МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
федеральное государственное автономное образовательное учреждение высшего образования
«Санкт–Петербургский государственный университет
аэрокосмического приборостроения»

ФАКУЛЬТЕТ СРЕДНЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ _____

преподаватель		И. Г. Бартасевич
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине: МДК 01.01

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	С326		Э.С. Тигранян
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2025

Contents

Лабораторная работа №1

Цель работы: Изучение основ языка C#

Код программы:

```
static int firstLaboratory()
{
    for (; ; )
    {
        Console.WriteLine("Меню\n1 - Решения первого задания\n2 - Решение второго
задания" +
            "\n3 - Решение третьего задания\n0 - Выход");
        Console.Write("Выберите действие: ");
        try
        {
            int choice = Convert.ToInt32(Console.ReadLine());
            switch (choice)
            {
                case 0: return 0;
                case 1:
                    Console.Write("Введите числа a: ");
                    double a = Convert.ToDouble(Console.ReadLine());
                    Console.Write("Введите числа b: ");
                    double b = Convert.ToDouble(Console.ReadLine());
                    Console.Write("Введите числа h: ");
                    double h = Convert.ToDouble(Console.ReadLine());
                    TableCOUT(a, b, h); break;
                case 2:
                    Console.Write("Введите число: ");
                    int n = Convert.ToInt32(Console.ReadLine());
                    Console.WriteLine(Task2(n)); break;
                case 3:
                    Console.Write("Введите число e < 1: ");
                    double e = Convert.ToDouble(Console.ReadLine());
                    Task3(e);
                    break;
                default:
                    Console.WriteLine("Недоступная опция повторите выбор\n");
break;
            }
```

```
        }
        catch (FormatException)
        {
            Console.WriteLine("Ошибка типа данных попробуйте ещё раз\n");
            continue;
        }
    }
}
static double Task1(double x)
{
    if (((x * x) + 1) + Math.Log(x / 2) == 0 || x / 2 <= 0) { return 0; }
    return (2 / (x * x + 1) + Math.Log(x / 2));
}
static void TableCOUT(double a, double b, double h)
{
    if ((a > b) && (h < 0)) {
        for (; a > b; a += h) {
            if (Task1(a) == 0)
            {
                Console.Write(Math.Round(a, 3) + "\t  |  " + "___" + "\n");
            }
            else
            {
                Console.Write(Math.Round(a, 3) + "\t  |  " + Math.Round(Task1(a),
3) + "\n");
            }}}
    else if ((a < b) && (h > 0)) {
        for (; a < b; a += h) {
            if (Task1(a) == 0)
            {
                Console.Write(Math.Round(a, 3) + "\t  |  " + "___" + "\n");
            }
            else
            {
                Console.Write(Math.Round(a, 3) + "\t  |  " + Math.Round(Task1(a),
3) + "\n");
            }}}
    else { Console.Write("Недопустимый шаг\n"); }
}
```

```
static int Task2(int n)
{
    int result = 0;
    for (int i = 1; i <= (2 * n - 1); i+=2)
    {
        result += Factorial(i);
    }
    return result;
}
static int Factorial(int x)
{
    switch (x) {
        case 0: return 1;
        case 1: return 1;
        default:
            int fact = 1;
            for (int i = 1; i < x; i++)
            {
                fact *= (i + 1);
            }
        return fact;
    }
}
static void Task3(double e) {
    double sum = 0, a = 1;
    Console.WriteLine("Слагаемые: ");

    int n = 1;
    for (;a>e;++n) {
        a = Math.Pow(3, n + 1) / Factorial(n + 2);
        sum += a;
        Console.WriteLine($"n = {n:D2}\ta = {a:F4}  s = {sum:F8}");
    }n -= 1;

    Console.WriteLine(
        "Значение суммы ряда " + Math.Round(sum, 4) +
        "\nКоличество слагаемых: " + n + "\n");
}
```

Лабораторная работа №2

Цель работы: изучение работы массивов на C#

Задание 1. Обработка одномерных массивов8 - Вариант

Z(N)	Расположите в массиве R сначала положительные, а затем отрицательные элементы массива Z

Код программы:

```
static int secondLaboratory()
{
    for (; ; )
    {
        Console.WriteLine("Меню\n1 – Решения первого задания\n2 – Решение второго задания" +
            "\n0 – Выход");
        Console.Write("Выберите действие: ");
        try
        {
            int choice = Convert.ToInt32(Console.ReadLine());
            switch (choice)
            {
                case 0: return 0;
                case 1:
                    Task21(); break;
                case 2:
                    Task22(); break;
                default:
                    Console.WriteLine("Недоступная опция повторите выбор\n");
break;
            }
        }
        catch (FormatException)
        {
            Console.WriteLine("Ошибка типа данных попробуйте ещё раз\n");
            continue;
        }
    }
```

```
}
static int Task21()
{
    Random randomeNum = new Random();
    Console.Write("\nВведите N: ");
    int N = Convert.ToInt32(Console.ReadLine());
    int[] Z = new int[N];
    int[] R = new int[N];
    for (int i = 0; i < N; i++)
    {
        Z[i] = randomeNum.Next(-4, 4);
    }

    Console.WriteLine("Изначальный массив:");
    for (int i = 0; i < N; i++)
    {
        Console.Write(Z[i] + " ");
    }

    ///////
    int k = 0;
    for (int i = 0; i < N; i++)
    {
        if(Z[i] > 0)
        {
            R[k] = Z[i];
            k++;
        }
    }
    for (int i = 0; i < N; i++)
    {
        if (Z[i] == 0)
        {
            R[k] = Z[i];
            k++;
        }
    }
    for (int i = 0; i < N; i++)
    {
```

```
            if (Z[i] < 0)
            {
                R[k] = Z[i];
                k++;
            }
        }
        ///////
        Console.WriteLine("\nОтсортированный массив: ");
        for (int i = 0; i < N; i++)
        {
            Console.Write(R[i] + " ");
        }
        return 0;
        Console.WriteLine("\n");
}
static int Task22()
{
        Random randomeNum = new Random();
        //Транспонировать матрицу и выведите на печать
        //элементы главной диагонали и диагонали,
        //расположенной под главной.Результаты
        //разместите в двух строках.
        int ROWS = 10;
        int COLS = 8;

        int[,] matrix = new int[ROWS, COLS];
        int[,] temp = new int[COLS, ROWS];

        for (int i = 0; i < ROWS; i++)
        {
            for (int j = 0; j < COLS; j++)
            {
                matrix[i, j] = randomeNum.Next(-100, 100);
            }
        }

        Console.WriteLine("Изначальная матрица");
        coutMatrix(matrix, ROWS, COLS);
```

```
    for (int i = 0; i < COLS; i++)
    {
        for (int j = 0; j < ROWS; j++)
        {
            temp[i,j] = matrix[j,i];
        }
    }

    Console.WriteLine("Транспонированная матрицаы\n");
    coutMatrix(temp, COLS, ROWS);

    Console.Write("\nГлавная диагональ: ");
    for (int i = 0; i < COLS; i++)
    {
        Console.Write(temp[i, i] + " ");
    }
    Console.Write("\nДиагональ под главной: ");
    for (int i = 1; i < COLS; i++)
    {
        Console.Write(temp[i, i - 1] + " ");
    }

    return 0;
}
static void coutMatrix(int[,] matrix, int ROWS, int COLS)
{
    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLS; j++)
            Console.Write(matrix[i, j] + "\t");
        Console.WriteLine();
    }
    Console.WriteLine();
}
////////////////////////////////////
```

Результат работы:

```
Введите N: 15
Изначальный массив:
0 2 2 -2 0 1 -2 1 -1 2 3 0 1 -4 -1
Отсортированный массив:
2 2 1 1 2 3 1 0 0 0 -2 -2 -1 -4 -1
```

Задание 2. Обработка двумерных массивов

F (10, 8)	Транспонировать матрицу и выведите на печать элементы главной диагонали и диагонали, расположенной под главной. Результаты разместите в двух строках

```
static int Task22()
{
    //Транспонировать матрицу и выведите на печать
    //элементы главной диагонали и диагонали,
    //расположенной под главной.Результаты
    //разместите в двух строках.

    Random randomeNum = new Random();
    int ROWS = 10;
    int COLS = 8;

    int[,] matrix = new int[ROWS, COLS];
    int[,] temp = new int[COLS, ROWS];

    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLS; j++)
        {
            matrix[i, j] = randomeNum.Next(-100, 100);
        }
    }

    Console.WriteLine("Изначальная матрица");
    coutMatrix(matrix, ROWS, COLS);

    for (int i = 0; i < COLS; i++)
```

```
    {
        for (int j = 0; j < ROWS; j++)
        {
            temp[i, j] = matrix[j, i];
        }
    }

    Console.WriteLine("Транспонированная матрицаы\n");
    coutMatrix(temp, COLS, ROWS);

    Console.Write("\nГлавная диагональ: ");
    for (int i = 0; i < COLS; i++)
    {
        Console.Write(temp[i, i] + " ");
    }
    Console.Write("\nДиагональ под главной: ");
    for (int i = 1; i < COLS; i++)
    {
        Console.Write(temp[i, i - 1] + " ");
    }

    return 0;
}
```

Результат работы:

```
Изначальная матрица
25      -4      -94     -76     72      -95     -57     1
50      -17     71      3       -97     6       -4      -14
-54     -55     -14     0       48      -81     32      68
12      22      59      -78     5       53      -80     6
-80     -97     75      -57     -56     13      71      -19
-51     29      33      20      -45     92      -37     75
61      41      20      52      -52     -66     -1      -53
-10     -17     -19     61      -18     -6      26      49
-85     80      40      45      -4      -97     97      18
28      -30     -37     97      15      20      -40     -45

Транспонированная матрицаы

25      50      -54     12      -80     -51     61      -10     -85     28
-4      -17     -55     22      -97     29      41      -17     80      -30
-94     71      -14     59      75      33      20      -19     40      -37
-76     3       0       -78     -57     20      52      61      45      97
72      -97     48      5       -56     -45     -52     -18     -4      15
-95     6       -81     53      13      92      -66     -6      -97     20
-57     -4      32      -80     71      -37     -1      26      97      -40
1       -14     68      6       -19     75      -53     49      18      -45


Главная диагональ: 25 -17 -14 -78 -56 92 -1 49
Диагональ под главной: -4 71 0 5 13 -37 -53 Меню
```

Лабораторная работа № 2 (Часть 2)

Цель работы: изучение работы строк на C#

«Обработка строк»

Задание 1:

Дана строка, содержащая последовательность слов. Напечатать те слова последовательности, которые отличны от последнего слова и удовлетворяют

следующему свойству: в слове нет повторяющихся букв.

