CanvasOnMainForm.BrushList = new List<IBrushDraw>(state.BrushSnapshot);

CanvasOnMainForm.ClearCanvas();

CanvasOnMainForm.DrawAll();

mainPicture.Invalidate();

}

private void toolStripLabel5\_Click(object sender, EventArgs e) //Redo

{

var state = HistoryOnMainForm.Redo();

if (state == null) return;

CanvasOnMainForm.ShapesList = new List<IShapeDraw>(state.ShapesSnapshot);

CanvasOnMainForm.BrushList = new List<IBrushDraw>(state.BrushSnapshot);

CanvasOnMainForm.ClearCanvas();

CanvasOnMainForm.DrawAll();

mainPicture.Invalidate();

}

}

}

**MainForm.Events.cs**

using LW\_Daryev\_WinForm\_NewEdition.Draw;

using LW\_Daryev\_WinForm\_NewEdition.File;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition

{

public partial class MainForm

{

private void MainForm\_SizeChanged(object sender, EventArgs e)

{

SetControlsLS();

}

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

{

mainPicture.Image = FileManager.LoadFromFile();

}

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

{

if (mainPicture.Image == null)

return;

FileManager.SaveAsToFile((mainPicture.Image as Bitmap));

}

private void rectDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.RECTANGLE;

ShowDrModeOnStatusStrip(MyDrawMode.RECTANGLE.ToString());

}

private void lineDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.LINE;

ShowDrModeOnStatusStrip(MyDrawMode.LINE.ToString());

}

private void tianDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.TRIANGLE;

ShowDrModeOnStatusStrip(MyDrawMode.TRIANGLE.ToString());

}

private void ellipseDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.ELLIPSE;

ShowDrModeOnStatusStrip(MyDrawMode.ELLIPSE.ToString());

}

private void circleDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.CIRCLE;

ShowDrModeOnStatusStrip(MyDrawMode.CIRCLE.ToString());

}

private void brushDrawTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.BRUSH;

ShowDrModeOnStatusStrip(MyDrawMode.BRUSH.ToString());

}

private void erraseToolTSSB\_ButtonClick(object sender, EventArgs e)

{

CanvasOnMainForm.DrMode = MyDrawMode.ERRASER;

ShowDrModeOnStatusStrip(MyDrawMode.ERRASER.ToString());

}

}

}

**UserCanvas.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LW\_Daryev\_WinForm\_NewEdition.Brush;

using LW\_Daryev\_WinForm\_NewEdition.Shape;

namespace LW\_Daryev\_WinForm\_NewEdition.Draw

{

public enum MyDrawMode

{

NONE,

LINE,

RECTANGLE,

TRIANGLE,

ELLIPSE,

CIRCLE,

BRUSH,

ERRASER

}

public class UserCanvas

{

public Bitmap CanvasSourse { get; set; }

public List<IShapeDraw> ShapesList;

public List<IBrushDraw> BrushList;

public bool IsDrawing { get; set; }

public MyDrawMode DrMode { get; set; }

public UserCanvas(Bitmap bmpSource,List<IBrushDraw> brushList)

{

CanvasSourse = bmpSource;

ShapesList = new List<IShapeDraw>();

BrushList = brushList;

IsDrawing = false;

DrMode = MyDrawMode.NONE;

}

public void ClearCanvas()

{

ShapesList.Clear();

BrushList.Clear();

using (Graphics g = Graphics.FromImage(CanvasSourse))

{

g.Clear(Color.White);

}

}

public void DrawAll()

{

foreach (var shape in ShapesList)

{

shape.DrawShape(CanvasSourse);

}

foreach (var brush in BrushList)

{

brush.DrawBrush(CanvasSourse);

}

}

public void DrawPreview(Graphics graphics)

{

foreach (var shape in ShapesList)

{

if (shape == ShapesList.Last())

{

shape.DrawShape(graphics);

}

}

foreach (var brush in BrushList)

{

if (brush == BrushList.Last())

{

brush.DrawBrush(graphics);

}

}

}

}

}

**IShapeDraw.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

public interface IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public void DrawShape(Bitmap bmp);

public void DrawShape(Graphics graphics); // Preview drawing

}

}

**IBrushDraw.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Brush

{

public interface IBrushDraw

{

public Point[] Position { get; set; }

public Color BrushColor { get; set; }

public float BrushSize { get; set; }

public void AddPosition(Point position);

public void DrawBrush(Bitmap bmp);

public void DrawBrush(Graphics graphics);

}

}

**MyLine.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LW\_Daryev\_WinForm\_NewEdition.Draw;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

internal class MyLine : IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public MyLine(Point startPoint, Point endPoint, Color shapeColor, float shapeSize)

{

StartPoint = startPoint;

EndPoint = endPoint;

ShapeColor = shapeColor;

ShapeSize = shapeSize;

}

public void DrawShape(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

graphics.DrawLine(pen, StartPoint, EndPoint);

}

}

}

public void DrawShape(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

graphics.DrawLine(pen, StartPoint, EndPoint);

}

}

}

}

**MyTriangle.cs**

using LW\_Daryev\_WinForm\_NewEdition.Draw;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

internal class MyTriangle : IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public MyTriangle(Point startPoint, Point endPoint, Color shapeColor, float shapeSize)

{

StartPoint = startPoint;

EndPoint = endPoint;

ShapeColor = shapeColor;

ShapeSize = shapeSize;

