Федеральное агентство по образованию

ФГБОУ ВО Уфимский государственный авиационный технический

университет

Кафедра ВМиК

ЛАБОРАТОРНАЯ РАБОТА №8

По дисциплине: «Объектно-Ориентированное Программирование»

«Дерево объектов, подписка»

Выполнил: студент группы ПРО-222

Хусамов А.Р.

Проверил: Котельников В.А.

Уфа 2020

1. **Цель работы:** на основе Л.Р.7 реализовать отображение объектов хранилища в TreeView и реализовать «липкий» объект с помощью паттерна Observer.
2. **Задачи:**

* 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 специальный «липкий» вид объекта: при касании/пересечении которого другие объекты «приклеиваются» к нему и при перемещении «липкого» объекта, вместе с ним перемещаются и «приклеенные» объекты.

1. **Ход** **работы**

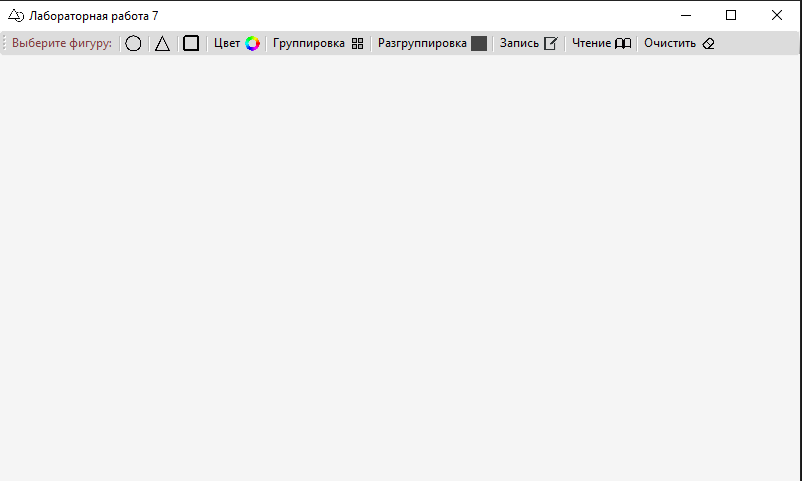
При запуске программы появляется стартовое окно, с полем для рисования и меню для редактирования объектов . (Рис. 1) 

Рис. 1 Стартовое окно

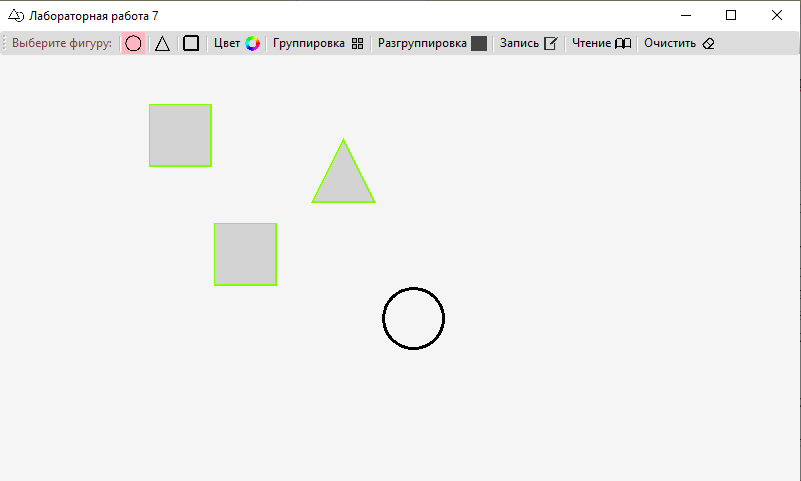
При выделении нескольких объектов и нажатии кнопки «Группировка» происходит объединение объектов в одну группу. С этого момента они будут реагировать как единый объект. (Рис. 2) 

Рис. 2 Группировка фигур

Кнопка «Разгруппировка» разделяет объекты группы, делая их самостоятельными. (Рис. 3)

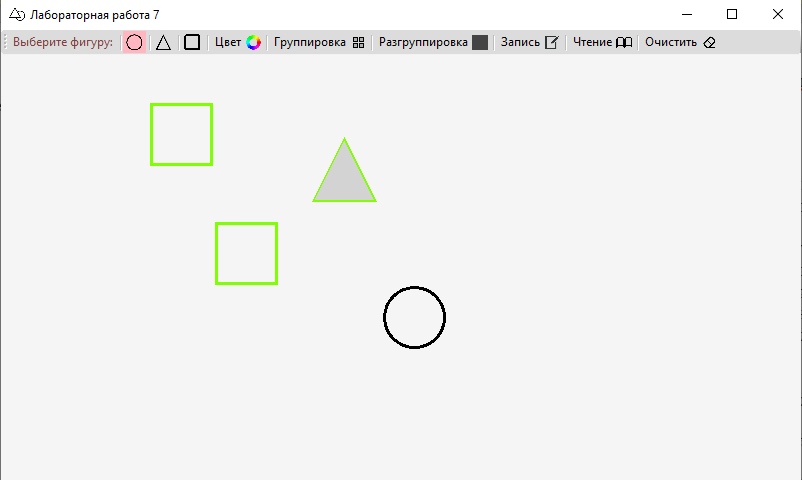


Рис. 3 Разгруппировка фигур

Так же, в группу могут входить другие группы. Одинаковым цветом выделены (для наглядности) объекты разных групп. Мы их объединили в одну группу. (Рис. 4)