Код программы:

```
static void Task23()
{
    //Дана строка, содержащая последовательность слов. Напечатать те слова
    //последовательности, которые отличны от последнего слова и удовлетворяют
    //следующему свойству: в слове нет повторяющихся букв.

    Console.Write("Введите строку: ");
    string str = Console.ReadLine() ?? "";
    while (string.IsNullOrWhiteSpace(str))
    {
        Console.Write("Пустой ввод, повторите попытку: ");
        str = Console.ReadLine() ?? "";
    }

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

    string currentWord = "";
    for (int i = 0; i < str.Length; i++)
    {
        if (str[i] != ' ')
        {
            currentWord += str[i];
        }
        else
        {
```

```
            if (!string.IsNullOrEmpty(currentWord))
            {
                words.Add(currentWord);
                currentWord = "";
            }
        }
    }

    if (!string.IsNullOrEmpty(currentWord))
    {
        words.Add(currentWord);
    }

    string lastWord = words.Count > 0 ? words[words.Count - 1] : "";

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

    foreach (string word in words)
    {
        if (word != lastWord)
        {
            if (word.Distinct().Count() == word.Length)
            {
                resultWords.Add(word);
            }
        }
    }

    Console.WriteLine("Слова, удовлетворяющие условиям:");
    foreach (string word in resultWords)
    {
        Console.WriteLine(word);
    }
}
```

Результат работы:

```
Введите строку: slov odin slovo DVA slov
 odin  DVA s
```

Задание 2:

Ввести строку, состоящую только из цифр и букв. Распечатать те группы цифр, в
которых цифра 7 встречается не более двух раз. Ввести строку, состоящую только из цифр и букв. Распечатать те группы цифр, в
которых цифра 7 встречается не более двух раз.

```
static void Task24()
{
    //Ввести строку, состоящую только из цифр и букв.Распечатать те группы цифр,
в
    //которых цифра 7 встречается не более двух раз.
    // Ввод строки

    string input;
    while (true)
    {
        Console.WriteLine("Введите строку, состоящую из цифр и букв:");
        input = Console.ReadLine() ?? "";

        for (int i = 0; i < input.Length; i++)
        {
            if (!char.IsLetterOrDigit(input[i]))
            {
                Console.WriteLine("В строке есть символ не являющийся цифрой или
буквой\n\n");
                input = "";
                continue;
            }
        }
        break;
    }

    int SevenNumCounter = 0;
    string temp = "";
    bool isDigitCol = false;
```

```
    for (int i = 0; i < input.Length; i++)
    {
        if (char.IsDigit(input[i])) {
            temp += input[i];
            isDigitCol = true;
            if (input[i] == '7')
            {
                SevenNumCounter += 1;
            }
            if (i == input.Length - 1) {
                if (SevenNumCounter < 3)
                {
                    Console.Write(temp + " ");
                }
            }
        }
        else if (isDigitCol)
        {
            if (SevenNumCounter < 3)
            {
                Console.Write(temp + " ");
            }
            SevenNumCounter = 0;
            temp = "";
            isDigitCol = false;
        }
    }
}
```

Результат:

```
Введите строку, состоящую из цифр и букв:
slovo33227dvaslovo7776542sss77321
Группы цифр не содержащие больше двух 7:
33227 77321
```

Технология WPF. Изучение разметки XAML.

Требования:

Родительский класс:

GeometricObject

Поля : int X, int Y, string color

Конструкторы:

Без параметров (X=0, Y =0, color = «Чёрный»)

С параметрами (X, Y, color)

Методы: Print(string message)

Move (int dx, int dy)

Класс Square (Родитель - GeometricObject)

Поле : side=1 (по умолчанию)

Без параметров (X=0, Y =0, side =1, color = «Чёрный»)

С параметрами (X, Y, side, color)

Переопределить метод Print

Класс Triangle (Родитель - GeometricObject)
Дополнительные поля:
a,b,c – стороны, XB,YB, XC,YC
Координаты X,Y из родительского класса считать координатами вершины A треугольника
XB,YB, XC,YC – координаты вершин B и C соответственно (требуется вычислить!)

Конструктор с параметрами a,b,c, color, X,Y
Метод isExist – возвращает True, если треугольник со сторонами a,b,c существует

void FindCoordinates()

String Print(string message)

17.03.2025

Класс GeometricObject:

```
namespace WPF_Geometric
{
    public class GeometricObject
    {
        public int x { get; private set; }
        public int y { get; private set; }
        public string color { get; private set; }
        public Brush brush { get; set; }
        public Color col { get; set; }
        public GeometricObject()
        {
            x = 0;
            y = 0;
            //col = new Color();
            col = Colors.Red;
            brush = new SolidColorBrush(col);
        }

        public GeometricObject(int x, int y, Color col)
        {
            this.x = x;
            this.y = y;
            this.col = col;
            brush = new SolidColorBrush(col);
        }
```

```
        public virtual string Print(string message)
        {
            return $"{message} x = {x}, y = {y}";
        }

        public virtual void Draw(Canvas canvas)
        {
            Ellipse point = new Ellipse
            {
                Width = 4, // Ширина окружности
                Height = 4, // Высота окружности
                Fill = brush
            };

            // Устанавливаем позицию окружности
            Canvas.SetLeft(point, x - 2); // Центрируем окружность по координатам
x
            Canvas.SetTop(point, y - 2); // Центрируем окружность по координатам
y

            // Добавляем точку на Canvas
            canvas.Children.Add(point);
        }
    }


     }
```

Класс квадрат:

```
namespace WPF_Geometric
{
    internal class Square:GeometricObject
    {
        private int side;

        public Square() :base()
        {
            side = 1;
            col = Colors.Yellow;
            brush = new SolidColorBrush(col);
        }

        public Square(int x, int y, int side, Color col )
            : base(x, y, col)
        {
            this.side = side;
            brush = new SolidColorBrush(col);
        }

        public override string Print(string message)
        {
            return base.Print(message) + $" a = {side}";
        }

        public override void Draw(Canvas canvas)
        {
```

```
            Rectangle square = new Rectangle
            {
                Width = side, // Ширина
                Height = side, // Высота
                Stroke = brush, //Brushes.Green,
                Fill = brush //Brushes.Red // Цвет заливки
            };

            Canvas.SetLeft(square, x );
            Canvas.SetTop(square, y );

            // Добавляем square на Canvas
            canvas.Children.Add(square);
        }
    }
}
```

Класс треугольник:

```
namespace WPF_Geometric
{
    internal class Triangle : GeometricObject
    {
        public double a { get; private set; }
        public double b { get; private set; }
        public double c { get; private set; }

        public System.Windows.Point A { get; set; }
        public System.Windows.Point B { get; set; }
        public System.Windows.Point C { get; set; }

        public Triangle() : base()
        {
            a = 1; b = 1; c = 1;
            A = new System.Windows.Point(x, y);
            B = new System.Windows.Point(x + c, y);
            C = new System.Windows.Point(x + c / 2, y - Math.Sqrt(b * b - (c / 2)
* (c / 2)));
        }

        public Triangle(int x, int y, double a, double b, double c, Color col) :
base(x, y, col)
        {
            if (!(a + b > c && a + c > b && b + c > a))
            {
                throw new Exception("Треугольник с такими сторонами не
существует");
            }

            this.a = a;
            this.b = b;
            this.c = c;

            A = new System.Windows.Point(x, y);
            B = new System.Windows.Point(x + c, y);
            C = new System.Windows.Point(x + (b * b - a * a + c * c) / (2 * c), y
```

```
                + //determine up or down
                Math.Sqrt(b * b - (x + (b * b - a * a + c * c) / (2 * c) - x) *
                (x + (b * b - a * a + c * c) / (2 * c) - x)));
        }

        public override void Draw(Canvas canvas)
        {
            Polygon triangle = new Polygon
            {
                Stroke = brush,
                Fill = brush,
                StrokeThickness = 2
            };

            PointCollection points = new PointCollection
            {
                A,
                B,
                C
            };

            triangle.Points = points;

            canvas.Children.Add(triangle);
        }
    }


     }
```

Класс моей фигуры:

```
namespace WPF_Geometric
{
    internal class MyPicture : Triangle
    {
        public double n { get; private set; }

        public MyPicture() : base() {
            n = 4;
        }

        public MyPicture(int x, int y, double a, double b, double c, int n, Color
col) : base(x,y,a,b,c,col)
        {

            y = Convert.ToInt32(C.Y);
            this.n = n;

            A = new System.Windows.Point(x, y);
            B = new System.Windows.Point(x + c, y);
            C = new System.Windows.Point(x + (b * b - a * a + c * c) / (2 * c), y
                + //determine up or down
                Math.Sqrt(b * b - (x + (b * b - a * a + c * c) / (2 * c) - x) *
                (x + (b * b - a * a + c * c) / (2 * c) - x)));
```

```
        }
        public void Rotate(double angle)
        {
            RotateTransform rotateTransform = new RotateTransform(angle, x, y);
            A = rotateTransform.Transform(A);
            B = rotateTransform.Transform(B);
            C = rotateTransform.Transform(C);
        }

        public SolidColorBrush GetRandomBrush()
        {
            Random random = new Random();

            byte r = (byte)random.Next(256);
            byte g = (byte)random.Next(256);
            byte b = (byte)random.Next(256);

            return new SolidColorBrush(Color.FromRgb(r, g, b));
        }

        public async void Draw(Canvas canvas, double angle, int delay)
        {
            DispatcherTimer timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(10000);
            for (int j = 0; j < 360/(4*angle); j++)
            {
                brush = GetRandomBrush();
                Rotate(angle);
                for (int i = 0; i < n; i++)
                {
                    Polygon triangle = new Polygon
                    {
                        Stroke = brush,
                        Fill = null,
                        StrokeThickness = 2
                    };

                    PointCollection points = new PointCollection
                    {
                        A,
                        B,
                        C
                    };

                    triangle.Points = points;
                    canvas.Children.Add(triangle);
                    Rotate(360 / n);
                }
                await Task.Delay(delay);
                //canvas.Children.Clear();
            }
        }
    }
}
```

Класс фигуры, наследующей квадрат:

```
namespace WPF_Geometric
{
    internal class MySquare : Square
    {
        public MySquare() : base() { }
        int side;
        public MySquare(int x, int y, int side, Color col)
            : base(x, y, side, col)
        {
            this.side = side;
        }

        public override void Draw(Canvas canvas)
        {
            // Основной квадрат
            base.Draw(canvas);

            // Параметры для повернутого квадрата
            double rotatedSide = side * Math.Sqrt(2) / 2; // Сторона повернутого
квадрата
            double offset = (side - rotatedSide) / 2;

            // Квадрат повернутый на 45 градусов
            Rectangle rotatedSquare = new Rectangle
            {
                Width = rotatedSide,
                Height = rotatedSide,
                Stroke = Brushes.Black,
                Fill = Brushes.Transparent,
                RenderTransform = new RotateTransform(45, rotatedSide / 2,
rotatedSide / 2)
            };

            Canvas.SetLeft(rotatedSquare, x + offset);
            Canvas.SetTop(rotatedSquare, y + offset);
            canvas.Children.Add(rotatedSquare);

            // Круг
            double circleDiameter = rotatedSide;
            Ellipse circle = new Ellipse
            {
                Width = circleDiameter,
                Height = circleDiameter,
                Stroke = Brushes.Red,
                Fill = Brushes.Transparent
            };

            Canvas.SetLeft(circle, x + offset);
            Canvas.SetTop(circle, y + offset);
            canvas.Children.Add(circle);
        }

        public override string Print(string message)
        {
            return base.Print(message);
        }
    }
```

```
    }
```

