Лабораторная работа №12. Разработка приложений в Windows Forms

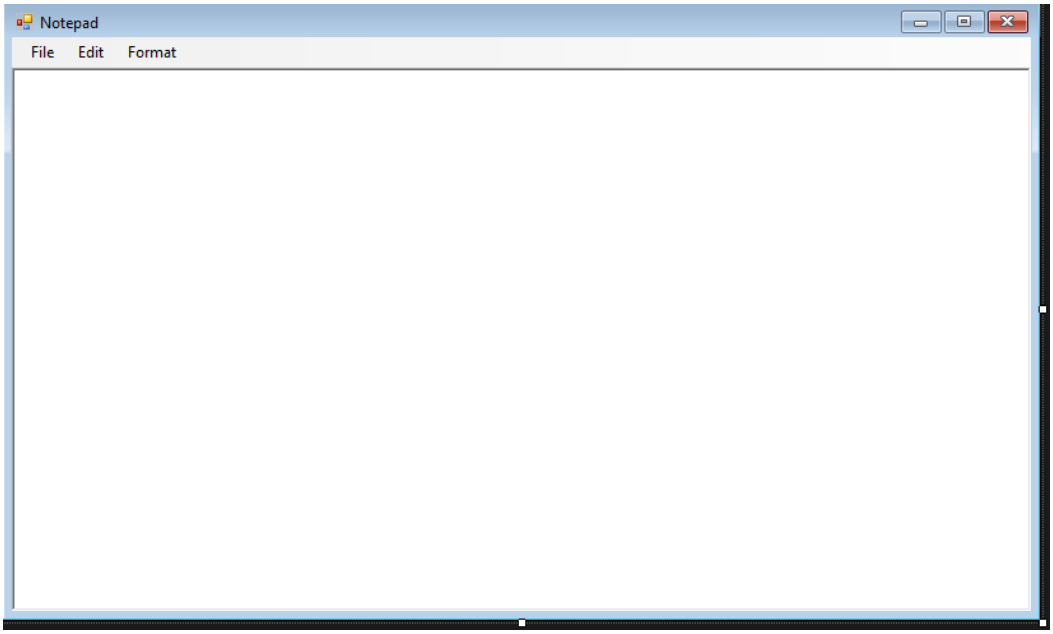
Приложение 1. Блокнот

1. Создаём **Windows Form Application**

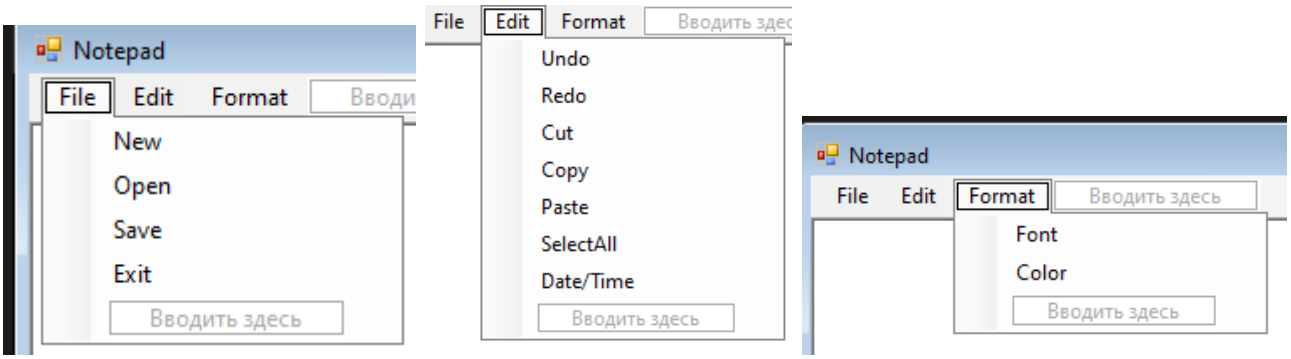
2. Называем его **NotePad**

3. Измените имя формы в ее свойствах. Оно будет отображаться в верхней части Блокнота как ее заголовок.

4. Из Toolbox выберите **MenuStrip** и поместите ее наверх в области формы. В **MenuStrip** вы можете указать названия различных опций, которые вы хотите видеть в своем блокноте. Мы добавляем в меню опции **File**, **Edit** и **Format** . Вы можете добавить больше по своему выбору.

