Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Уфимский университет науки и технологий»

Факультет информатики и робототехники

Кафедра вычислительной математики и кибернетики

Лабораторная работа №8

“Дерево объектов, подписка”

По дисциплине “Объектно-ориентированное программирование”

Выполнил:

Студент группы ПРО-232Б

Степанов К.А.

Проверил:

доцент кафедры ВМиК

Котельников В. А.

Уфа-2023

# Цель работы

Создание дерева объектов, используя TreeView и паттерн Observer. Используя этот же паттерн реализовать класс “Однонаправленная стрелка”.

# Задание

* Добавить на форму приложения объект TreeView для отображения текущего содержания хранилища
* Реализовать синхронизацию объекта TreeView с хранилищем с помощью паттерна Observer, при этом должна выполняться синхронизация в обоих направлениях: при выборе объекта в дереве он должен выбираться в рабочей области и наоборот, при выборе объекта в рабочей области он должен выбираться в дереве.
* Реализовать с помощью паттерна Observer новый вид объекта: однонаправленная стрелка, явно создаваемая пользователем и соединяющая объект А с объектом Б таким образом, что при перемещении объекта А перемещается и объект Б.
* Факультативно: сделать стрелку двунаправленной.

# Ход работы

Лабораторная построена на основе ЛР 7, дополняет и расширяет ее возможности. Появляется объект TreeView, который отображает все объекты, а также добавлен новый объект “Однонаправленная стрелка”.

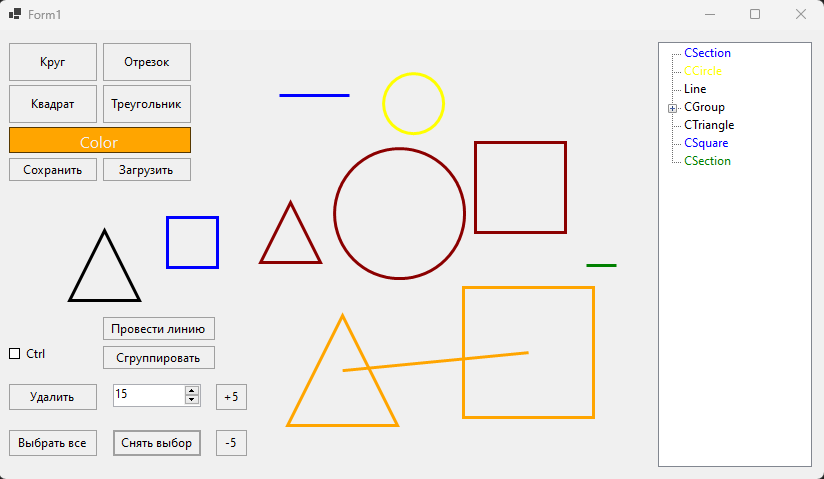


Рис. . Скриншот работы программы

# Вывод

По ходу выполнения работы были закреплены паттерны factory и observer, чтения и записи массивов в файлы.

Ссылка на GitHub: <https://github.com/stkirill15/OOP-8>.

# Приложение 1

**Form1.cs:**

using Microsoft.VisualBasic.Devices;

using System.Data;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Globalization;

using System.Net.Http.Headers;

using System.Reflection;

using System.Text;

using System.Windows.Forms;

using System.Windows.Forms.VisualStyles;

using static OOP6.Form1;

using static System.ComponentModel.Design.ObjectSelectorEditor;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace OOP6

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

SavedData savedData = new SavedData();

SaverLoader loader = new SaverLoader();

private List<CShape> Shapes = new List<CShape>();

List<string> treeData = new List<string>();

public int object\_radius = 10;

public bool Ctrl;

Color[] colors = { Color.Black, Color.Blue, Color.Yellow, Color.Green, Color.Orange, Color.Purple };

Color color = Color.Black;

int colorIndex = 0;

int selectedShape = 0;

private void Form1\_Paint(object sender, PaintEventArgs e) // Отрисовка фигур

{

e.Graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.AntiAlias; // Сглаживание

foreach (CShape shape in Shapes)

{

shape.Draw(e.Graphics); // Метод круга для отрисовки самого себя

}

}

private void Form1\_MouseClick(object sender, MouseEventArgs e)