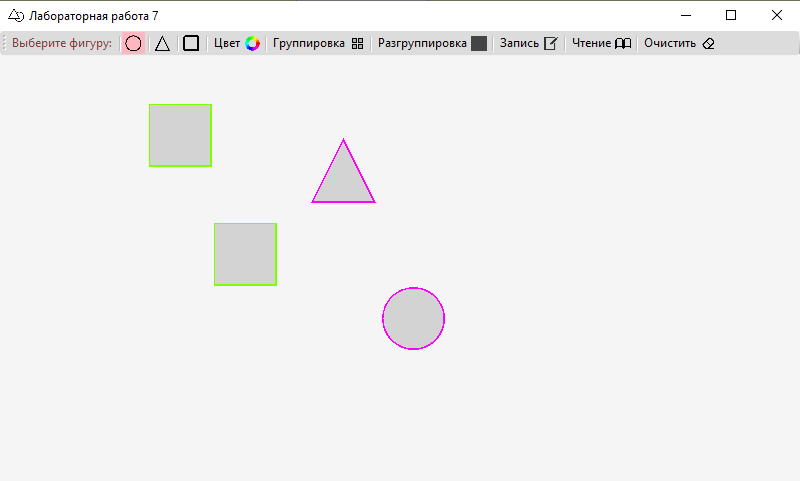


Рис. 4 Группировка групп

Помимо всех вышеперечисленных функций, можно выполнить запись всех объектов в человеко-читаемый текст. (Рис. 5)

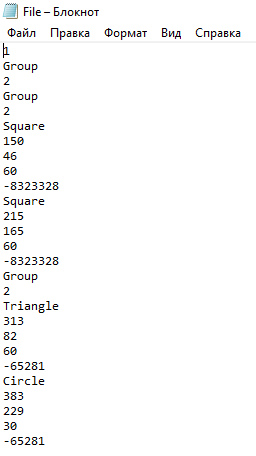


Рис. 5 Фигуры записанные в человеко-читаемый текст

И наоборот, «прочитать» объекты из текста в хранилище и полотно. (рис. 6)

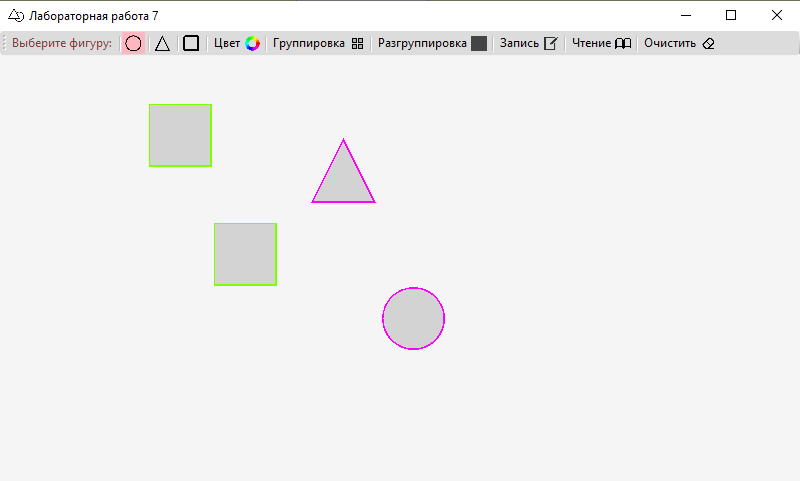


Рис. 6 Чтение из текста и заполнение хранилища и полотна

1. **Вывод:**

В рамках лабораторной работы я разобрался с паттернами Composite и Factory Method, научился группировать/разгруппировывать объекты, а также работать с текстовыми файлами.

Приложение 1. Листинг программы:

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 System.IO;

namespace Tree\_Of\_Objects\_And\_Subscription

{

public partial class Form1 : Form

{

MVC model;

public Form1()

{

InitializeComponent();

model = new MVC();

model.observers += new System.EventHandler(this.UpdateFromMVC);

storage.RegisterObserver(tree);

}

#region Классы фигур

public class Shape // Родительский (класс) фигуры

{

public int x, y;

public int radius = 30, size = 60;

public int X\_min, X\_max, Y\_min, Y\_max;

protected Color color = default\_color;

private bool is\_selected = false;

private bool is\_sticky = false;

public Shape()

{

}

public virtual void SetColor(Color color)

{

this.color = color;

}

public Color GetColor()

{

return color;

}

public void Select(bool is\_selected)

{

this.is\_selected = is\_selected;

}

public bool IsSelected()

{

return is\_selected;

}

public void Stick(bool is\_sticky)

{

this.is\_sticky = is\_sticky;

}

public bool IsSticky()

{

return is\_sticky;