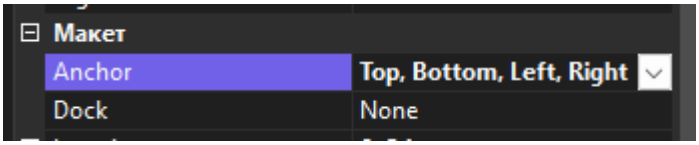


5. Теперь в опциях, представленных в **MenuBar**, нам нужно иметь диалоговое окно, которое будет открываться, когда пользователь нажимает на опции **File**, **Edit** или **Font**:



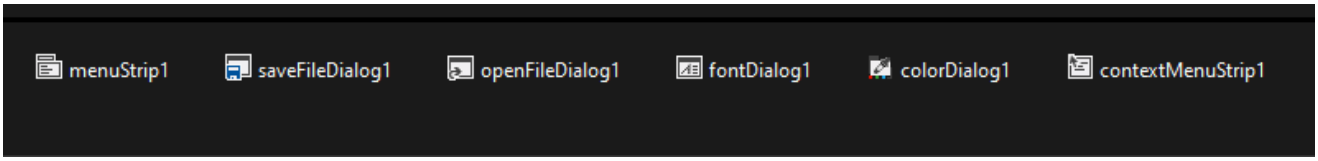
6. Теперь нам нужно добавить элемент управления **RichTextBox** из панели инструментов в форму, чтобы пользователь мог вводить данные в это поле. Здесь мы не добавляем простое текстовое поле, поскольку оно может принимать однострочный ввод, тогда как **RichTextBox** обеспечивает больший контроль над стилизацией текста.

В свойствах **RichTextBox** задайте свойство привязки сверху, снизу, слева и справа, а свойство закрепления — как заполнение, чтобы поле **RichTextBox** было распределено по всей области экрана блокнота и полностью покрывало ее.



7. Теперь нам нужно предоставить функциональность различным опциям в диалоговом окне. Для таких опций, как **Сохранить**, **Открыть**, **Шрифт** и **Цвет**, нам нужны некоторые специальные элементы управления из панели инструментов.

- Для сохранения: нам нужно добавить элемент управления **saveFileDialog** из панели инструментов.
- Для открытия: нам нужно добавить элемент управления **openFileDialog** из панели инструментов.
- Для шрифта: нам нужно добавить элемент управления **fontDialog** из панели инструментов.
- Для цвета: нам нужно добавить элемент управления **colorDialog** из панели инструментов.



8. Дважды щёлкаем по нашим вкладкам в **MenuBar**.

```
namespace NotePad2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void newToolStripMenuItem_Click(object sender, EventArgs
e)
```

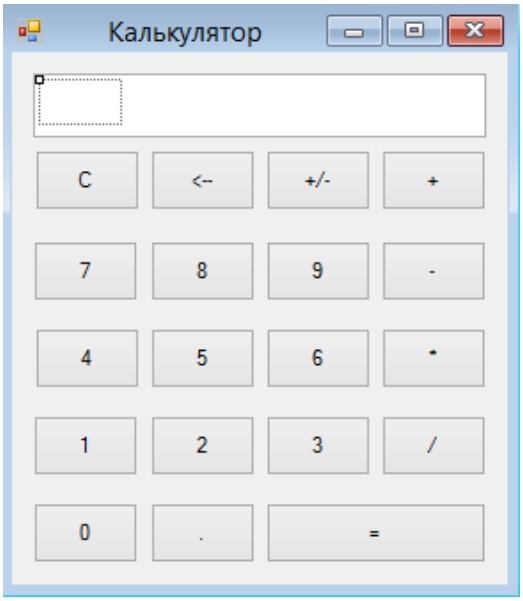
```
        {
            richTextBox1.Clear();
        }

        private void openToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
                richTextBox1.Text =
File.ReadAllText(openFileDialog1.FileName);
        }
        private void saveToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            saveFileDialog1.DefaultExt = ".txt";
            saveFileDialog1.Filter = "Text File|*.txt|PDF file|*.pdf|Word
File|*.doc";
            DialogResult dialogResult = saveFileDialog1.ShowDialog();
            if (dialogResult == DialogResult.OK)
                File.WriteAllText(saveFileDialog1.FileName,
richTextBox1.Text);
        }
        private void exitToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            Application.Exit();
        }
        private void undoToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            richTextBox1.Undo();
        }
        private void redoToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            richTextBox1.Redo();
        }
        private void cutToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            richTextBox1.Cut();
        }
        private void copyToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            richTextBox1.Copy();
        }
        private void pasteToolStripMenuItem_Click(object sender,
EventArgs e)
        {
```