{

if (!Ctrl) // Если не зажат Ctrl

{

foreach (CShape shape in Shapes) // Снятие выделения со всех фигур

{

shape.Select(false);

}

CShape newShape = null;

switch (selectedShape)

{

case 0:

newShape = new CCircle(e.X, e.Y, object\_radius, color);

break;

case 1:

newShape = new CSquare(e.X, e.Y, object\_radius, color);

break;

case 2:

newShape = new CTriangle(e.X, e.Y, object\_radius, color);

break;

case 3:

newShape = new CSection(e.X, e.Y, object\_radius, color);

break;

}

newShape.Select(true);

Shapes.Add(newShape);

newShape.observers += new System.EventHandler(this.check\_borders);

Refresh();

}

else if (Ctrl) // Если зажат ctrl

{

foreach (CShape shape in Shapes)

{

if (shape.MouseCheck(e)) // Если попала мышь

{

shape.Select(true); // Выделение кругов

break;

}

}

Refresh();

}

SyncLtoTree();

}

private void Form1\_KeyUp(object sender, KeyEventArgs e) // Отжатие кнопки

{

check\_ctrl.Checked = false;

}

private void Form1\_KeyDown(object sender, KeyEventArgs e) // Нажатие кнопок delete и ctrl

{

if (Control.ModifierKeys == Keys.Control)

{

check\_ctrl.Checked = true;

}

else if (e.KeyCode == Keys.Delete)

{

DelFigures();

}

else if (e.KeyCode == Keys.Up)

{

foreach (CShape shape in Shapes)

{

shape.MoveUp(this);

}

Refresh();

}

else if (e.KeyCode == Keys.Down)

{

foreach (CShape shape in Shapes)

{

shape.MoveDown(this);

}

Refresh();

}

else if (e.KeyCode == Keys.Left)

{

foreach (CShape shape in Shapes)

{

shape.MoveLeft(this);

}

Refresh();

}

else if (e.KeyCode == Keys.Right)

{

foreach (CShape shape in Shapes)

{

shape.MoveRight(this);

}

Refresh();

}

else if (e.KeyCode == Keys.Oemplus)

{

Plus5\_button\_Click(sender, e);

}

else if (e.KeyCode == Keys.OemMinus)

{

Minus5\_button\_Click(sender, e);

}

}

public void check\_borders(object sender, EventArgs e)

{

int x = (sender as CShape).coords.X;

int y = (sender as CShape).coords.Y;

int rad = (sender as CShape).radius;

if (x + rad >= this.ClientSize.Width)

(sender as CShape).coords.X = this.ClientSize.Width - rad;

else if (x - rad <= 0)

(sender as CShape).coords.X = rad;

if (y + rad >= this.ClientSize.Height)

(sender as CShape).coords.Y = this.ClientSize.Height - rad;

else if (y - rad <= 0)

(sender as CShape).coords.Y = rad;

}

private void check\_ctrl\_CheckedChanged(object sender, EventArgs e)

{

Ctrl = check\_ctrl.Checked;

foreach (CShape shape in Shapes)

{

shape.Ctrled(Ctrl);

}

}

private void Plus5\_button\_Click(object sender, EventArgs e)

{

if (shapeSize\_NumericUpDown.Value <= shapeSize\_NumericUpDown.Maximum - 5)

shapeSize\_NumericUpDown.Value += 5;

else shapeSize\_NumericUpDown.Value = 100;

foreach (CShape shape in Shapes)

{

shape.GetBigger();

shape.sendShape();

}

Refresh();

}

private void Minus5\_button\_Click(object sender, EventArgs e)

{

if (shapeSize\_NumericUpDown.Value >= shapeSize\_NumericUpDown.Minimum + 5)

shapeSize\_NumericUpDown.Value -= 5;

else shapeSize\_NumericUpDown.Value = 10;

foreach (CShape shape in Shapes)

{

shape.GetSmaller();

}

Refresh();

}

private void delete\_button\_Click(object sender, EventArgs e)

{

DelFigures();

}

void DelFigures() // Метод удаления фигур

{

for (int i = 0; i < Shapes.Count; i++)

{

if (Shapes[i].selected == true)

{

Shapes.Remove(Shapes[i]);

i--;

}

}

Refresh();