Класс фрактала:

```
namespace WPF_Geometric
{
    // Перечисление типов фракталов
    public enum FractileType
    {
        KochSnowflake,    // Снежинка Коха
        SierpinskiTriangle, // Треугольник Серпинского
        SquareFractal     // Квадратный фрактал
    }
    internal class Fractal : GeometricObject
    {
        public FractileType fractileType { get; set; } // Тип фрактала
        public int Depth { get; set; } = 4;            // Глубина фрактала

        public Fractal() : base()
        {
            col = Colors.Yellow;
            brush = new SolidColorBrush(col);
            fractileType = FractileType.KochSnowflake;
        }
        public Fractal(int x, int y, Color col, FractileType fractileType, int
depth = 5)
            : base(x, y, col)
        {
            this.fractileType = fractileType;
            brush = new SolidColorBrush(col);
            Depth = depth;
        }
        public override void Draw(Canvas canvas)
        {
            switch (fractileType)
            {
                case FractileType.KochSnowflake:
                    DrawKochSnowflake(canvas);
                    break;
                case FractileType.SierpinskiTriangle:
                    DrawSierpinskiTriangle(canvas);
                    break;
                case FractileType.SquareFractal:
                    DrawSquareFractal(canvas);
                    break;
            }
        }
        // Метод для отрисовки снежинки Коха
        private void DrawKochSnowflake(Canvas canvas)
        {
            // Размер стороны треугольника
            double size = 200;
            // Вычисление координат вершин равностороннего треугольника
            System.Windows.Point p1 = new System.Windows.Point(x, y + size *
Math.Sqrt(3) / 2);
```

```
            System.Windows.Point p2 = new System.Windows.Point(x + size, y + size
* Math.Sqrt(3) / 2);
            System.Windows.Point p3 = new System.Windows.Point(x + size / 2, y);

            // Отрисовка трех сторон снежинки Коха с рекурсией
            DrawKochLine(canvas, p1, p2, Depth);
            DrawKochLine(canvas, p2, p3, Depth);
            DrawKochLine(canvas, p3, p1, Depth);
        }
        // Рекурсивный метод для отрисовки линии Коха
        private void DrawKochLine(Canvas canvas, System.Windows.Point start,
System.Windows.Point end, int depth)
        {
            // Базовый случай рекурсии - отрисовка прямой линии
            if (depth == 0)
            {
                // Создание объекта линии
                Line line = new Line
                {
                    X1 = start.X,
                    Y1 = start.Y,
                    X2 = end.X,
                    Y2 = end.Y,
                    Stroke = brush,
                    StrokeThickness = 1
                };
                // Добавление линии на холст
                canvas.Children.Add(line);
                return;
            }
            // Вычисление промежуточных точек для кривой Коха
            System.Windows.Point p1 = start;
            System.Windows.Point p2 = new System.Windows.Point((2 * start.X +
end.X) / 3, (2 * start.Y + end.Y) / 3);
            System.Windows.Point p3 = new System.Windows.Point(
                (start.X + end.X) / 2 - (end.Y - start.Y) * Math.Sqrt(3) / 6,
                (start.Y + end.Y) / 2 + (end.X - start.X) * Math.Sqrt(3) / 6);
            System.Windows.Point p4 = new System.Windows.Point((start.X + 2 *
end.X) / 3, (start.Y + 2 * end.Y) / 3);
            System.Windows.Point p5 = end;

            // Рекурсивная отрисовка четырех сегментов кривой Коха
            DrawKochLine(canvas, p1, p2, depth - 1);
            DrawKochLine(canvas, p2, p3, depth - 1);
            DrawKochLine(canvas, p3, p4, depth - 1);
            DrawKochLine(canvas, p4, p5, depth - 1);
        }
        // Метод для отрисовки треугольника Серпинского
        private void DrawSierpinskiTriangle(Canvas canvas)
        {
            // Размер стороны треугольника
            double size = 200;
            // Вычисление координат вершин треугольника
            System.Windows.Point top = new System.Windows.Point(x + size / 2, y);
            System.Windows.Point left = new System.Windows.Point(x, y + size);
            System.Windows.Point right = new System.Windows.Point(x + size, y +
size);
            // Вызов рекурсивного метода отрисовки
            DrawSierpinski(canvas, top, left, right, Depth);
```

```
        }
        // Рекурсивный метод для отрисовки треугольника Серпинского
        private void DrawSierpinski(Canvas canvas, System.Windows.Point a,
System.Windows.Point b, System.Windows.Point c, int depth)
        {
            if (depth == 0)
            {
                Polygon triangle = new Polygon
                {
                    Points = new PointCollection { a, b, c }, // Установка вершин
                    Stroke = brush,
                    Fill = brush,
                    StrokeThickness = 1
                };
                // Добавление треугольника на холст
                canvas.Children.Add(triangle);
                return;
            }
            // Вычисление середин сторон треугольника
            System.Windows.Point ab = new System.Windows.Point((a.X + b.X) / 2,
(a.Y + b.Y) / 2);
            System.Windows.Point bc = new System.Windows.Point((b.X + c.X) / 2,
(b.Y + c.Y) / 2);
            System.Windows.Point ca = new System.Windows.Point((c.X + a.X) / 2,
(c.Y + a.Y) / 2);
            // Рекурсивная отрисовка трех подтреугольников
            DrawSierpinski(canvas, a, ab, ca, depth - 1);
            DrawSierpinski(canvas, ab, b, bc, depth - 1);
            DrawSierpinski(canvas, ca, bc, c, depth - 1);
        }

        // Метод для отрисовки квадратного фрактала
        private void DrawSquareFractal(Canvas canvas)
        {
            // Вызов рекурсивного метода отрисовки вложенных квадратов
            DrawNestedSquares(canvas, new System.Windows.Point(x + 100, y + 100),
150, Depth);
        }

        // Рекурсивный метод для отрисовки вложенных квадратов
        private void DrawNestedSquares(Canvas canvas, System.Windows.Point
center, double size, int depth)
        {
            // Условие выхода из рекурсии
            if (depth <= 0) return;

            // Создание объекта прямоугольника
            Rectangle square = new Rectangle
            {
                Width = size,
                Height = size,
                Stroke = brush,
                StrokeThickness = 1,
                Fill = Brushes.Transparent
            };
            // Установка позиции прямоугольника на холсте
            Canvas.SetLeft(square, center.X - size / 2);
            Canvas.SetTop(square, center.Y - size / 2);
            // Добавление прямоугольника на холст
```

```
            canvas.Children.Add(square);

            // Рекурсивная отрисовка меньших квадратов по углам
            double newSize = size / 2.5;
            DrawNestedSquares(canvas, new System.Windows.Point(center.X - size /
2, center.Y - size / 2), newSize, depth - 1); // Левый верхний квадрат
            DrawNestedSquares(canvas, new System.Windows.Point(center.X + size /
2, center.Y - size / 2), newSize, depth - 1); // Правый верхний квадрат
            DrawNestedSquares(canvas, new System.Windows.Point(center.X - size /
2, center.Y + size / 2), newSize, depth - 1); // Левый нижний квадрат
            DrawNestedSquares(canvas, new System.Windows.Point(center.X + size /
2, center.Y + size / 2), newSize, depth - 1); // Правый нижний квадрат
        }
    }


     }
```

Главное окно:

```
<Window x:Class="WPF_Geometric.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPF_Geometric"
        mc:Ignorable="d"
        Title="Геометрические объекты" Height="695" Width="1240">

    <Window.Resources>
        <Style x:Key="NumberTextBox" TargetType="TextBox">
            <Setter Property="HorizontalAlignment" Value="Left"/>
            <Setter Property="FontSize" Value="25"/>
            <Setter Property="Width" Value="158"/>
            <Setter Property="Height" Value="47"/>
            <Setter Property="VerticalAlignment" Value="Center"/>
            <Style.Triggers>
                <Trigger Property="Text" Value="">
                    <Setter Property="Foreground" Value="Gray"/>
                    <Setter Property="FontStyle" Value="Italic"/>
                </Trigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition />
```

```
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <Grid Margin="130,35,130,34" Grid.ColumnSpan="6" Grid.Row="4"
Grid.RowSpan="2">
            <Grid.RowDefinitions>
                <RowDefinition Height="64*" />
                <RowDefinition Height="64*" />
                <RowDefinition Height="151*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
                <ColumnDefinition />
            </Grid.ColumnDefinitions>

            <Button x:Name="buttonSquare"
                    Grid.Column="1" Grid.Row="2" Content="Квадрат" FontSize="20"
                    Click="buttonSquare_Click" Margin="6,20,7,21"/>
            <Button x:Name="buttonTriangle"
                    Grid.Column="2" Grid.Row="2" Content="Треугольник"
FontSize="20"
                    Click="buttonTriangle_Click" Margin="7,20,6,21"/>
            <Button x:Name="buttonPoint"
                    Grid.Column="0" Grid.Row="2" Content="Точка" FontSize="20"
                    Click="buttonPoint_Click" Margin="7,20,6,21" />
            <Button x:Name="buttonMyFigure"
                    Grid.Column="3" Grid.Row="2" Content="Моя фигура"
FontSize="20"
                    Click="buttonMyPicture_Click" Margin="6,20,7,21"/>
            <Button x:Name="buttonMySquare"
                    Grid.Column="4" Grid.Row="2" Content="Мой квадрат"
FontSize="20"
                    Click="buttonMySquare_Click" Margin="6,20,7,21"/>
            <Button x:Name="ButtonFractal"
                    Grid.Column="5" Grid.Row="2" Content="Фрактал" FontSize="20"
                    Click="buttonFractal_Click" Margin="6,20,7,21"/>
            <Button x:Name="ButtonClear"
                    Grid.Column="6" Grid.Row="2" Content="Очистить" FontSize="20"
                    Click="buttonClear_Click" Margin="6,20,7,21"/>

            <!-- Parameters -->

            <Label Content="Ввод X" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="0"></Label>
            <TextBox HorizontalAlignment="Left" x:Name="TextBoxX" FontSize="25"
Grid.Row="1"
                     Width="158" PreviewTextInput="NumberValidationTextBox"
                     Grid.Column="0" Height="47" VerticalAlignment="Center"/>

            <Label Content="Ввод Y" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="1"></Label>
```

```
            <TextBox HorizontalAlignment="Left" x:Name="TextBoxY" FontSize="25"
Grid.Row="1"
                     Width="158" PreviewTextInput="NumberValidationTextBox"
                     Grid.Column="1" Height="47" VerticalAlignment="Center"/>

            <Label Content="Ввод A" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="2"></Label>
            <TextBox HorizontalAlignment="Left" x:Name="TextBoxA" FontSize="25"
Grid.Row="1"
                     Width="158" PreviewTextInput="NumberValidationTextBox"
                     Grid.Column="2" Height="47" VerticalAlignment="Center"/>

            <Label Content="Ввод B" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="3"></Label>
            <TextBox HorizontalAlignment="Left" x:Name="TextBoxB" FontSize="25"
Grid.Row="1"
                     Width="158" PreviewTextInput="NumberValidationTextBox"
                     Grid.Column="3" Height="47" VerticalAlignment="Center"/>

            <Label Content="Ввод C" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="4"></Label>
            <TextBox HorizontalAlignment="Left" x:Name="TextBoxC" FontSize="25"
Grid.Row="1"
                     Width="158" PreviewTextInput="NumberValidationTextBox"
                     Grid.Column="4" Height="47" VerticalAlignment="Center"/>


            <Label Content="Выбор фрактала" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="5"></Label>
            <ComboBox x:Name="FractalTypeComboBox" Grid.Column="5" Grid.Row="1"
></ComboBox>


            <Label Content="Цвет" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="20" Grid.Column="6"></Label>
            <Button x:Name="ColorPickerButton" Width="120" Height="30"
Click="ColorPickerButton_Click" Grid.Column="6" Grid.Row="1" >
                <Button.Content>
                    <StackPanel Orientation="Horizontal">
                        <Rectangle Width="20" Height="20" Fill="White"
Stroke="Black" StrokeThickness="1" Margin="0,0,5,0"/>
                        <TextBlock Text="Цвет" FontSize="22"
VerticalAlignment="Center" HorizontalAlignment="Right"/>
                    </StackPanel>
                </Button.Content>
            </Button>


        </Grid>

        <Label x:Name="labelInfo" Content="Label" HorizontalAlignment="Left"
Margin="130,31,0,0" VerticalAlignment="Top" Height="26" Width="687"
Grid.ColumnSpan="4"/>
        <Canvas Name="myCanvas" Width="400" Height="400" Background="White"
            Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="4" Grid.RowSpan="3"
            HorizontalAlignment="Center" VerticalAlignment="Center"/>

    </Grid>
</Window>
```

cs code:

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;

namespace WPF_Geometric
{
    public partial class MainWindow : Window
    {
        private Color _selectedColor = Colors.Blue; // Текущий выбранный цвет
по умолчанию
        private SolidColorBrush _selectedColorBrush; // Кисть с текущим
цветом
        private bool _myPictureMode = false; // Флаг режима рисования
пользовательской фигуры
        public MainWindow()
        {
            InitializeComponent();
            _selectedColorBrush = new SolidColorBrush(_selectedColor); //
Создание кисти с выбранным цветом
            Loaded += MainWindow_Loaded;                    // Событие
загрузки окна
```

```
        FractalTypeComboBox.ItemsSource = new List<string>         //
Инициализация ComboBox типами фракталов
        {
            "Koch Snowflake",
            "Sierpinski Triangle",
            "Square Fractal"
        };
    }

    // Метод для получения выбранного типа фрактала
    public FractileType GetSelectedFractalType()
    {
        return (FractileType)FractalTypeComboBox.SelectedIndex;
    }

    // Обработчик события загрузки окна
    private void MainWindow_Loaded(object sender, RoutedEventArgs e)
    {
        DrawCoordinatePlane();                  // Отрисовка координатной
плоскости
        myCanvas.MouseDown += MyCanvas_MouseDown; // Событие клика
мыши по холсту
    }

    // Метод для отрисовки координатной плоскости
    private void DrawCoordinatePlane()
```

```
    {
        // Получение размеров холста
        int width = (int)myCanvas.ActualWidth;
        int height = (int)myCanvas.ActualHeight;
        int step = 20; // Шаг сетки

        // Установка преобразований для центрального расположения
координат
        myCanvas.RenderTransform = new TransformGroup
        {
            Children = new TransformCollection
            {
                new ScaleTransform(1, -1), // Инверсия оси Y
                new TranslateTransform(width / 2, height / 2) // Смещение в
центр
            }
        };

        // Отрисовка вертикальных линий сетки
        for (int i = -width / 2; i <= width / 2; i += step)
        {
            DrawLine(i, -height / 2, i, height / 2, Brushes.LightGray);
        }

        // Отрисовка горизонтальных линий сетки
        for (int i = -height / 2; i <= height / 2; i += step)
        {
```

```
            DrawLine(-width / 2, i, width / 2, i, Brushes.LightGray);
        }

        // Отрисовка основных осей координат
        DrawLine(-width / 2, 0, width / 2, 0, Brushes.Black); // Ось X
        DrawLine(0, -height / 2, 0, height / 2, Brushes.Black); // Ось Y
    }

    // Метод для отрисовки линии на холсте
    private void DrawLine(double x1, double y1, double x2, double y2, Brush
color)
    {
        Line line = new Line
        {
            X1 = x1,
            Y1 = y1,
            X2 = x2,
            Y2 = y2,
            Stroke = color,
            StrokeThickness = 1
        };
        myCanvas.Children.Add(line);
    }

    // Свойство для выбранного цвета с логикой обновления
    public Color SelectedColor
    {
```

```
            get => _selectedColor;
            set
            {
                _selectedColor = value;
                _selectedColorBrush = new SolidColorBrush(value);
            }
        }

        // Свойство для кисти с выбранным цветом
        public SolidColorBrush SelectedColorBrush
        {
            get => _selectedColorBrush;
            set => _selectedColorBrush = value;
        }

        // Метод валидации ввода чисел в текстовые поля
        private void NumberValidationTextBox(object sender,
TextCompositionEventArgs e)
        {
            TextBox textBox = (TextBox)sender;
            string newText = textBox.Text.Insert(textBox.SelectionStart, e.Text);
            // Проверка ввода на цифру или минус
            bool isDigit = char.IsDigit(e.Text, 0);
            bool isMinus = e.Text == "-";
            // Блокировка недопустимых символов
            if (!isDigit && !isMinus)
            {
```

```
            e.Handled = true;
            return;
        }
        // Дополнительная проверка для минуса
        if (isMinus)
        {
            // Минус разрешен только в начале и только один
            if (textBox.SelectionStart != 0 || textBox.Text.Contains("-"))
            {
                e.Handled = true;
            }
        }
    }

    // Генератор случайных чисел
    Random rnd = new Random();

    ////////////////////////////////////////////////////////////////////////////
    // События нажатия кнопок
    private void buttonPoint_Click(object sender, RoutedEventArgs e)
    {
        _myPictureMode = false; // Отключение режима пользовательской
фигуры

        // Парсинг координат или генерация случайных значений
        int x = int.TryParse(TextBoxX.Text, out int xVal) ? xVal : rnd.Next(-
200, 200);
```

```
            int y = int.TryParse(TextBoxY.Text, out int yVal) ? yVal : rnd.Next(-
200, 200);

            // Создание и отрисовка точки
            GeometricObject point = new GeometricObject(x, y, SelectedColor);
            point.Draw(myCanvas);
            labelInfo.Content = point.Print($"Я - точка!");
        }
        private void buttonSquare_Click(object sender, RoutedEventArgs e)
        {
            _myPictureMode = false; // Отключение режима пользовательской
фигуры

            // Парсинг параметров или генерация случайных значений
            int x = int.TryParse(TextBoxX.Text, out int xVal) ? xVal : rnd.Next(-
200, 200);
            int y = int.TryParse(TextBoxY.Text, out int yVal) ? yVal : rnd.Next(-
200, 200);
            int a = int.TryParse(TextBoxA.Text, out int aVal) ? aVal : rnd.Next(10,
150);

            // Создание и отрисовка квадрата
            Square square = new Square(x, y, a, SelectedColor);
            square.Draw(myCanvas);
            labelInfo.Content = square.Print($"Я квадрат!");
        }
        private void buttonTriangle_Click(object sender, RoutedEventArgs e)
```

```
    {
        _myPictureMode = false; // Отключение режима пользовательской фигуры

        // Парсинг параметров или установка значений по умолчанию
        int x = int.TryParse(TextBoxX.Text, out int xVal) ? xVal : rnd.Next(-200, 200);
        int y = int.TryParse(TextBoxY.Text, out int yVal) ? yVal : rnd.Next(-200, 200);
        int a = int.TryParse(TextBoxA.Text, out int aVal) ? aVal : 50;
        int b = int.TryParse(TextBoxB.Text, out int bVal) ? bVal : 100;
        int c = int.TryParse(TextBoxC.Text, out int cVal) ? cVal : 125;

        // Создание и отрисовка треугольника
        Triangle triangle = new Triangle(x, y, a, b, c, SelectedColor);
        triangle.Draw(myCanvas);
        labelInfo.Content = triangle.Print($"Я треугольник!");
    }
    ////////////
    private void buttonMyPicture_Click(object sender, RoutedEventArgs e)
    {
        // Переключение состояния режима
        _myPictureMode = !_myPictureMode;

        if (_myPictureMode)
        {
            // Обновление интерфейса для активного режима
```

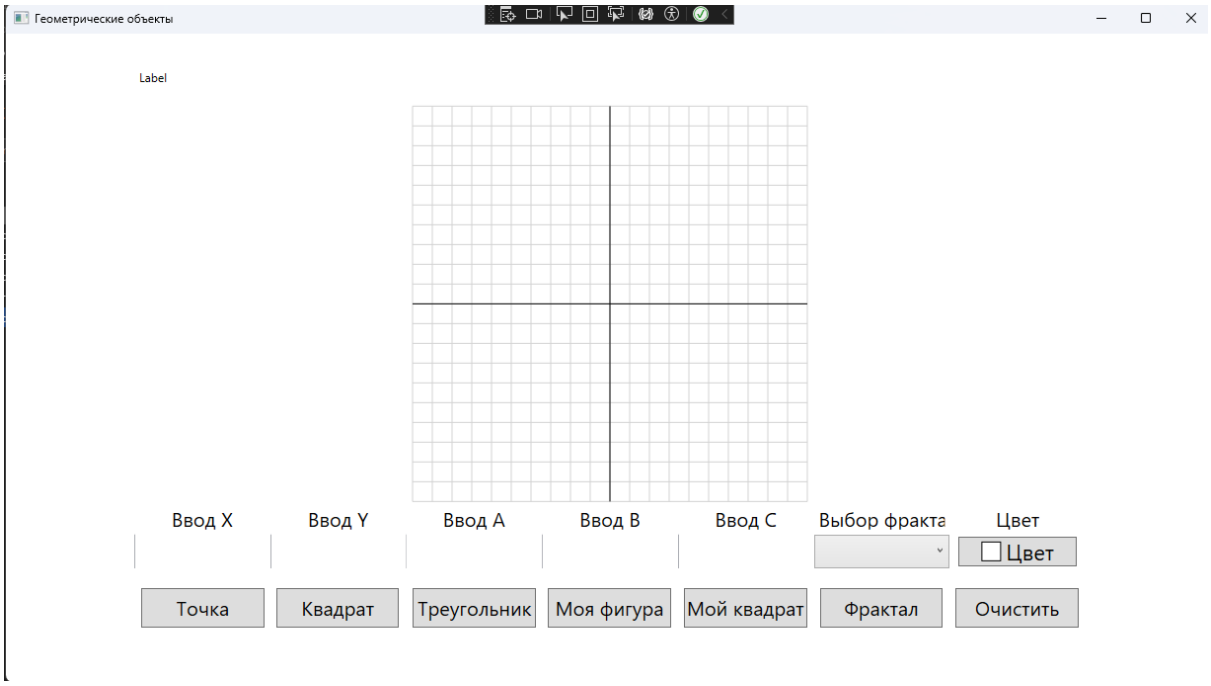
```
            labelInfo.Content = "Режим создания моей фигуры: кликните на холст, чтобы создать фигуру";
            buttonMyFigure.Background = Brushes.LightGreen;
        }
        else
        {
            // Обновление интерфейса для неактивного режима
            labelInfo.Content = "Режим создания моей фигуры выключен";
            buttonMyFigure.Background = SystemColors.ControlBrush;
        }
    }
    // Обработчик клика мыши по холсту
    private void MyCanvas_MouseDown(object sender, MouseButtonEventArgs e)
    {
        if (_myPictureMode)
        {
            // Получение позиции клика относительно холста
            Point position = e.GetPosition(myCanvas);