}

public virtual void Draw\_Shape(Pen pen, Brush solidBrush, Panel Canvas\_Panel) { }

public virtual void Move\_x(int x, Panel Canvas\_Panel) { }

public virtual void Move\_y(int y, Panel Canvas\_Panel) { }

public virtual void Change\_Size(int size) { }

public virtual bool Check\_Shape(int x, int y) { return false; }

public virtual string Save() { return ""; }

public virtual void Load(string x, string y, string c, string color) { }

public virtual void Load(ref StreamReader sr, Shape Shape, CreateShape createShape) { }

public virtual void Add\_to\_Group(ref Shape shape) { }

public virtual void Ungroup(ref Storage storage, int c) { }

public virtual void Choice(ref StreamReader sr, ref Shape shape, CreateShape createShape) { }

public virtual int getX\_min() { return 0; }

public virtual int getX\_max() { return 0; }

public virtual int getY\_min() { return 0; }

public virtual int getY\_max() { return 0; }

public virtual string Name() { return ""; }

};

public class CreateShape : Shape

{

public override void Choice(ref StreamReader sr, ref Shape shape, CreateShape createShape)

{

string str = sr.ReadLine();

switch (str)

{ // В зависимости какая фигура выбрана

case "Circle":

shape = new Circle();

shape.Load(sr.ReadLine(), sr.ReadLine(), sr.ReadLine(), sr.ReadLine());

break;

case "Triangle":

shape = new Triangle();

shape.Load(sr.ReadLine(), sr.ReadLine(), sr.ReadLine(), sr.ReadLine());

break;

case "Square":

shape = new Square();

shape.Load(sr.ReadLine(), sr.ReadLine(), sr.ReadLine(), sr.ReadLine());

break;

case "Group":

shape = new Group();

shape.Load(ref sr, shape, createShape);

break;

}

}

}

class Circle : Shape// Класс круга

{

//public int radius = 30;

public Circle() { }

public Circle(int x, int y)

{

this.x = x - radius;

this.y = y - radius;

}

public override void Draw\_Shape(Pen pen, Brush solidBrush, Panel Canvas\_Panel)

{

Canvas\_Panel.CreateGraphics().DrawEllipse(

pen, x, y, radius \* 2, radius \* 2);

Canvas\_Panel.CreateGraphics().FillEllipse(

solidBrush, x, y, radius \* 2, radius \* 2);

}

public override void Move\_x(int x, Panel Canvas\_Panel)

{

int c = this.x + x;

int limit = Canvas\_Panel.ClientSize.Width - (radius \* 2);

Check\_borders(c, x, limit, limit - 2, ref this.x);

}

public override void Move\_y(int y, Panel Canvas\_Panel)

{

int c = this.y + y;

int limit = Canvas\_Panel.ClientSize.Height - (radius \* 2);

Check\_borders(c, y, limit, limit - 2, ref this.y);

}

public override void Change\_Size(int size)

{

radius += size;

}

public override bool Check\_Shape(int x, int y)

{

return ((x - this.x - radius) \* (x - this.x - radius) +

(y - this.y - radius) \* (y - this.y - radius)) <= (radius \* radius);

}

public override string Save()

{

return "Circle" + "\n" + x + "\n" + y + "\n" + radius + "\n" + GetColor().ToArgb().ToString();

}

public override void Load(string x, string y, string radius, string color)

{

this.x = Convert.ToInt32(x);

this.y = Convert.ToInt32(y);

this.radius = Convert.ToInt32(radius);

SetColor(Color.FromArgb(Convert.ToInt32(color)));

}

public override int getX\_min() { return x; }

public override int getX\_max() { return x + 2 \* radius; }

public override int getY\_min() { return y; }

public override int getY\_max() { return y + 2 \* radius; }

public override string Name() { return "Circle"; }

}

class Triangle : Shape // Класс треугольника

{

//public int size = 60;

public Triangle() { }

public Triangle(int x, int y) // Конструктор с параметрами

{

this.x = x - size / 2;

this.y = y - size / 2;

}

public override void Draw\_Shape(Pen pen, Brush solidBrush, Panel Canvas\_Panel)

{

Point[] points = new Point[3]; // Массив точек для прорисовки треугольника

points[0].X = this.x + size / 2; points[0].Y = this.y;

points[1].X = this.x; points[1].Y = this.y + size;

points[2].X = this.x + size; points[2].Y = this.y + size;

Canvas\_Panel.CreateGraphics().DrawPolygon(pen, points);

Canvas\_Panel.CreateGraphics().FillPolygon(solidBrush, points);

}

public override void Move\_x(int x, Panel Canvas\_Panel)

{

int t = this.x + x;

int limit = Canvas\_Panel.ClientSize.Width - size;

Check\_borders(t, x, limit, limit - 2, ref this.x);

}

public override void Move\_y(int y, Panel Canvas\_Panel)

{

int t = this.y + y;

int limit = Canvas\_Panel.ClientSize.Height - size;

Check\_borders(t, y, limit, limit - 2, ref this.y);

}

public override void Change\_Size(int size)

{

this.size += size;

}

public override bool Check\_Shape(int x, int y)