SyncLtoTree();

}

private void removeSelection\_button\_Click(object sender, EventArgs e)

{

foreach (CShape figure in Shapes) // снятие выделения со всех объектов

{

figure.Select(false);

}

Refresh();

SyncLtoTree();

}

private void button\_circle\_Click(object sender, EventArgs e)

{

selectedShape = 0;

}

private void button\_square\_Click(object sender, EventArgs e)

{

selectedShape = 1;

}

private void button\_triangle\_Click(object sender, EventArgs e)

{

selectedShape = 2;

}

private void button\_section\_Click(object sender, EventArgs e)

{

selectedShape = 3;

}

private void Color\_Button\_Click(object sender, EventArgs e)

{

colorIndex = colorIndex < colors.Length - 1 ? colorIndex + 1 : 0;

color = colors[colorIndex];

Color\_Button.BackColor = color;

foreach (CShape figure in Shapes) // Выделенные фигуры меняют цвет

{

if (figure.selected)

figure.shape\_color = color;

}

Refresh();

SyncLtoTree();

}

private void Form1\_Load(object sender, EventArgs e)

{

foreach (Control control in this.Controls)

{

control.PreviewKeyDown += new PreviewKeyDownEventHandler(control\_PreviewKeyDown);

}

}

void control\_PreviewKeyDown(object sender, PreviewKeyDownEventArgs e)

{

if (e.KeyCode == Keys.Up || e.KeyCode == Keys.Down || e.KeyCode == Keys.Left || e.KeyCode == Keys.Right)

{

e.IsInputKey = true;

}

}

private void ChoiceAll\_button\_Click(object sender, EventArgs e)

{

foreach (CShape figure in Shapes) // выделения всех объектов

{

figure.Select(true);

}

Refresh();

SyncLtoTree();

}

private void shapeSize\_NumericUpDown\_ValueChanged(object sender, EventArgs e)

{

object\_radius = ((int)shapeSize\_NumericUpDown.Value);

foreach (CShape shape in Shapes)

{

if (shape.selected)

{

shape.radius = (int)shapeSize\_NumericUpDown.Value;

shape.sendShape();

}

}

Refresh();

}

private void Form1\_SizeChanged(object sender, EventArgs e)

{

foreach (CShape shape in Shapes)

{

shape.sendShape();

}

Refresh();

}

private void Group\_Button\_Click(object sender, EventArgs e)

{

CGroup newgroup = new CGroup();

foreach (CShape shape in Shapes)

{

if (shape.selected)

{

newgroup.Add(shape);

}

}

newgroup.iAmGroup = true;

foreach (CShape shape in newgroup.childrens)

{

Shapes.Remove(shape);

}

newgroup.observers += new System.EventHandler(this.check\_borders);

Shapes.Add(newgroup);

Refresh();

SyncLtoTree();

}

private void save\_button\_Click(object sender, EventArgs e)

{

foreach (CShape shape in Shapes)

{

shape.Save(savedData);

}

File.Delete("D:\\test.txt");

loader.Save(savedData, "D:\\test.txt");

}

CShape read(StreamReader sr)

{

string line = sr.ReadLine();

string[] data = line.Split(';');

switch (data[0])

{

case "CGroup":

{

int count = int.Parse(data[1]);

CGroup newfigure = new CGroup();

for (int i = 0; i < count; i++)

{

newfigure.Add(read(sr));

}

newfigure.observers += new System.EventHandler(this.check\_borders);

return newfigure;

}

default:

{

int x = int.Parse(data[2]);

int y = int.Parse(data[3]);

int rad = int.Parse(data[4]);

bool selected = bool.Parse(data[5]);

Color color = Color.FromArgb(int.Parse(data[1]));

switch (data[0])

{

case "CCircle":

{

CCircle newfigure = new CCircle(x, y, rad, color);

newfigure.Select(selected);

newfigure.observers += new System.EventHandler(this.check\_borders);

return newfigure;

}

case "CSquare":

{

CSquare newfigure = new CSquare(x, y, rad, color);

newfigure.Select(selected);

newfigure.observers += new System.EventHandler(this.check\_borders);

return newfigure;

}

case "CTriangle":