            // Извлечение координат X и Y
            double x = position.X;
            double y = position.Y;

            // Парсинг параметров фигуры из текстовых полей с значениями по умолчанию
            int a = int.TryParse(TextBoxA.Text, out int aVal) ? aVal : 50;
```

```
            int b = int.TryParse(TextBoxB.Text, out int bVal) ? bVal : 100;
            int c = int.TryParse(TextBoxC.Text, out int cVal) ? cVal : 125;

            // Создание и отрисовка пользовательской фигуры
            MyPicture myPicture = new MyPicture((int)x, (int)y, a, b, c, 6,
SelectedColor);
            myPicture.Draw(myCanvas, 0.7, 50);

            // Обновление информационной метки
            labelInfo.Content = myPicture.Print($"Моя фигура создана");
        }
    }
    ////////////
    private void buttonFractal_Click(object sender, RoutedEventArgs e)
    {
        _myPictureMode = false; // Отключение режима пользовательской
фигуры

        // Парсинг координат или генерация случайных значений
        int x = int.TryParse(TextBoxX.Text, out int xVal) ? xVal : rnd.Next(-
200, 200);
        int y = int.TryParse(TextBoxY.Text, out int yVal) ? yVal : rnd.Next(-
200, 200);

        // Создание и отрисовка фрактала
        Fractal fractal = new Fractal(x, y, Colors.Blue,
GetSelectedFractalType());
```

```
            fractal.Draw(myCanvas);
            labelInfo.Content = "Фрактал";
        }
        private void buttonMySquare_Click(object sender, RoutedEventArgs e)
        {
            _myPictureMode = false; // Отключение режима пользовательской
фигуры

            // Парсинг параметров или генерация случайных значений
            int x = int.TryParse(TextBoxX.Text, out int xVal) ? xVal : rnd.Next(-
200, 200);
            int y = int.TryParse(TextBoxY.Text, out int yVal) ? yVal : rnd.Next(-
200, 200);
            int a = int.TryParse(TextBoxA.Text, out int aVal) ? aVal : rnd.Next(10,
150);

            // Создание и отрисовка пользовательского квадрата
            MySquare mysquare = new MySquare(x, y, a, SelectedColor);
            mysquare.Draw(myCanvas);
            labelInfo.Content = mysquare.Print($"Я квадрат!");
        }
        private void buttonClear_Click(object sender, RoutedEventArgs e)
        {
            _myPictureMode = false; // Отключение режима пользовательской
фигуры
            myCanvas.Children.Clear(); // Очистка холста
```

```
        DrawCoordinatePlane(); // Повторная отрисовка координатной плоскости
        buttonMyFigure.Background = SystemColors.ControlBrush; // Сброс цвета кнопки
    }
    private void ColorPickerButton_Click(object sender, RoutedEventArgs e)
    {
        // Создание диалога выбора цвета
        var colorDialog = new System.Windows.Forms.ColorDialog();
        colorDialog.Color = System.Drawing.Color.FromArgb(
            _selectedColor.A, _selectedColor.R, _selectedColor.G, _selectedColor.B);

        // Проверка результата диалога
        if (colorDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
            // Обновление выбранного цвета
            SelectedColor = Color.FromArgb(
                colorDialog.Color.A,
                colorDialog.Color.R,
                colorDialog.Color.G,
                colorDialog.Color.B);

            // Обновление кисти
            SelectedColorBrush = new SolidColorBrush(SelectedColor);
        }
```

```
        }
        ////////////////////////////////////////////////////////////////////////////
        ////////////////////////////////////////////////////////////////////////////
    }
}
```

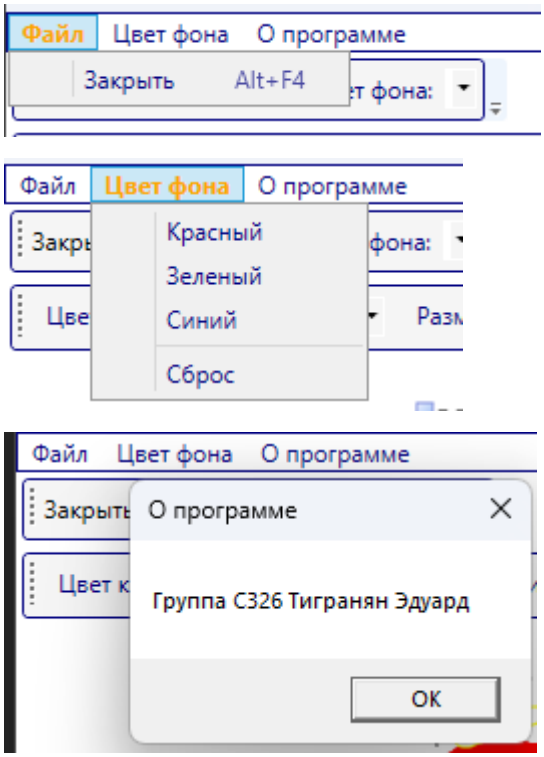

Геометрические объекты
Label
Ввод X
Ввод Y
Ввод A
Ввод B
Ввод C
Выбор фракта
Цвет
Цвет
Точка
Квадрат
Треугольник
Моя фигура
Мой квадрат
Фрактал
Очистить

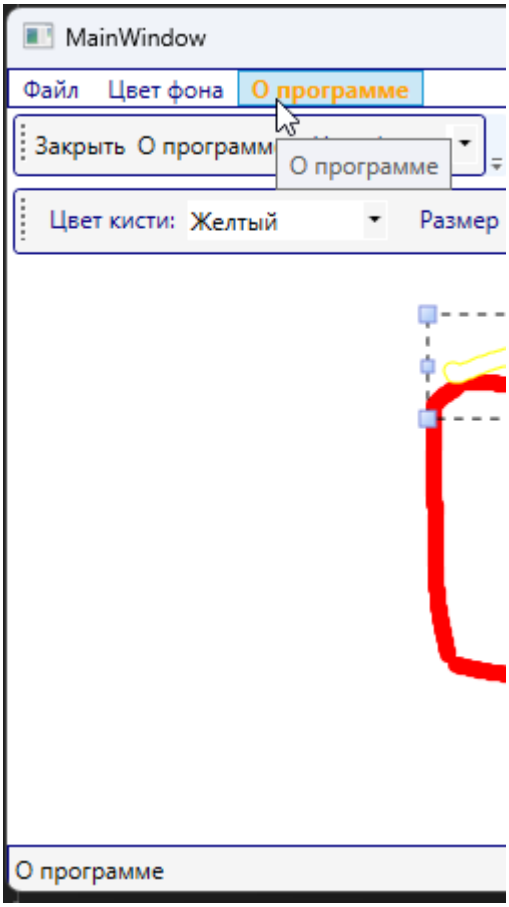
Лабораторная работа № 9-10.

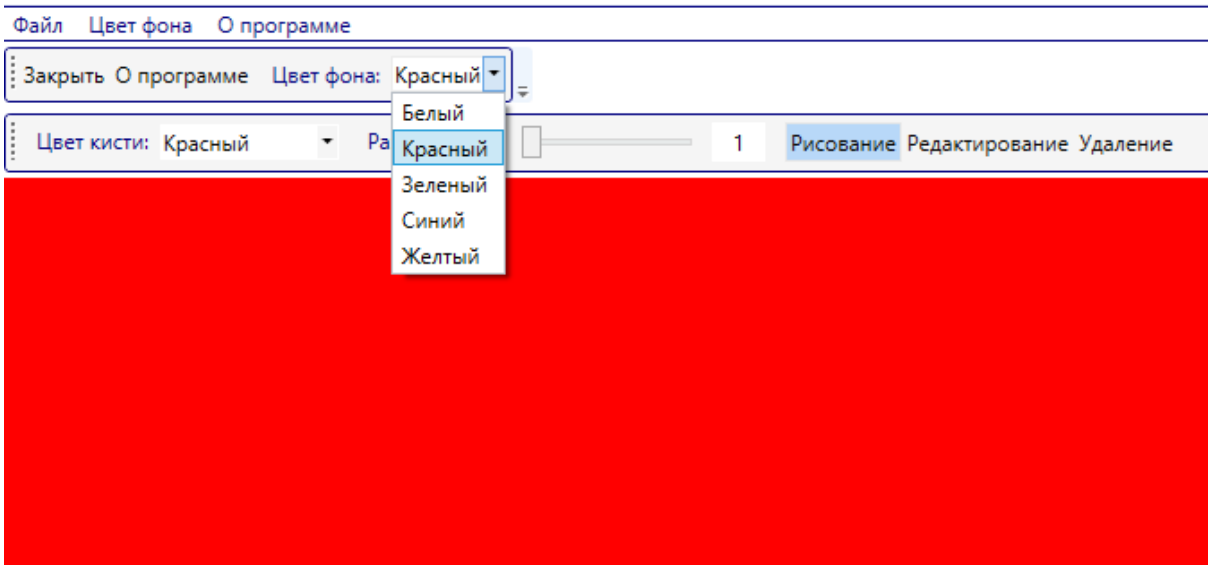
Цель работы:

Изучение элементов управления и событий мыши

Разработать WPF-приложение с меню, панелью инструментов и строкой состояния. С помощью пунктов меню пользователь может изменять цвет фона окна, получить информацию о разработчике, а также закрыть окно. Кнопки панели инструментов дублируют команды меню. При наведении на пункты меню или кнопки панели инструментов в строке состояния отображается информация об этих элементах управления.







Меню:

```
        <Menu Background="White" BorderBrush="Navy" BorderThickness="1"
Grid.Row="0">
            <MenuItem Header="_Файл">
                <MenuItem Header="_Закрыть"
                    ToolTip="Закрыть приложение"
                    InputGestureText="Alt+F4"
                    MouseLeave="MenuItem_MouseLeave"
                    MouseEnter="MenuItem_MouseEnter"
                    Click="MenuItem_Click"
                    Tag="Закрытие приложения"/>
            </MenuItem>

            <MenuItem Header="_Цвет фона"
                      MouseLeave="MenuItem_MouseLeave"
MouseEnter="MenuItem_MouseEnter" Tag="Изменение цвета фона">
                <MenuItem Header="_Красный" Click="ChangeBackgroundColor"
                          Tag="Red" ToolTip="Изменить цвет фона на красный"/>
                <MenuItem Header="_Зеленый" Click="ChangeBackgroundColor"
                          Tag="Green" ToolTip="Изменить цвет фона на зеленый"/>
                <MenuItem Header="_Синий" Click="ChangeBackgroundColor"
                          Tag="Blue" ToolTip="Изменить цвет фона на синий"/>
                <Separator/>
                <MenuItem Header="_Сброс" Click="ChangeBackgroundColor"
                          Tag="White" ToolTip="Вернуть белый цвет фона"/>
            </MenuItem>
            <MenuItem Header="_О программе"
                      ToolTip="О программе"
                      Click="MenuItem_Click"
                      MouseLeave="MenuItem_MouseLeave"
                      MouseEnter="MenuItem_MouseEnter"
                      Tag="О программе"/>
        </Menu>
```

Статус бар:

```
<StatusBar DockPanel.Dock="Bottom" VerticalAlignment="Bottom" Grid.Row="4">
```

```
<TextBlock x:Name="StatusBarLeft" Text="" />
<StatusBarItem HorizontalAlignment="Right">
<TextBlock x:Name="StatusBarRight" Text="" />
</StatusBarItem>
</StatusBar>
```

Тулбар для окна:

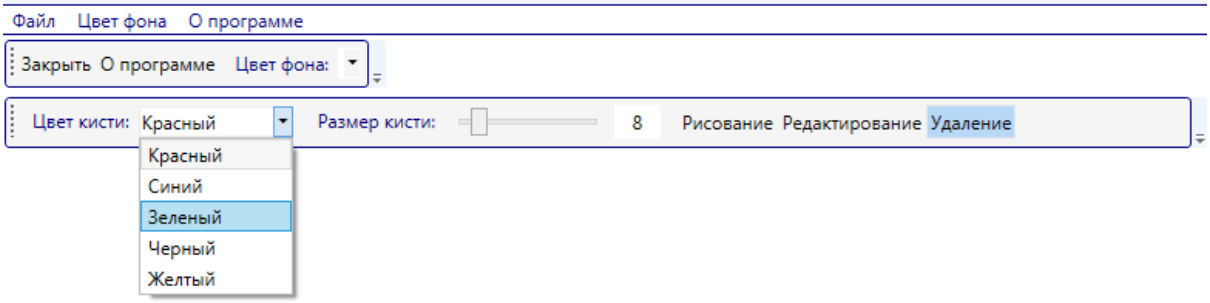
```
            <ToolBar>
                <Button Content="Закрыть"
                        Tag="Закрытие приложения"
                        ToolTip="Закрыть приложение"
                        MouseLeave="MenuItem_MouseLeave"
                        MouseEnter="MenuItem_MouseEnter"
                        Click="MenuItem_Click"/>
                <Button Content="О программе"
                        Click="MenuItem_Click"
                        ToolTip="О программе"
                        MouseLeave="MenuItem_MouseLeave"
                        MouseEnter="MenuItem_MouseEnter"
                        Tag="О программе"/>

                <Label Content="Цвет фона:" VerticalAlignment="Center"/>
                <ComboBox SelectedValuePath="Tag"
SelectionChanged="ColorChanged">
                    <ComboBoxItem Tag="White">Белый</ComboBoxItem>
                    <ComboBoxItem Tag="Red">Красный</ComboBoxItem>
                    <ComboBoxItem Tag="Green">Зеленый</ComboBoxItem>
                    <ComboBoxItem Tag="Blue">Синий</ComboBoxItem>
                    <ComboBoxItem Tag="Yellow">Желтый</ComboBoxItem>
                </ComboBox>
            </ToolBar>
```

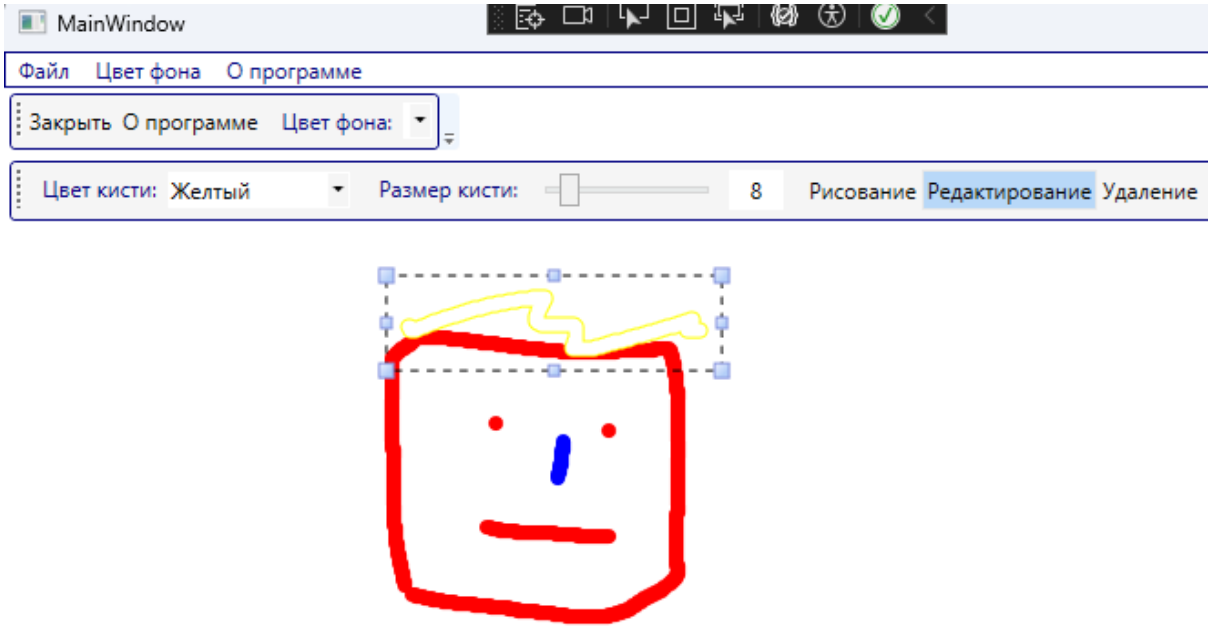
cs:

```
        private void MenuItem_Click(object sender, RoutedEventArgs e)
        {
            if (sender is FrameworkElement element && element.Tag != null)
            {
                switch (element.Tag.ToString())
                {
                    case "Закрытие приложения":
                        Application.Current.Shutdown();
                        break;
                    case "О программе":
                        MessageBox.Show("Группа С326 Тигранян Эдуард", "О
программе");
                        break;
                }
            }
        }

        private void MenuItem_MouseEnter(object sender, MouseEventArgs e)
        {
            if (sender is FrameworkElement element && element.Tag != null)
            {
                StatusBarLeft.Text = element.Tag.ToString();
```

```
    }
}

private void MenuItem_MouseLeave(object sender, MouseEventArgs e)
{
    StatusBarLeft.Text = "";
    StatusBarRight.Text = "";
}
```

Разработать WPF-приложение «Графический редактор» с выпадающим списком для выбора цвета
кисти, ползунком для выбора размеров кисти и зависимыми переключателями для выбора режима
работы: «рисование», «редактирование», «удаление».





Код:

xaml:

```
        <InkCanvas Grid.Row="3" x:Name="InkCanvas">
            <InkCanvas.DefaultDrawingAttributes>
                <DrawingAttributes Color="Red" Height="5" Width="5"/>
            </InkCanvas.DefaultDrawingAttributes>
        </InkCanvas>



        <!-- Toolbar для редактирования кисти -->
        <ToolBar Grid.Row="2">
            <Label Content="Цвет кисти:"/>
            <ComboBox x:Name="BrushColorComboBox" Width="100" SelectedIndex="0">
                <ComboBoxItem Tag="Red">Красный</ComboBoxItem>
                <ComboBoxItem Tag="Blue">Синий</ComboBoxItem>
                <ComboBoxItem Tag="Green">Зеленый</ComboBoxItem>
                <ComboBoxItem Tag="Black">Черный</ComboBoxItem>
                <ComboBoxItem Tag="Yellow">Желтый</ComboBoxItem>
            </ComboBox>

            <Label Content="Размер кисти:" Margin="10,0,0,0"/>

            <Slider x:Name="BrushSizeSlider" Width="100" Minimum="1" Maximum="50"
Value="{Binding BrushSize, Mode=TwoWay}"
                    ValueChanged="BrushSizeSlider_ValueChanged"/>

            <TextBox x:Name="BrushSizeTextBox" Width="30"
                     Text="{Binding BrushSize, StringFormat={}{0:F0},
UpdateSourceTrigger=PropertyChanged}"
```

```
                    TextAlignment="Center"/>

            <RadioButton x:Name="DrawRadioButton" Content="Рисование"
GroupName="InkMode"
                         IsChecked="True" Checked="ModeRadioButton_Checked"
Margin="10,0,0,0"/>
            <RadioButton x:Name="EditRadioButton" Content="Редактирование"
GroupName="InkMode"
                         Checked="ModeRadioButton_Checked"/>
            <RadioButton x:Name="EraseRadioButton" Content="Удаление"
GroupName="InkMode"
                         Checked="ModeRadioButton_Checked"/>
        </ToolBar>
```

Модифицируйте приложения, разработанные в предыдущей лабораторной работе: удалите как можно
больше обработчиков событий и реализуйте ту же функциональность приложения с помощью привязки
данных.

```
<Slider x:Name="BrushSizeSlider" Width="100" Minimum="1" Maximum="50"
Value="{Binding BrushSize, Mode=TwoWay}"
ValueChanged="BrushSizeSlider_ValueChanged"/>


<TextBox x:Name="BrushSizeTextBox" Width="30" ="{Binding BrushSize,
StringFormat={}{0:F0}, UpdateSourceTrigger=PropertyChanged}"
TextAlignment="Center"/>
```

Полный код программы

```
using System;
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Ink;
using System.Diagnostics;

namespace code
{
    public partial class MainWindow : Window, INotifyPropertyChanged
    {
        private double _brushSize = 5;
        private bool _updatingBrushSize = false;

        public event PropertyChangedEventHandler PropertyChanged;

        public double BrushSize
        {
            get => _brushSize;
            set
            {
                if (Math.Abs(_brushSize - value) > double.Epsilon &&
!_updatingBrushSize)
                {
                    _updatingBrushSize = true;
                    _brushSize = value;
                    OnPropertyChanged(nameof(BrushSize));
                    UpdateBrushSize();
                    _updatingBrushSize = false;
                }
            }
        }

        public MainWindow()
        {
            InitializeComponent();
            DataContext = this;
            Loaded += MainWindow_Loaded;
        }

        private void MainWindow_Loaded(object sender, RoutedEventArgs e)
        {
            // Initialize brush color
            BrushColorComboBox.SelectionChanged +=
BrushColorComboBox_SelectionChanged;
            UpdateBrushColor();

            // Initialize brush size controls
            if (BrushSizeSlider != null)
            {
                BrushSizeSlider.ValueChanged += BrushSizeSlider_ValueChanged;
                BrushSizeSlider.Value = BrushSize;
            }

            if (BrushSizeTextBox != null)
```

```
            {
                BrushSizeTextBox.TextChanged += BrushSizeTextBox_TextChanged;
                BrushSizeTextBox.Text = BrushSize.ToString("F0");
            }