{

Point[] points = new Point[3]; // Массив точек для прорисовки треугольника

points[0].X = this.x + size / 2; points[0].Y = this.y;

points[1].X = this.x; points[1].Y = this.y + size;

points[2].X = this.x + size; points[2].Y = this.y + size;

int a = (points[0].X - x) \* (points[1].Y - points[0].Y) - (points[1].X - points[0].X) \* (points[0].Y - y);

int b = (points[1].X - x) \* (points[2].Y - points[1].Y) - (points[2].X - points[1].X) \* (points[1].Y - y);

int c = (points[2].X - x) \* (points[0].Y - points[2].Y) - (points[0].X - points[2].X) \* (points[2].Y - y);

return ((a > 0 && b > 0 && c > 0) || (a < 0 && b < 0 && c < 0));

}

public override string Save()

{

return "Triangle" + "\n" + x + "\n" + y + "\n" + size + "\n" + GetColor().ToArgb().ToString();

}

public override void Load(string x, string y, string size, string color)

{

this.x = Convert.ToInt32(x);

this.y = Convert.ToInt32(y);

this.size = Convert.ToInt32(size);

SetColor(Color.FromArgb(Convert.ToInt32(color)));

}

public override int getX\_min() { return x; }

public override int getX\_max() { return x + size; }

public override int getY\_min() { return y; }

public override int getY\_max() { return y + size; }

public override string Name() { return "Triangle"; }

}

class Square : Shape // Класс квадрата

{

//public int size = 60;

public Square() { }

public Square(int x, int y)

{

this.x = x - size / 2;

this.y = y - size / 2;

}

public override void Draw\_Shape(Pen pen, Brush solidBrush, Panel Canvas\_Panel)

{

Canvas\_Panel.CreateGraphics().DrawRectangle(pen,

x, y, size, size);

Canvas\_Panel.CreateGraphics().FillRectangle(solidBrush,

x, y, size, size);

}

public override void Move\_x(int x, Panel Canvas\_Panel)

{

int s = this.x + x;

int limit = Canvas\_Panel.ClientSize.Width - size;

Check\_borders(s, x, limit, --limit, ref this.x);

}

public override void Move\_y(int y, Panel Canvas\_Panel)

{

int s = this.y + y;

int limit = Canvas\_Panel.ClientSize.Height - size;

Check\_borders(s, y, limit, --limit, ref this.y);

}

public override void Change\_Size(int size)

{

this.size += size;

}

public override bool Check\_Shape(int x, int y)

{

return (this.x <= x && x <= (this.x + size) &&

this.y <= y && y <= (this.y + size));

}

public override string Save()

{

return "Square" + "\n" + x + "\n" + y + "\n" + size + "\n" + GetColor().ToArgb().ToString();

}

public override void Load(string x, string y, string size, string color)

{

this.x = Convert.ToInt32(x);

this.y = Convert.ToInt32(y);

this.size = Convert.ToInt32(size);

SetColor(Color.FromArgb(Convert.ToInt32(color)));

}

public override int getX\_min() { return x; }

public override int getX\_max() { return x + size; }

public override int getY\_min() { return y; }

public override int getY\_max() { return y + size; }

public override string Name() { return "Square"; }

}

class Group : Shape

{

public int maxcount = 10;

public Shape[] group;

public int count;

public Group()

{ // Выделяем maxcount мест в хранилище

count = 0;

group = new Shape[maxcount];

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

group[i] = null;

}

public override string Save()

{

string str = "Group" + "\n" + count;

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

str += "\n" + group[i].Save();

return str;

}

public override void Load(ref StreamReader sr, Shape shape, CreateShape createShape)

{

int chislo = Convert.ToInt32(sr.ReadLine());

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

{

createShape.Choice(ref sr, ref shape, createShape);

Add\_to\_Group(ref shape);

}

}

public override void Add\_to\_Group(ref Shape shape)

{

if (count >= maxcount)

return;

group[count] = shape;

++count;

}

public override void Ungroup(ref Storage storage, int c)

{

storage.Delete\_object(c);

storage.Sort(count\_cells);

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

{

storage.Add\_object(count\_elements, ref group[i], count\_cells, ref indexin);

}

}

public override void Draw\_Shape(Pen pen, Brush solidBrush, Panel Canvas\_Panel)

{

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

{

pen.Color = group[i].GetColor();

group[i].Draw\_Shape(pen, solidBrush, Canvas\_Panel);

}

}

public void Group\_Borders()

{

X\_min = int.MaxValue; X\_max = 0; Y\_min = int.MaxValue; Y\_max = 0;

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

{

int f = 0;

f = group[i].getX\_min();

if (f < X\_min)

X\_min = f;

f = group[i].getX\_max();

if (f > X\_max)

X\_max = f;

f = group[i].getY\_min();

if (f < Y\_min)

Y\_min = f;

f = group[i].getY\_max();

if (f > Y\_max)

Y\_max = f;

}

}

public override void Move\_x(int x, Panel Canvas\_Panel)

{

Group\_Borders();

if ((X\_min + x) > 0 && (X\_max + x) < Canvas\_Panel.ClientSize.Width)

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

group[i].Move\_x(x, Canvas\_Panel);