{

CTriangle newfigure = new CTriangle(x, y, rad, color);

newfigure.Select(selected);

newfigure.observers += new System.EventHandler(this.check\_borders);

return newfigure;

}

case "CSection":

{

CSection newfigure = new CSection(x, y, rad, color);

newfigure.Select(selected);

newfigure.observers += new System.EventHandler(this.check\_borders);

return newfigure;

}

}

return null;

}

}

}

private void load\_button\_Click(object sender, EventArgs e)

{

ChoiceAll\_button\_Click(this, e);

DelFigures();

StreamReader sr = new StreamReader("D:\\test.txt");

while (!sr.EndOfStream)

{

Shapes.Add(read(sr));

}

sr.Close();

Refresh();

SyncLtoTree();

}

TreeNode readdata(StreamReader sr)

{

string line = sr.ReadLine();

string[] data = line.Split(';');

switch (data[0])

{

case "CGroup":

{

int count = int.Parse(data[1]);

TreeNode newnode = new TreeNode();

newnode.Text = data[0].ToString();

for (int i = 0; i < count; i++)

{

newnode.Nodes.Add(readdata(sr));

}

return newnode;

}

default:

{

Color color = Color.FromArgb(int.Parse(data[1]));

TreeNode treeNode = new TreeNode();

treeNode.Text = data[0].ToString();

if (data[2] == "0")

{

treeNode.ForeColor = color;

}

else

{

treeNode.ForeColor = Color.Red;

}

return treeNode;

}

}

}

public void SyncLtoTree()

{

foreach (CShape shape in Shapes)

{

shape.RetData(treeData);

}

File.WriteAllLines("D:\\tree.txt", treeData);

treeView1.Nodes.Clear();

StreamReader sr = new StreamReader("D:\\tree.txt");

while (!sr.EndOfStream)

{

treeView1.Nodes.Add(readdata(sr));

}

sr.Close();

treeData.Clear();

}

private void treeView1\_AfterSelect(object sender, TreeViewEventArgs e)

{

foreach (CShape shape in Shapes)

{

shape.Select(false);

}

Shapes[e.Node.Index].Select(true);

SyncLtoTree();

e.Node.ForeColor = Color.Purple;

Refresh();

}

private void MakeLine\_button\_Click(object sender, EventArgs e)

{

int objectsSelected = 0;

foreach (CShape shape in Shapes)

{

if (shape.selected) objectsSelected++;

}

if (objectsSelected == 2)

{

Line line = new Line();

foreach (CShape shape in Shapes)

{

if (shape.selected)

line.AddFigure(shape);

}

foreach (CShape fig in line.two\_shapes)

{

Shapes.Remove(fig);

}

Shapes.Add(line);

Refresh();

SyncLtoTree();

}

}

}

}

public class CShape

{

public Point coords; // координаты

public int radius; // радиус

public bool selected = false; // отмеченность

public bool fctrl = false; // зажатый ctrl

public bool iAmGroup = false;

public System.EventHandler observers;

public Color selected\_color = Color.Red; // Цвет "отметки"

public Color shape\_color = Color.Black; // Цвет фигуры

public virtual void Select(bool condition) // метод переключения выделения

{

selected = condition;

}

public virtual void SetColor(Color newcolor)

{

shape\_color = newcolor;

}

public virtual void Ctrled(bool pressed)

{

fctrl = pressed;

}

public virtual void Draw(Graphics g) // Метод для отрисовки самого себя

{

}

public virtual void Save(SavedData savedData) // Метод для сохранения самого себя

{

StringBuilder line = new StringBuilder();

line.Append(ToString()).Append(";");

line.Append(shape\_color.ToArgb()).Append(";");

line.Append(coords.X.ToString()).Append(";");

line.Append(coords.Y.ToString()).Append(";");

line.Append(radius.ToString()).Append(";");

line.Append(selected.ToString()).Append(";");

savedData.linesToWrite.Add(line.ToString());

}

public virtual void RetData(List<string> treeData)

{

StringBuilder line = new StringBuilder();

line.Append(ToString()).Append(";");

line.Append(shape\_color.ToArgb()).Append(";");

if (selected)

{

line.Append("1").Append(";");

}

else

{

line.Append("0").Append(";");

}

treeData.Add(line.ToString());

}

public virtual bool MouseCheck(MouseEventArgs e) // Проверка объекта на попадание в него курсора