            // Initialize drawing mode
            InkCanvas.EditingMode = InkCanvasEditingMode.Ink;
            StatusBarRight.Text = "Режим: Рисование";
        }

        private void MenuItem_Click(object sender, RoutedEventArgs e)
        {
            if (sender is FrameworkElement element && element.Tag != null)
            {
                switch (element.Tag.ToString())
                {
                    case "Закрытие приложения":
                        Application.Current.Shutdown();
                        break;
                    case "О программе":
                        MessageBox.Show("Группа С326 Тигранян Эдуард", "О
программе");
                        break;
                }
            }
        }

        private void MenuItem_MouseEnter(object sender, MouseEventArgs e)
        {
            if (sender is FrameworkElement element && element.Tag != null)
            {
                StatusBarLeft.Text = element.Tag.ToString();
            }
        }

        private void MenuItem_MouseLeave(object sender, MouseEventArgs e)
        {
            StatusBarLeft.Text = "";
            StatusBarRight.Text = "";
        }

        private void ChangeBackgroundColor(object sender, RoutedEventArgs e)
        {
            if (sender is FrameworkElement element && element.Tag != null)
            {
                var color =
(Color)ColorConverter.ConvertFromString(element.Tag.ToString());
                InkCanvas.Background = new SolidColorBrush(color);
                Background = new SolidColorBrush(color);
            }
        }

        private void ColorChanged(object sender, SelectionChangedEventArgs e)
        {
            if (sender is ComboBox comboBox && comboBox.SelectedItem is
ComboBoxItem item && item.Tag != null)
            {
                InkCanvas.Background = new SolidColorBrush(
```

```
(Color)ColorConverter.ConvertFromString(item.Tag.ToString()));
            }
        }

        private void BrushColorComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            UpdateBrushColor();
        }

        private void UpdateBrushColor()
        {
            if (BrushColorComboBox?.SelectedItem is ComboBoxItem item && item.Tag
!= null)
            {
                var color =
(Color)ColorConverter.ConvertFromString(item.Tag.ToString());
                InkCanvas.DefaultDrawingAttributes.Color = color;
            }
        }

        private void BrushSizeSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            // Prevent recursive updates
            if (Math.Abs(BrushSize - e.NewValue) > double.Epsilon)
            {
                BrushSize = e.NewValue;
                if (BrushSizeTextBox != null && !_updatingBrushSize)
                {
                    BrushSizeTextBox.Text = e.NewValue.ToString("F0");
                }
            }
        }

        private void BrushSizeTextBox_TextChanged(object sender,
TextChangedEventArgs e)
        {
            if (double.TryParse(BrushSizeTextBox.Text, out double newSize) &&
                Math.Abs(BrushSize - newSize) > double.Epsilon &&
!_updatingBrushSize)
            {
                BrushSize = newSize;
                if (BrushSizeSlider != null)
                {
                    BrushSizeSlider.Value = newSize;
                }
            }
        }

        private void UpdateBrushSize()
        {
            if (InkCanvas?.DefaultDrawingAttributes == null) return;

            try
            {
                InkCanvas.DefaultDrawingAttributes.Height = BrushSize;
                InkCanvas.DefaultDrawingAttributes.Width = BrushSize;
```

```
            }
            catch (Exception ex)
            {
                Debug.WriteLine($"Error updating brush size: {ex.Message}");
            }
        }

        private void ModeRadioButton_Checked(object sender, RoutedEventArgs e)
        {
            if (InkCanvas == null) return;

            if (DrawRadioButton?.IsChecked == true)
            {
                InkCanvas.EditingMode = InkCanvasEditingMode.Ink;
                StatusBarRight.Text = "Режим: Рисование";
            }
            else if (EditRadioButton?.IsChecked == true)
            {
                InkCanvas.EditingMode = InkCanvasEditingMode.Select;
                StatusBarRight.Text = "Режим: Редактирование";
            }
            else if (EraseRadioButton?.IsChecked == true)
            {
                InkCanvas.EditingMode = InkCanvasEditingMode.EraseByStroke;
                StatusBarRight.Text = "Режим: Удаление";
            }
        }

        protected virtual void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

```
<Window x:Class="code.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:code"
        mc:Ignorable="d"
        Title="MainWindow" Height="450" Width="800">

    <Window.Resources>
        <Style TargetType="MenuItem">
            <Setter Property="Foreground" Value="Navy"/>
            <Setter Property="FontWeight" Value="Normal"/>
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Foreground" Value="Orange"/>
                    <Setter Property="Background" Value="Navy"/>
                    <Setter Property="FontWeight" Value="Bold"/>
                </Trigger>
            </Style.Triggers>
        </Style>

        <Style TargetType="Button">
            <Setter Property="Foreground" Value="Navy"/>
            <Setter Property="Background" Value="Transparent"/>
            <Setter Property="BorderBrush" Value="Transparent"/>
            <Setter Property="Margin" Value="2"/>
            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Foreground" Value="Orange"/>
                    <Setter Property="Background" Value="Navy"/>
                    <Setter Property="BorderBrush" Value="Navy"/>
                </Trigger>
            </Style.Triggers>
        </Style>

        <Style TargetType="ToolBar">
            <Setter Property="Background" Value="WhiteSmoke"/>
            <Setter Property="BorderBrush" Value="Navy"/>
            <Setter Property="BorderThickness" Value="1"/>
            <Setter Property="Margin" Value="0,2,0,2"/>
        </Style>

        <Style TargetType="ComboBox">
            <Setter Property="Margin" Value="5"/>
            <Setter Property="Foreground" Value="Navy"/>
            <Setter Property="BorderBrush" Value="Navy"/>
        </Style>

        <Style TargetType="Label">
            <Setter Property="Foreground" Value="Navy"/>
            <Setter Property="Margin" Value="5,0,0,0"/>
        </Style>

        <Style TargetType="RadioButton">
            <Setter Property="Foreground" Value="Navy"/>
            <Setter Property="Margin" Value="5,0,0,0"/>
        </Style>
```

```
        <Style TargetType="Slider">
            <Setter Property="Margin" Value="5"/>
        </Style>

        <Style TargetType="TextBox">
            <Setter Property="Margin" Value="5"/>
            <Setter Property="BorderBrush" Value="Navy"/>
        </Style>

        <Style TargetType="StatusBar">
            <Setter Property="Background" Value="WhiteSmoke"/>
            <Setter Property="BorderBrush" Value="Navy"/>
            <Setter Property="BorderThickness" Value="1,1,0,0"/>
        </Style>
    </Window.Resources>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
            <RowDefinition></RowDefinition>
            <RowDefinition Height="auto"></RowDefinition>
        </Grid.RowDefinitions>

        <Menu Background="White" BorderBrush="Navy" BorderThickness="1"
Grid.Row="0">
            <MenuItem Header="_Файл">
                <MenuItem Header="_Закрыть"
                    ToolTip="Закрыть приложение"
                    InputGestureText="Alt+F4"
                    MouseLeave="MenuItem_MouseLeave"
                    MouseEnter="MenuItem_MouseEnter"
                    Click="MenuItem_Click"
                    Tag="Закрытие приложения"/>
            </MenuItem>

            <MenuItem Header="_Цвет фона"
                      MouseLeave="MenuItem_MouseLeave"
MouseEnter="MenuItem_MouseEnter" Tag="Изменение цвета фона">
                <MenuItem Header="_Красный" Click="ChangeBackgroundColor"
                          Tag="Red" ToolTip="Изменить цвет фона на красный"/>
                <MenuItem Header="_Зеленый" Click="ChangeBackgroundColor"
                          Tag="Green" ToolTip="Изменить цвет фона на зеленый"/>
                <MenuItem Header="_Синий" Click="ChangeBackgroundColor"
                          Tag="Blue" ToolTip="Изменить цвет фона на синий"/>
                <Separator/>
                <MenuItem Header="_Сброс" Click="ChangeBackgroundColor"
                          Tag="White" ToolTip="Вернуть белый цвет фона"/>
            </MenuItem>
            <MenuItem Header="_О программе"
                      ToolTip="О программе"
                      Click="MenuItem_Click"
                      MouseLeave="MenuItem_MouseLeave"
                      MouseEnter="MenuItem_MouseEnter"
                      Tag="О программе"/>
        </Menu>

        <StackPanel Grid.Row="1" Orientation="Horizontal">
```

```
            <ToolBar>
                <Button Content="Закрыть"
                        Tag="Закрытие приложения"
                        ToolTip="Закрыть приложение"
                        MouseLeave="MenuItem_MouseLeave"
                        MouseEnter="MenuItem_MouseEnter"
                        Click="MenuItem_Click"/>
                <Button Content="О программе"
                        Click="MenuItem_Click"
                        ToolTip="О программе"
                        MouseLeave="MenuItem_MouseLeave"
                        MouseEnter="MenuItem_MouseEnter"
                        Tag="О программе"/>

                <Label Content="Цвет фона:" VerticalAlignment="Center"/>
                <ComboBox SelectedValuePath="Tag"
SelectionChanged="ColorChanged">
                    <ComboBoxItem Tag="White">Белый</ComboBoxItem>
                    <ComboBoxItem Tag="Red">Красный</ComboBoxItem>
                    <ComboBoxItem Tag="Green">Зеленый</ComboBoxItem>
                    <ComboBoxItem Tag="Blue">Синий</ComboBoxItem>
                    <ComboBoxItem Tag="Yellow">Желтый</ComboBoxItem>
                </ComboBox>
            </ToolBar>
        </StackPanel>

        <!-- Toolbar для редактирования кисти -->
        <ToolBar Grid.Row="2">
            <Label Content="Цвет кисти:"/>
            <ComboBox x:Name="BrushColorComboBox" Width="100" SelectedIndex="0">
                <ComboBoxItem Tag="Red">Красный</ComboBoxItem>
                <ComboBoxItem Tag="Blue">Синий</ComboBoxItem>
                <ComboBoxItem Tag="Green">Зеленый</ComboBoxItem>
                <ComboBoxItem Tag="Black">Черный</ComboBoxItem>
                <ComboBoxItem Tag="Yellow">Желтый</ComboBoxItem>
            </ComboBox>

            <Label Content="Размер кисти:" Margin="10,0,0,0"/>

            <Slider x:Name="BrushSizeSlider" Width="100" Minimum="1" Maximum="50"
Value="{Binding BrushSize, Mode=TwoWay}"
                    ValueChanged="BrushSizeSlider_ValueChanged"/>

            <TextBox x:Name="BrushSizeTextBox" Width="30"
                    Text="{Binding BrushSize, StringFormat={}{0:F0},
UpdateSourceTrigger=PropertyChanged}"
                    TextAlignment="Center"/>

            <RadioButton x:Name="DrawRadioButton" Content="Рисование"
GroupName="InkMode"
                         IsChecked="True" Checked="ModeRadioButton_Checked"
Margin="10,0,0,0"/>
            <RadioButton x:Name="EditRadioButton" Content="Редактирование"
GroupName="InkMode"
                         Checked="ModeRadioButton_Checked"/>
            <RadioButton x:Name="EraseRadioButton" Content="Удаление"
GroupName="InkMode"
                         Checked="ModeRadioButton_Checked"/>
        </ToolBar>
```

```
        <InkCanvas Grid.Row="3" x:Name="InkCanvas">
            <InkCanvas.DefaultDrawingAttributes>
                <DrawingAttributes Color="Red" Height="5" Width="5"/>
            </InkCanvas.DefaultDrawingAttributes>
        </InkCanvas>

        <StatusBar DockPanel.Dock="Bottom" VerticalAlignment="Bottom"
Grid.Row="4">
            <TextBlock x:Name="StatusBarLeft" Text="" />
            <StatusBarItem HorizontalAlignment="Right">
                <TextBlock x:Name="StatusBarRight" Text="" />
            </StatusBarItem>
        </StatusBar>
    </Grid>
</Window>
```

Лабораторная работа № 11-12.