}

public override void Move\_y(int y, Panel Canvas\_Panel)

{

Group\_Borders();

if ((Y\_min + y) > 0 && (Y\_max + y) < Canvas\_Panel.ClientSize.Height)

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

group[i].Move\_y(y, Canvas\_Panel);

}

public override void Change\_Size(int size)

{

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

{

group[i].Change\_Size(size);

}

}

public override bool Check\_Shape(int x, int y)

{

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

{

if (group[i].Check\_Shape(x, y))

return true;

}

return false;

}

public override void SetColor(Color color)

{

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

{

group[i].SetColor(color);

}

}

public override string Name() { return "Group"; }

}

#endregion

public interface IObservable

{ // Наблюдаемый объект

void RegisterObserver(IObserver o);

void RemoveObserver(IObserver o);

void NotifyObservers();

}

public interface IObserver

{ // Наблюдатель

void Update(ref TreeView treeView, Storage storage);

}

public class Storage : IObservable

{

public Shape[] objects;

public TreeView treeView;

public List<IObserver> observers;

public Storage(int amount)

{ // Конструктор по умолчанию

objects = new Shape[amount];

observers = new List<IObserver>();

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

objects[i] = null;

}

public void Initialization(int amount)

{ // Выделяем amount мест в хранилище

objects = new Shape[amount];

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

objects[i] = null;

}

public void Initialization\_tree(ref TreeView treeView)

{

this.treeView = treeView;

}

public void Add\_object(int ind, ref Shape new\_object, int count, ref int indexin)

{ // Добавляет ячейку в хранилище

// Если ячейка занята ищем свободное место

while (objects[ind] != null)

{

ind = (ind + 1) % count;

}

objects[ind] = new\_object;

indexin = ind;

Sort(count\_cells);

NotifyObservers();

}

public void Delete\_object(int ind)

{ // Удаляет объект из хранилища

objects[ind] = null;

if (ind > 0)

ind--;

NotifyObservers();

}

public void Sort(int size)

{

Storage storage1 = new Storage(size);

int col = 0;

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

{

if (!Is\_empty(i))

{

storage1.objects[col] = objects[i];

++col;

}

}

Initialization(size);

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

objects[i] = storage1.objects[i];

}

public bool Is\_empty(int ind)

{ // Проверяет занято ли место в хранилище

if (objects[ind] == null)

return true;

else return false;

}

public int Occupied(int size)

{ // Определяет кол-во занятых мест в хранилище

int count\_occupied = 0;

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

if (!Is\_empty(i))

++count\_occupied;

return count\_occupied;

}

public void Increase\_Storage(ref int size)

{ // Увеличивает хранилище в 2 раза

Storage new\_storage = new Storage(size \* 2);

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

new\_storage.objects[i] = objects[i];

for (int i = size; i < (size \* 2) - 1; ++i)

new\_storage.objects[i] = null;

size \*= 2;

Initialization(size);

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

objects[i] = new\_storage.objects[i];

}

public void RegisterObserver(IObserver o)

{

observers.Add(o);

}

public void RemoveObserver(IObserver o)

{

observers.Remove(o);

}

public void NotifyObservers()

{

foreach (IObserver observer in observers)

observer.Update(ref treeView, this);

}

~Storage() { }

};

public class TreeViews : IObserver

{

public TreeViews() { }

public void Update(ref TreeView treeView, Storage storage)

{

treeView.Nodes.Clear();

treeView.Nodes.Add("Фигуры");

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

{

if (!storage.Is\_empty(i))

{

FillNode(treeView.Nodes[0], storage.objects[i]);

}

}

treeView.ExpandAll();

}

public void TreeSelect(ref TreeView treeView, int index) //выбор узла

{ // Выделяем узел

treeView.SelectedNode = treeView.Nodes[0].Nodes[index];

treeView.Focus();

}

public void FillNode(TreeNode treeNode, Shape shape)

{

TreeNode nodes = treeNode.Nodes.Add(shape.Name());

if (shape.Name() == "Group")

{

for (int i = 0; i < (shape as Group).count; ++i)

{

FillNode(nodes, (shape as Group).group[i]);

}

}

}

}

public class StickyObserver: IObserver

{

public StickyObserver() { }

public bool checkCircle(Storage stg, int i, int j)

{

if ((stg.objects[j].x - stg.objects[i].x) \* (stg.objects[j].x - stg.objects[i].x) +

(stg.objects[j].y - stg.objects[i].y) \* (stg.objects[j].y - stg.objects[i].y)

<= (stg.objects[i].radius + stg.objects[j].radius) \* (stg.objects[i].radius + stg.objects[j].radius))

return true;

else return false;

}

public bool checkTriangle(Storage stg, int i, int j)

{

if (stg.objects[i].x + (stg.objects[i].size / 2) >= stg.objects[j].x - (stg.objects[j].size / 2)

&& stg.objects[i].x - (stg.objects[i].size / 2) <= stg.objects[j].x + (stg.objects[j].size / 2)

&& stg.objects[i].y >= stg.objects[j].y - (stg.objects[j].size)