{

return false;

}

public virtual void sendShape() // Отправка фигуры обработчику

{

observers.Invoke(this, null);

}

public virtual bool CanMoveDown(Form form)

{

if ((coords.Y + radius) < (int)form.ClientSize.Height)

{

return true;

}

else

{

return false;

}

}

public virtual void GetSmaller()

{

if (selected && radius > 10)

{

radius -= 5;

}

}

public virtual void GetBigger()

{

if (selected && radius <= 95)

{

radius += 5;

}

}

public virtual bool CanMoveLeft(Form form)

{

if ((coords.X - radius) > 0)

{

return true;

}

else

{

return false;

}

}

public virtual bool CanMoveRight(Form form)

{

if ((coords.X + radius) < (int)form.ClientSize.Width)

{

return true;

}

else

{

return false;

}

}

public virtual bool CanMoveUp(Form form)

{

if (((coords.Y - radius) > 0))

{

return true;

}

else

{

return false;

}

}

public virtual void MoveUp(Form form)

{

if (selected && CanMoveUp(form))

{

coords.Y -= 1;

}

}

public virtual void MoveDown(Form form)

{

if (selected && CanMoveDown(form))

{

coords.Y += 1;

}

}

public virtual void MoveLeft(Form form)

{

if (selected && CanMoveLeft(form))

{

coords.X -= 1;

}

}

public virtual void MoveRight(Form form)

{

if (selected && CanMoveRight(form))

{

coords.X += 1;

}

}

}

public class CCircle : CShape // класс круга

{

public CCircle(int x, int y, int radius, Color color) // конструктор по умолчанию

{

coords.X = x;

coords.Y = y;

this.radius = radius;

shape\_color = color;

}

public override void Draw(Graphics g) // Метод для отрисовки самого себя

{

if (selected == true)

g.DrawEllipse(new Pen(selected\_color, 3), coords.X - radius, coords.Y - radius, radius \* 2, radius \* 2);

else

g.DrawEllipse(new Pen(shape\_color, 3), coords.X - radius, coords.Y - radius, radius \* 2, radius \* 2);

}

public override bool MouseCheck(MouseEventArgs e) // Проверка объекта на попадание в него курсора

{

if (fctrl)

{

if (Math.Pow(e.X - coords.X, 2) + Math.Pow(e.Y - coords.Y, 2) <= Math.Pow(radius, 2) && !selected)

{

selected = true;

return true;

}

}

return false;

}

}

public class CSquare : CShape // класс квадрата

{

public CSquare(int x, int y, int radius, Color color) // конструктор по умолчанию

{

coords.X = x;

coords.Y = y;

this.radius = radius;

shape\_color = color;

}

public override void Draw(Graphics g) // Метод для отрисовки самого себя

{

if (selected == true)

g.DrawRectangle(new Pen(selected\_color, 3), coords.X - radius, coords.Y - radius, radius \* 2, radius \* 2);

else

g.DrawRectangle(new Pen(shape\_color, 3), coords.X - radius, coords.Y - radius, radius \* 2, radius \* 2);

}

public override bool MouseCheck(MouseEventArgs e) // Проверка объекта на попадание в него курсора

{

if (fctrl)

{

if (Math.Pow(e.X - coords.X, 2) + Math.Pow(e.Y - coords.Y, 2) <= Math.Pow(radius, 2) && !selected)

{

selected = true;

return true;

}

}

return false;

}

}

public class CTriangle : CShape // класс треугольника

{

public CTriangle(int x, int y, int radius, Color color) // конструктор по умолчанию

{

coords.X = x;

coords.Y = y;

this.radius = radius;

shape\_color = color;

}

public override void Draw(Graphics g) // Метод для отрисовки самого себя

{

Point point1 = new Point(coords.X, coords.Y - radius);

Point point2 = new Point(coords.X + radius, coords.Y + radius);

Point point3 = new Point(coords.X - radius, coords.Y + radius);

Point[] curvePoints = { point1, point2, point3 };

if (selected == true)

g.DrawPolygon(new Pen(selected\_color, 3), curvePoints);

else

g.DrawPolygon(new Pen(shape\_color, 3), curvePoints);