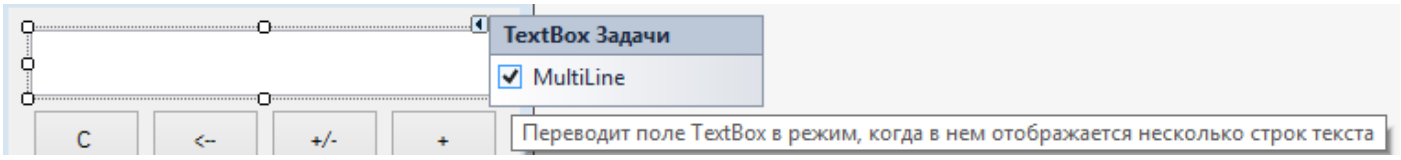
```
            richTextBox1.Paste();
        }
        private void selectAllToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            richTextBox1.SelectAll();
        }
        private void dateTimeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            richTextBox1.Text = System.DateTime.Now.ToString();
        }
        private void fontToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (fontDialog1.ShowDialog() == DialogResult.OK)
            {
                richTextBox1.Font = fontDialog1.Font;
            }
        }
        private void colorToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (colorDialog1.ShowDialog() == DialogResult.OK)
            {
                richTextBox1.ForeColor = colorDialog1.Color;
            }
        }
    }
}
```

Приложение 2. Калькулятор

1. Создаём следующую форму:



Здесь у нас 19 кнопок **Button**, 1 **Textbox** и ещё 1 пустой **Label**. Увеличим ширину **TextBox**, используя **MultiLine**:



Также в Свойствах увеличим размер шрифта в **TextBox** и **Label** до **12** пт.

2. Переименовываем кнопки в нормальные названия:

3. Для этого дважды кликаем на кнопке ***«0»*** и в открывшемся коде пишем:

```
private void button0_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 0;
}
```

Делаем то же самое с остальными цифровыми кнопками:

```
private void button1_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 1;
}

private void button2_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 2;
}

private void button3_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 3;
}

private void button4_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 4;
}

private void button5_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 5;
}
```

```
private void button6_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 6;
}

private void button7_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 7;
}

private void button8_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 8;
}

private void button_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + 9;
}
```

4. Таким же образом кликаем дважды на кнопку «.» в форме. Она будет использоваться для создания десятичной дроби. Пишем следующий код:

```
private void buttonPoint_Click(object sender, EventArgs e)
{
    textBox1.Text = textBox1.Text + ",";
}
```

5. Кнопки нажимаются, в **TextBox** отображаются нажатые цифры. Теперь надо научить программу производить с ними какие-либо операции. Как видно из формы, наш калькулятор сможет производить стандартные математические операции: сложение, вычитание, умножение и деление. Для начала мы создадим в самом начале программы несколько переменных, которые нам для этого понадобятся:

```
float number1, number2;
int count;
bool operations = true;
```

Первым двум переменным будут присваиваться значения, набранные пользователем в калькуляторе. В последствии с ними будут производиться нужные математические операции. Тип float — это тип с плавающей точкой, позволяющий работать с десятичными дробями, что нам, безусловно, нужно при наличии кнопки «.»

Благодаря второй переменной мы будем давать программе указания, какую именно операцию производить с переменными, описанными выше. Здесь нам не нужна дробь, поэтому обойдёмся целочисленным типом int.

Последняя переменная **operations** нам понадобится для того, чтобы менять знаки у введённых чисел. Тип bool может иметь два значения — true и false. Мы представим, что если operations имеет значение true в программе, то это означает, что у числа знак +, если false — число отрицательное и перед собой имеет знак —. Изначально в калькуляторе вбиваются положительные числа, поэтому мы сразу присвоили переменной значение true.

6. Далее мы дважды нажимаем на кнопку «+» и пишем следующий код:

```
private void buttonAddition_Click(object sender, EventArgs e)
{
    number1 = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 1;
    label1.Text = number1.ToString() + "+";
    operations = true;
}
```

В строке 3 мы присваиваем первой переменной а то, что будет написано в TextBox'е (а именно число, которое введёт пользователь перед тем, как нажать кнопку «+»).

Затем TextBox очищается, число, введённое пользователем, в нём пропадает (но остаётся в переменной number1)

Переменной count присваивается число 1, которая потом укажет программе, что именно операцию сложения надо будет произвести с числами.

Затем в Label записывается число из переменной a (то самое, которое изначально ввёл пользователь) и знак плюса.