}

private void DrawOnly(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

Point point1 = new Point((StartPoint.X + EndPoint.X) / 2, StartPoint.Y);

Point point2 = new Point(StartPoint.X, EndPoint.Y);

Point point3 = new Point(EndPoint.X, EndPoint.Y);

graphics.DrawPolygon(pen, new Point[] { point1, point2, point3 });

}

}

public void DrawShape(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

{

DrawOnly(graphics);

}

}

public void DrawShape(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

DrawOnly(graphics);

}

}

}

}

**MyRectangle.cs**

using LW\_Daryev\_WinForm\_NewEdition.Draw;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

internal class MyRectangle : IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public MyRectangle(Point startPoint, Point endPoint, Color shapeColor, float shapeSize)

{

StartPoint = startPoint;

EndPoint = endPoint;

ShapeColor = shapeColor;

ShapeSize = shapeSize;

}

private void DrawOnly(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

int x = Math.Min(StartPoint.X, EndPoint.X);

int y = Math.Min(StartPoint.Y, EndPoint.Y);

int width = Math.Abs(StartPoint.X - EndPoint.X);

int height = Math.Abs(StartPoint.Y - EndPoint.Y);

graphics.DrawRectangle(pen, x, y, width, height);

}

}

public void DrawShape(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

{

DrawOnly(graphics);

}

}

public void DrawShape(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

DrawOnly(graphics);

}

}

}

}

**MyEllipse.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LW\_Daryev\_WinForm\_NewEdition.Draw;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

internal class MyEllipse : IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public MyEllipse(Point startPoint, Point endPoint, Color shapeColor, float shapeSize)

{

StartPoint = startPoint;

EndPoint = endPoint;

ShapeColor = shapeColor;

ShapeSize = shapeSize;

}

private void DrawOnly(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

int x = Math.Min(StartPoint.X, EndPoint.X);

int y = Math.Min(StartPoint.Y, EndPoint.Y);

int width = Math.Abs(StartPoint.X - EndPoint.X);

int height = Math.Abs(StartPoint.Y - EndPoint.Y);

graphics.DrawEllipse(pen, x, y, width, height);

}

}

public void DrawShape(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

{

DrawOnly(graphics);

}

}

public void DrawShape(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

DrawOnly(graphics);

}

}

}

}

**MyCircle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using LW\_Daryev\_WinForm\_NewEdition.Draw;

namespace LW\_Daryev\_WinForm\_NewEdition.Shape

{

public class MyCircle : IShapeDraw

{

public Point StartPoint { get; set; }

public Point EndPoint { get; set; }

public Color ShapeColor { get; set; }

public float ShapeSize { get; set; }

public MyCircle(Point startPoint, Point endPoint, Color shapeColor, float shapeSize)

{

StartPoint = startPoint;

EndPoint = endPoint;

ShapeColor = shapeColor;

ShapeSize = shapeSize;

}

private void DrawOnly(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

int x = Math.Min(StartPoint.X, EndPoint.X);

int y = Math.Min(StartPoint.Y, EndPoint.Y);

int diameter = Math.Max(Math.Abs(StartPoint.X - EndPoint.X), Math.Abs(StartPoint.Y - EndPoint.Y));

graphics.DrawEllipse(pen, x, y, diameter, diameter);

}

}

public void DrawShape(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

{

DrawOnly(graphics);

}

}

public void DrawShape(Graphics graphics)

{

using (Pen pen = new Pen(ShapeColor, ShapeSize))

{

DrawOnly(graphics);

}

}

}

}

**MyBrush.cs**

using LW\_Daryev\_WinForm\_NewEdition.Brush;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LW\_Daryev\_WinForm\_NewEdition.Brush

{

internal class MyBrush : IBrushDraw

{

public Point[] Position { get; set; }

public Color BrushColor { get; set; }

public float BrushSize { get; set; }

public MyBrush(Point position, Color brushColor, float brushSize)

{

Position = new Point[0];

Position = Position.Append(position).ToArray();

BrushColor = brushColor;

BrushSize = brushSize;

}

public void AddPosition(Point position)

{

if (Position.Length > 0)

{

Point last = Position[^1]; // остання точка

// Додавання проміжних точок для плавності

int dx = position.X - last.X;

int dy = position.Y - last.Y;

int steps = Math.Max(Math.Abs(dx), Math.Abs(dy));

for (int i = 1; i <= steps; i++)

{

int x = last.X + dx \* i / steps;

int y = last.Y + dy \* i / steps;

Position = Position.Append(new Point(x, y)).ToArray();

}

}

else

{

Position = Position.Append(position).ToArray();

}

}

public void DrawBrush(Bitmap bmp)

{

using (Graphics graphics = Graphics.FromImage(bmp))

using (Pen pen = new Pen(BrushColor, BrushSize))

for (int i = 1; i < Position.Length; i++)

graphics.DrawEllipse(pen, Position[i].X, Position[i].Y, BrushSize, BrushSize);

}

public void DrawBrush(Graphics graphics)

{

using (Pen pen = new Pen(BrushColor, BrushSize))

for (int i = 1; i < Position.Length; i++)

graphics.DrawEllipse(pen, Position[i].X, Position[i].Y, BrushSize, BrushSize);

}

}

}

**Висновок:**

На лабораторній роботі було одержано практичні навички створення об'єктів-груп з використанням таких зв’язків як агрегація, композиція, асоціація).