}

public override bool MouseCheck(MouseEventArgs e) // Проверка объекта на попадание в него курсора

{

if (fctrl)

{

if (Math.Pow(e.X - coords.X, 2) + Math.Pow(e.Y - coords.Y, 2) <= Math.Pow(radius, 2) && !selected)

{

selected = true;

return true;

}

}

return false;

}

}

public class CSection : CShape // класс отрезка

{

public CSection(int x, int y, int radius, Color color) // конструктор по умолчанию

{

coords.X = x;

coords.Y = y;

this.radius = radius;

shape\_color = color;

}

public override void Draw(Graphics g) // Отрисовка отрезка

{

Point point1 = new Point(coords.X - radius, coords.Y);

Point point2 = new Point(coords.X + radius, coords.Y);

Point[] curvePoints = { point1, point2 };

if (selected == true)

g.DrawPolygon(new Pen(selected\_color, 3), curvePoints);

else

g.DrawPolygon(new Pen(shape\_color, 3), curvePoints);

}

public override bool MouseCheck(MouseEventArgs e) // Проверка попадания курсора на объект

{

if (fctrl)

{

if (Math.Pow(e.X - coords.X, 2) + Math.Pow(e.Y - coords.Y, 2) <= Math.Pow(radius, 2) && !selected)

{

selected = true;

return true;

}

}

return false;

}

}

public class CGroup : CShape

{

public List<CShape> childrens = new List<CShape>();

public CGroup()

{

}

public void Add(CShape component)

{

component.shape\_color = Color.DarkRed;

component.Select(false);

childrens.Add(component);

}

public override void Ctrled(bool pressed)

{

foreach (CShape component in childrens)

{

component.fctrl = pressed;

}

fctrl = pressed;

}

public override void sendShape() // Отправка фигур обработчику

{

foreach (CShape child in childrens)

{

observers.Invoke(child, null);

}

}

public override void Select(bool condition)

{

foreach (CShape child in childrens)

{

child.Select(condition);

}

selected = condition;

}

public override void Draw(Graphics g)

{

foreach (CShape child in childrens)

{

child.Draw(g);

}

}

public override void SetColor(Color newcolor)

{

foreach (CShape child in childrens)

{

child.SetColor(newcolor);

}

shape\_color = newcolor;

}

public override void Save(SavedData savedData)

{

StringBuilder tmp = new StringBuilder();

tmp.Append(ToString()).Append(";");

tmp.Append(childrens.Count().ToString()).Append(";");

savedData.linesToWrite.Add(tmp.ToString());

foreach (CShape figure in childrens)

{

figure.Save(savedData);

}

}

public override void RetData(List<string> treeData)

{

StringBuilder line = new StringBuilder();

line.Append(ToString()).Append(";");

line.Append(childrens.Count.ToString()).Append(";");

treeData.Add(line.ToString());

foreach (CShape child in childrens)

{

child.RetData(treeData);

}

}

public override bool MouseCheck(MouseEventArgs e)

{

foreach (CShape child in childrens)

{

if (child.MouseCheck(e))

{

return true;

}

}

return false;

}

public override void GetSmaller()

{

foreach (CShape child in childrens)

{

child.GetSmaller();

}

}

public override void GetBigger()

{

foreach (CShape child in childrens)

{

child.GetBigger();

//child.sendShape();

}

}

public override bool CanMoveUp(Form form)

{

foreach (CShape child in childrens)

{

if (!child.CanMoveUp(form))

{

return false;

}

}

return true;

}

public override bool CanMoveDown(Form form)

{

foreach (CShape child in childrens)

{

if (!child.CanMoveDown(form))

{

return false;

}

}

return true;

}

public override bool CanMoveLeft(Form form)

{

foreach (CShape child in childrens)

{

if (!child.CanMoveLeft(form))

{

return false;

}

}

return true;

}

public override bool CanMoveRight(Form form)

{

foreach (CShape child in childrens)

{

if (!child.CanMoveRight(form))

{

return false;

}

}

return true;

}

public override void MoveUp(Form form)

{

if (CanMoveUp(form))

{

foreach (CShape child in childrens)

{

child.MoveUp(form);

}

}

}

public override void MoveDown(Form form)

{

if (CanMoveDown(form))

{

foreach (CShape child in childrens)