&& stg.objects[i].y <= stg.objects[j].y + (stg.objects[j].size))

return true;

else return false;

}

public bool checkSquare(Storage stg, int i, int j)

{

if (stg.objects[i].x + (stg.objects[i].size / 2) >= stg.objects[j].x - (stg.objects[j].size / 2)

&& stg.objects[i].x - (stg.objects[i].size / 2) <= stg.objects[j].x + (stg.objects[j].size / 2)

&& stg.objects[i].y >= stg.objects[j].y - (stg.objects[j].size)

&& stg.objects[i].y <= stg.objects[j].y + (stg.objects[j].size))

return true;

else return false;

}

public bool FigureCheck(Storage stg, int i, int j, string b, int d)

{

string h;

if (d == 1)

{

h = b;

}

else h = stg.objects[j].Name();

switch (h)

{

case "Circle":

if (checkCircle(stg, i, j))

return true;

break;

case "Triangle":

if (checkTriangle(stg, i, j))

return true;

break;

case "Square":

if (checkSquare(stg, i, j))

return true;

break;

case "Group":

//for (int v = 0; v < (stg.objects[j] as Group).count; ++v)

//{

// string l = (stg.objects[j] as Group).Name();

// if (FigureCheck(stg, i, v, l, 1))

// return true;

//}

(stg.objects[j] as Group).Group\_Borders();

if (stg.objects[i].x <= (stg.objects[j] as Group).X\_max && (stg.objects[i].x + (stg.objects[i].radius \* 2)) >=

(stg.objects[j] as Group).X\_min &&

stg.objects[i].y <= (stg.objects[j] as Group).Y\_max &&

(stg.objects[i].y + (stg.objects[i].radius \* 2)) >= (stg.objects[j] as Group).Y\_min)

return true;

break;

case null:

return false;

}

return false;

}

public void Update(ref TreeView treeView, Storage stg)

{

int p = 0;

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

{

if (!stg.Is\_empty(i))

{

if (stg.objects[i].IsSticky() == true)

{

p = i;

break;

}

}

}

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

{

if (!stg.Is\_empty(i))

{

if (p == i)

{

continue;

}

string f = "";

if (FigureCheck(stg, p, i, f, 0))

{

stg.objects[i].Select(true);

}

}

}

}

//public void Update(ref TreeView treeView, Storage storage)

//{

// int x = 0, y = 0;

// for (int i = 0; i < count\_cells; ++i)

// if (!storage.Is\_empty(i))

// if (storage.objects[i].IsSelected() && storage.objects[i].IsSticky())

// {

// x = storage.objects[i].x;

// y = storage.objects[i].y;

// }

// for (int i = 0; i < count\_cells; ++i)

// if (!storage.Is\_empty(i))

// if (storage.objects[i].Check\_Shape(x, y))

// storage.objects[i].Select(true);

//}

}

public class MVC

{

private string figure;

public System.EventHandler observers;

public void setFigure(string figure)

{

this.figure = figure;

observers.Invoke(this, null);

}

public string getFigure()

{

return figure;

}

}

#region Объявление переменных

TreeViews tree = new TreeViews();

string figure\_now; // Хранит значение нынешней фигуры

static int count\_cells = 5; // Кол-во ячеек в хранилище

static int indexin = 0; // Индекс, в какое место был помещён круг

static int count\_elements = 0; // Кол-во элементов в хранилище

Storage storage = new Storage(count\_cells); // Хранилище объектов

static Color default\_color = Color.Black; // Цвет по умолчанию

Shape shape = new Shape(); // Объект класса Shape

#endregion

#region Проверки

private int Check\_Shape(ref Storage storage, int x, int y, int start)

{

// Проверяет есть ли уже фигура с такими же координатами в хранилище

if (storage.Occupied(count\_cells) != 0)

{

for (int i = start; i < count\_cells; ++i)

{

if (!storage.Is\_empty(i))

{ // Если под i индексом в хранилище есть объект

if (storage.objects[i].Check\_Shape(x, y))

return i;

}

}

}

return -1;

}

static public void Check\_borders(int f, int number, int limit1, int limit2, ref int coord)

{ // Проверка на выход фигуры за границы

if (f > 0 && f < limit1)

coord += number;

else

{

if (f <= 0)

coord = 1;

else

if (f >= limit2)

coord = limit2;

}

}

#endregion

#region Манипуляции с фигурами

private void Move\_y(ref Storage storage, int y)

{ // Функция для перемещения фигур по оси Y

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())// Если объект выделен

{

storage.objects[i].Move\_y(y, Canvas\_Panel);

storage.NotifyObservers();

}

}

private void Move\_x(ref Storage storage, int x)

{ // Функция для перемещения фигур по оси X

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())// Если объект выделен

{

storage.objects[i].Move\_x(x, Canvas\_Panel);

storage.NotifyObservers();

}

}

private void Change\_Size(ref Storage storage, int size)

{ // Увеличивает или уменьшает размер фигур, в зависимости от size

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

if (!storage.Is\_empty(i))// Если под i индексом в хранилище есть объект

if (storage.objects[i].IsSelected())

storage.objects[i].Change\_Size(size);