Цель работы: сокращение xaml код путём использования стилей и тригерров.

Стиль S_PlaceHolderText для textbox с placeholder text

```
<Style x:Key="S_PlaceHolderText" TargetType="TextBox" BasedOn="{StaticResource
S_TextBox}">
        <Setter Property="VerticalContentAlignment" Value="Center"></Setter>
        <Setter Property="BorderBrush" Value="{Binding BorderColor,
Source={x:Static local:Properties.Instance}}"/>
        <Setter Property="FontSize" Value="14"></Setter>
        <Setter Property="FontFamily" Value="Segoe UI" />
        <Setter Property="FontWeight" Value="Normal" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type TextBox}">
                    <Grid>
                        <Rectangle Margin="0,0,0,3" x:Name="border"
                              Fill="{Binding FG_Control, Source={x:Static
local:Properties.Instance}}"
                              Height="1"
                              VerticalAlignment="Bottom"/>
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition
Width="auto"></ColumnDefinition>
                                <ColumnDefinition Width="*"></ColumnDefinition>
                            </Grid.ColumnDefinitions>
                            <Image Grid.Column="0"
                                  Source="{Binding
Path=(local:TagProperties.CustomTag),
                                  RelativeSource={RelativeSource
TemplatedParent}}"
                                  Width="22"
                                  Height="18"
                                  Visibility="Visible">
                            </Image>
                            <TextBlock x:Name="textBlock"
                                  Grid.Column="{Binding
Path=(local:TagProperties.IntTag),
                                  RelativeSource={RelativeSource
TemplatedParent}}"
                                  Margin="6,0,0,0"
                                  FontSize="14"
                                  Foreground="DimGray"
                                  IsHitTestVisible="False"
                                  Text="{Binding Tag,
RelativeSource={RelativeSource TemplatedParent}}"
                                  Visibility="Collapsed"
                                  VerticalAlignment="Center"
                                  HorizontalAlignment="Left"
                                  >
                                <TextBlock.Style>
```

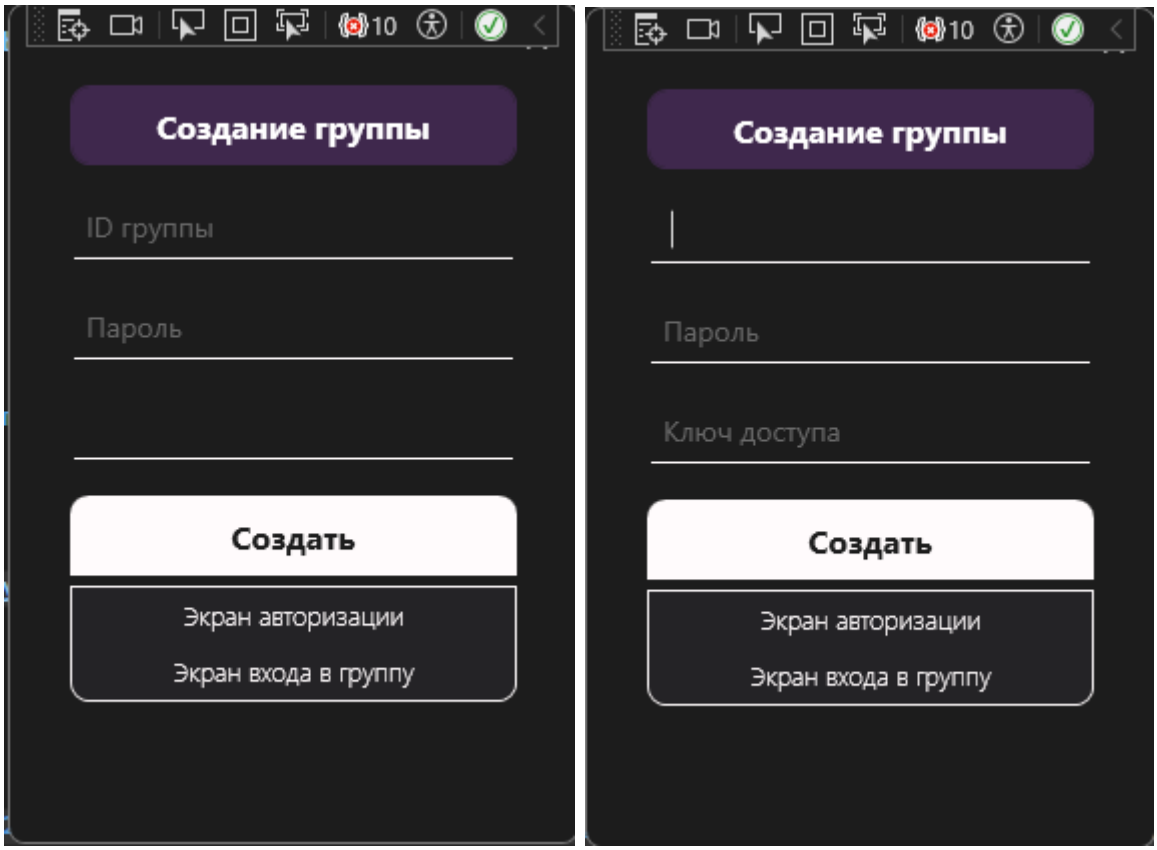
```
                                    <Style TargetType="TextBlock">
                                        <Setter Property="Visibility"
Value="Visible"/>
                                        <Style.Triggers>
                                            <DataTrigger Binding="{Binding
Path=(local:TagProperties.IntTag),
                                            RelativeSource={RelativeSource
TemplatedParent}}" Value="0">
                                                <Setter Property="Width"
Value="1000"/>
                                            </DataTrigger>
                                        </Style.Triggers>
                                    </Style>
                                </TextBlock.Style>
                            </TextBlock>
                            <ScrollViewer x:Name="PART_ContentHost"
                                          Margin="6,0,0,0"
                                          Grid.Column="{Binding
Path=(local:TagProperties.IntTag),
                                          RelativeSource={RelativeSource
TemplatedParent}}"
                                          />
                        </Grid>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger Property="Text" Value="">
                            <Setter TargetName="textBlock" Property="Visibility"
Value="Visible" />
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="False">
                            <Setter TargetName="border" Property="Opacity"
Value="0.56" />
                        </Trigger>
                        <Trigger Property="IsMouseOver" Value="True">
                        </Trigger>
                        <Trigger Property="IsKeyboardFocused" Value="True">
                            <Setter TargetName="textBlock" Property="Visibility"
Value="Collapsed" />
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

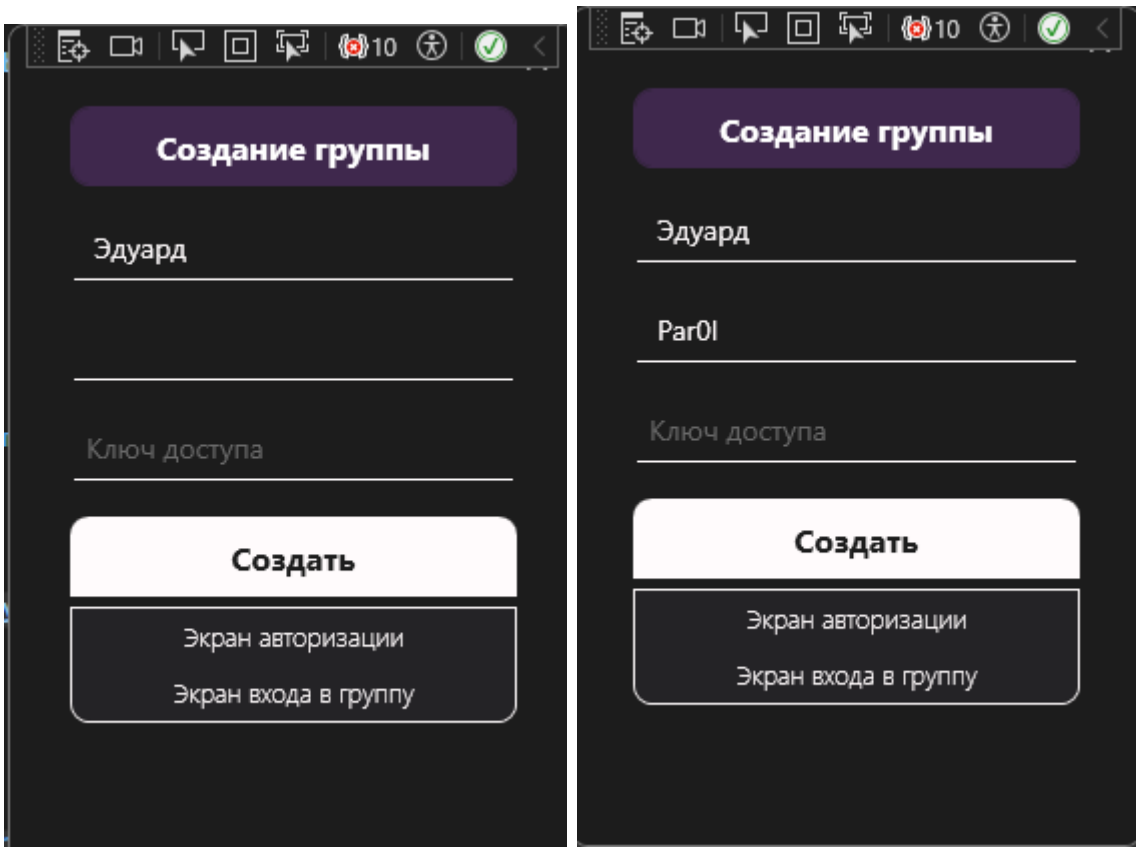
Применения стиля:

```
<TextBox x:Name ="GroupNameCreateGroupTextBox" Grid.Row="1"
Style="{StaticResource S_PlaceHolderText}"
    Tag="ID группы"
    Margin="12,5,12,5"
    local:TagProperties.IntTag="0"
    PreviewTextInput="TextBox_PreviewTextInput"
         PreviewKeyDown="TextBox_PreviewKeyDown"
    ></TextBox>
<TextBox x:Name  ="GroupPasswordCreateGroupTextBox" Grid.Row="2"
Style="{StaticResource S_PlaceHolderText}"
```

```
Tag="Пароль"
local:TagProperties.IntTag="0"
Margin="12,5,12,5"
PreviewTextInput="TextBox_PreviewTextInput"
     ></TextBox>
```

Результат:




Создание группы
Эдуард
Ключ доступа
Создать
Экран авторизации
Экран входа в группу
Создание группы
Эдуард
Par0l
Ключ доступа
Создать
Экран авторизации
Экран входа в группу