{

child.MoveDown(form);

}

}

}

public override void MoveLeft(Form form)

{

if (CanMoveLeft(form))

{

foreach (CShape child in childrens)

{

child.MoveLeft(form);

}

}

}

public override void MoveRight(Form form)

{

if (CanMoveRight(form))

{

foreach (CShape child in childrens)

{

child.MoveRight(form);

}

}

}

}

public class Line : CShape

{

public List<CShape> two\_shapes = new List<CShape>();

public bool shape\_1 = false;

public bool shape\_2 = false;

public void AddFigure(CShape shape)

{

if (two\_shapes.Count < 2)

{

shape.Select(false);

two\_shapes.Add(shape);

}

}

public override void Ctrled(bool pressed)

{

foreach (CShape component in two\_shapes)

{

component.fctrl = pressed;

}

fctrl = pressed;

}

public override void sendShape() // Отправка фигур обработчику

{

}

public override void Select(bool cond)

{

if (!selected)

{

two\_shapes[0].Select(shape\_1);

two\_shapes[1].Select(shape\_2);

selected = cond;

}

else

{

foreach (CShape component in two\_shapes)

{

component.Select(cond);

}

selected = cond;

shape\_1 = cond;

shape\_2 = cond;

}

}

public override bool MouseCheck(MouseEventArgs e)

{

if (two\_shapes[0].MouseCheck(e))

{

shape\_1 = true;

return true;

}

else if (two\_shapes[1].MouseCheck(e))

{

shape\_2 = true;

return true;

}

else

{

shape\_1 = false;

shape\_2 = false;

return false;

}

}

public override void MoveUp(Form form)

{

if (two\_shapes[0].CanMoveUp(form) && two\_shapes[0].selected && !two\_shapes[1].selected)

{

two\_shapes[1].Select(true);

two\_shapes[0].MoveUp(form);

two\_shapes[1].MoveUp(form);

two\_shapes[1].Select(false);

}

else if (two\_shapes[1].CanMoveUp(form) && two\_shapes[1].selected && !two\_shapes[0].selected)

{

two\_shapes[1].MoveUp(form);

}

}

public override void MoveDown(Form form)

{

if (two\_shapes[0].CanMoveDown(form) && two\_shapes[0].selected && !two\_shapes[1].selected)

{

two\_shapes[1].Select(true);

two\_shapes[0].MoveDown(form);

two\_shapes[1].MoveDown(form);

two\_shapes[1].Select(false);

}

else if (two\_shapes[1].CanMoveDown(form) && two\_shapes[1].selected && !two\_shapes[0].selected)

{

two\_shapes[1].MoveDown(form);

}

}

public override void MoveLeft(Form form)

{

if (two\_shapes[0].CanMoveLeft(form) && two\_shapes[0].selected && !two\_shapes[1].selected)

{

two\_shapes[1].Select(true);

two\_shapes[0].MoveLeft(form);

two\_shapes[1].MoveLeft(form);

two\_shapes[1].Select(false);

}

else if (two\_shapes[1].CanMoveLeft(form) && two\_shapes[1].selected && !two\_shapes[0].selected)

{

two\_shapes[1].MoveLeft(form);

}

}

public override void MoveRight(Form form)

{

if (two\_shapes[0].CanMoveRight(form) && two\_shapes[0].selected && !two\_shapes[1].selected)

{

two\_shapes[1].Select(true);

two\_shapes[0].MoveRight(form);

two\_shapes[1].MoveRight(form);

two\_shapes[1].Select(false);

}

else if (two\_shapes[1].CanMoveRight(form) && two\_shapes[1].selected && !two\_shapes[0].selected)

{

two\_shapes[1].MoveRight(form);

}

}

public override void Draw(Graphics g)

{

two\_shapes[0].Draw(g);

two\_shapes[1].Draw(g);

g.DrawLine(new Pen(two\_shapes[0].shape\_color, 3), two\_shapes[0].coords, two\_shapes[1].coords);

}

}

public class SavedData

{

public List<string> linesToWrite = new List<string>();

public void Add(string line)

{

linesToWrite.Add(line);

}

}

public class SaverLoader

{

public void Save(SavedData savedData, string way)

{

File.WriteAllLines(way, savedData.linesToWrite);

}

}