}

#endregion

#region Обработчики Событий

private void Canvas\_Panel\_MouseDown(object sender, MouseEventArgs e)//Обработчик нажатия на полотно

{

int ind = Check\_Shape(ref storage, e.X, e.Y, 0);

storage.Initialization\_tree(ref TreeView);

if (e.Button == MouseButtons.Left)

{

if (count\_elements == count\_cells)

// Увеличиваем хранилище

storage.Increase\_Storage(ref count\_cells);

switch (figure\_now)//Узнаем какая сейчас выбрана фигура

{

case "Круг":

shape = new Circle(e.X, e.Y);

break;

case "Треугольник":

shape = new Triangle(e.X, e.Y);

break;

case "Квадрат":

shape = new Square(e.X, e.Y);

break;

}

if (ind == -1)//Если место свободно

{

Deselect(ref storage);//Убираем выделение у всех объектов

storage.Add\_object(count\_elements, ref shape, count\_cells, ref indexin);//Добавляем объект в хранилище

shape.Select(true);//Объект выделяем

Redraw\_Shapes(ref storage);//Перерисовываем все объекты

count\_elements++;//Увеличиваем кол-во элементов

}

}

if (e.Button == MouseButtons.Right)

{

if (ind != -1)//Если там есть фигуры

{

if (Control.ModifierKeys == Keys.Control)

{ // Если нажат ctrl, то выделяем несколько объектов

// Вызываем функцию отрисовки фигуры

storage.objects[ind].Select(true);

Draw\_Shape(ref storage, ind);

}

else

{ // Иначе выделяем только один объект

// Снимаем выделение у всех объектов хранилища

Deselect(ref storage);

Draw\_Shape(ref storage, ind);

tree.TreeSelect(ref TreeView, ind);

}

return;

}

}

}

private void Form1\_KeyDown(object sender, KeyEventArgs e)//Обработчик нажатия на клавиши

{

if (e.KeyCode == Keys.Delete) //Если нажили на Delete

Remove\_Selected(ref storage);

if (e.KeyCode == Keys.Add) //Если +

Change\_Size(ref storage, 5);

if (e.KeyCode == Keys.Subtract) //Если -

Change\_Size(ref storage, -5);

if (e.KeyCode == Keys.W)

Move\_y(ref storage, -5);

if (e.KeyCode == Keys.S)

Move\_y(ref storage, 5);

if (e.KeyCode == Keys.A)

Move\_x(ref storage, -5);

if (e.KeyCode == Keys.D)

Move\_x(ref storage, 5);

Redraw\_Shapes(ref storage);

}

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

{

if (ColorDialog.ShowDialog() == DialogResult.Cancel)

return;

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

{

storage.objects[i].SetColor(ColorDialog.Color);

Redraw\_Shapes(ref storage);

}

}

private void Circle\_ToolStripButton\_Click(object sender, EventArgs e)

{

figure\_now = "Круг";

model.setFigure(figure\_now);

}

private void Triangle\_ToolStripButton\_Click(object sender, EventArgs e)

{

figure\_now = "Треугольник";

model.setFigure(figure\_now);

}

private void Square\_ToolStripButton\_Click(object sender, EventArgs e)

{

figure\_now = "Квадрат";

model.setFigure(figure\_now);

}

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

{

Shape group = new Group();

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

{

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

{

group.Add\_to\_Group(ref storage.objects[i]);

storage.Delete\_object(i);

}

}

storage.Add\_object(count\_elements, ref group, count\_cells, ref indexin);

}

private void Ungrouping\_ToolStripButton\_Click(object sender, EventArgs e)

{

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

{

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

{

storage.objects[i].Ungroup(ref storage, i);

return;

}

}

}

bool sticky\_now = false;

private void Stickiness\_ToolStripButton\_Click(object sender, EventArgs e)

{

StickyObserver sticky\_observer = new StickyObserver();

storage.RegisterObserver(sticky\_observer);

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

{

if (!storage.objects[i].IsSticky() && !sticky\_now)

{

storage.objects[i].Stick(true);

sticky\_now = true;

break;

}

else

if (storage.objects[i].IsSticky() && sticky\_now)

{

storage.objects[i].Stick(false);

sticky\_now = false;

}

}

Redraw\_Shapes(ref storage);

}

// создаем каталог для файла

readonly string path = @"C:\Users\User\OneDrive\Рабочий стол\УНИВЕР\2 КУРС\3 СЕМЕСТР\ООП\Лабораторная работа 8\File.txt";

private void Write\_ToolStripButton\_Click(object sender, EventArgs e)

{

using (StreamWriter sw = new StreamWriter(path, false, System.Text.Encoding.Default))

{

sw.WriteLine(storage.Occupied(count\_cells));

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

{

if (!storage.Is\_empty(i))

{

sw.WriteLine(storage.objects[i].Save());

}

}

}

}

private void Read\_ToolStripButton\_Click(object sender, EventArgs e)

{

StreamReader sr = new StreamReader(path, System.Text.Encoding.Default);

{

string str = sr.ReadLine();

int strend = Convert.ToInt32(str);

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

{

Shape shape = new Shape();

CreateShape create = new CreateShape();

create.Choice(ref sr, ref shape, create);

if (count\_elements == count\_cells)

storage.Increase\_Storage(ref count\_cells);

storage.Add\_object(count\_elements, ref shape, count\_cells, ref indexin);

++count\_elements;

}

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

if (!storage.Is\_empty(i))

Draw\_Shape(ref storage, i);

sr.Close();

}

}

private void Clear\_toolStripButton\_Click(object sender, EventArgs e)

{

while (storage.Occupied(count\_cells) != 0)

{

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

{

if (!storage.Is\_empty(i))

{

storage.Delete\_object(i);

}

}

}

sticky\_now = false;

Canvas\_Panel.Refresh();//Обновляем панель

}

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

{

Deselect(ref storage);

Redraw\_Shapes(ref storage);

int g;

if (e.Node.Level != 1)

g = e.Node.Parent.Index;

else

g = e.Node.Index;

storage.objects[g].Select(true);

Draw\_Shape(ref storage, g);

}

#endregion

#region Рисовка и перерисовка фигур

void Draw\_Shape(ref Storage storage, int index)

{

Pen pen = new Pen(default\_color, 3); // Ручка для рисования

SolidBrush solidBrush = new SolidBrush(Color.LightGray); // Цвет для заливки

SolidBrush stickyBrush = new SolidBrush(Color.LightGreen); // Цвет для заливки

SolidBrush defaultBrush = new SolidBrush(Color.Transparent);

if (!storage.Is\_empty(index))

{

pen.Color = storage.objects[index].GetColor();

if (storage.objects[index].IsSelected() && storage.objects[index].IsSticky())

storage.objects[index].Draw\_Shape(pen, stickyBrush, Canvas\_Panel);

else if (storage.objects[index].IsSelected() && !storage.objects[index].IsSticky())

storage.objects[index].Draw\_Shape(pen, solidBrush, Canvas\_Panel);

else

storage.objects[index].Draw\_Shape(pen, defaultBrush, Canvas\_Panel);

}

}

void Redraw\_Shapes(ref Storage storage)//Отрисовка объектов

{

Canvas\_Panel.Refresh();

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

if (!storage.Is\_empty(i))

Draw\_Shape(ref storage, i);

}

#endregion

#region Работа с выделением

void Deselect(ref Storage storage)//Отменяет выделение у всех объектов

{

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

storage.objects[i].Select(false);

}

private void Remove\_Selected(ref Storage storage)

{ // Удаляет выделенные элементы из хранилища

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

if (!storage.Is\_empty(i))

if (storage.objects[i].IsSelected())

{

storage.Delete\_object(i);

storage.Sort(count\_cells);

count\_elements--;

}

}

#endregion

void UpdateFromMVC(object sender, EventArgs e)//Изменение выбранной фигуры и подсветка ее кнопки

{

Color back\_color = Color.Transparent;

Color selected\_back\_color = Color.LightPink;

Circle\_ToolStripButton.BackColor = back\_color;

Triangle\_ToolStripButton.BackColor = back\_color;

Square\_ToolStripButton.BackColor = back\_color;

switch (model.getFigure())

{

case "Круг":

Circle\_ToolStripButton.BackColor = selected\_back\_color;

break;

case "Треугольник":

Triangle\_ToolStripButton.BackColor = selected\_back\_color;

break;

case "Квадрат":

Square\_ToolStripButton.BackColor = selected\_back\_color;

break;

}

}

}

}

Приложение 2. Скриншот коммитов

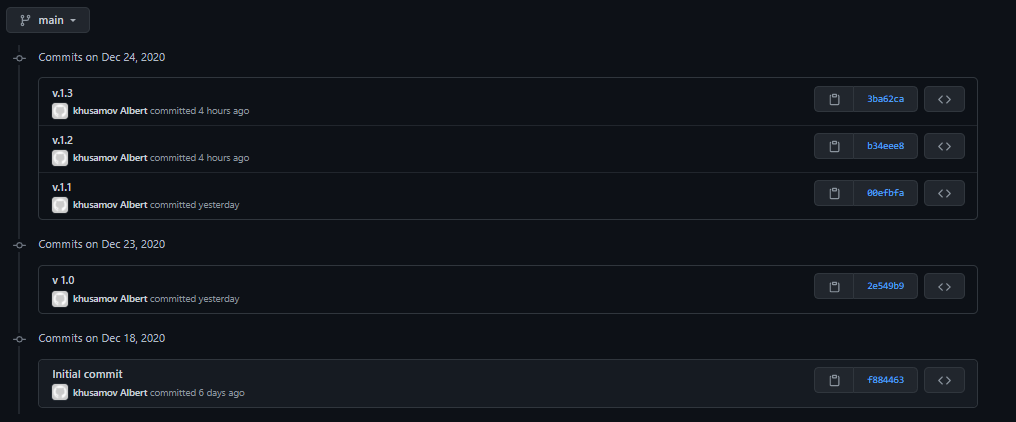


Рис. 7 Коммиты на GitHub

Приложение 3. Ссылка на GitHub

<https://github.com/albrettto/-7.git>