Кроме того, как бы не было странным с первого взгляда, мы присваиваем переменной **operations** значение true, хотя выше, в начале кода, мы и так присваивали это же значение. Подробнее данную переменную мы опишем ниже, но смысл в том, что мы присваиваем значение true, когда хотим сделать введённое число отрицательным, если оно положительно, а значение false, когда хотим сделать число положительным, если оно отрицательное.

7. Подобным образом заполняем код для кнопок «-«, «*» и «/»:

```
private void buttonSubdivision_Click(object sender, EventArgs e)
{
    number1 = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 2;
    label1.Text = number1.ToString() + "-";
    operations = true;
}

private void buttonMultiplication_Click(object sender, EventArgs e)
{
    number1 = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 3;
    label1.Text = number1.ToString() + "*";
    operations = true;
}

private void buttonDivision_Click(object sender, EventArgs e)
{
    number1 = float.Parse(textBox1.Text);
    textBox1.Clear();
    count = 4;
    label1.Text = number1.ToString() + "/";
    operations = true;
}
```

8. Далее нам понадобится создать функцию, которая будет применять нужные нам математические операции к числам. Назовём её **calculate**. Но перед этим мы кликнем дважды на кнопку «=» на форме и в коде к ней мы запишем:

```
private void buttonEquvals_Click(object sender, EventArgs e)
{
    Calculate();
    label1.Text = "";
}
```

То есть, при нажатии пользователем на кнопку «=», как раз выполнится наша функция подсчёта calculate, и, заодно, очистится Label, так как результат мы в будущем коде выведем в TextBox.

Теперь создаём нашу функцию calculate и пишем следующий код:

```
private void Calculate()
{
    switch(count)
    {
        case 1:
            number2 = number1 + float.Parse(textBox1.Text);
            textBox1.Text = number2.ToString();
            break;
        case 2:
            number2 = number1 - float.Parse(textBox1.Text);
            textBox1.Text = number2.ToString();
            break;
        case 3:
            number2 = number1 * float.Parse(textBox1.Text);
            textBox1.Text = number2.ToString();
            break;
        case 4:
            number2 = number1 / float.Parse(textBox1.Text);
            textBox1.Text = number2.ToString();
            break;

        default:
            break;
    }
}
```

9. Дважды жмём в форме на кнопку «C». Она будет очищать все записи из TextBox’a и Label’a.

Код у неё элементарный:

```
private void buttonC_Click(object sender, EventArgs e)
{
    textBox1.Text = "";
    label1.Text = "";
}
```

10. На очереди у нас кнопка «<—«. Она будет удалять последнюю цифру записанного в TextBox’e числа. Код:

```
private void buttonDelete_Click(object sender, EventArgs e)
{
    int lenght = textBox1.Text.Length - 1;
    string text = textBox1.Text;
    textBox1.Clear();
    for (int i = 0; i < lenght; i++)
    {
        textBox1.Text = textBox1.Text + text[i];
    }
}
```

Мы вводим новую переменную lenght целочисленного типа и записываем в неё количество символов в TextBox’е минус один символ.

Также мы вводим новую переменную text, в которую полностью заносим текст из TextBox’а. Затем мы очищаем TextBox и вводим цикл for, через который записываем в TextBox строку text, но уже на символ короче.

Например, в TextBox’е записано число 504523

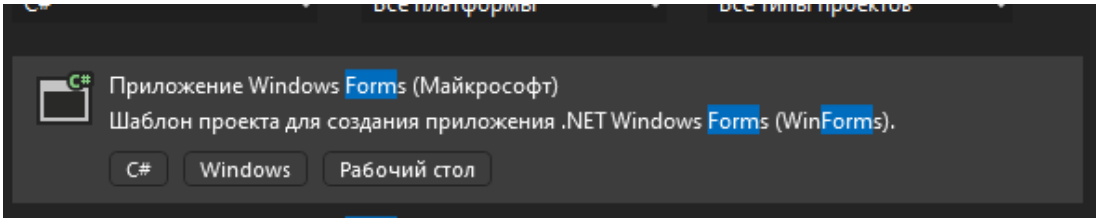
При нажатии на кнопку «<—« в переменную lenght записывается число 5 (6 цифр — 1), в переменную text записывается строка «504523», TextBox очищается, а затем в него по одному записываются символы из text, но в этот раз их будет не 6, а 5, то есть в TextBox’е появится число 50452.

11. У нас остаётся последняя кнопка, которая отвечает за знак первого слагаемого. Переходим к её коду. Тут мы будет работать с переменной operation, которую описывали выше. Код выглядит вот так:

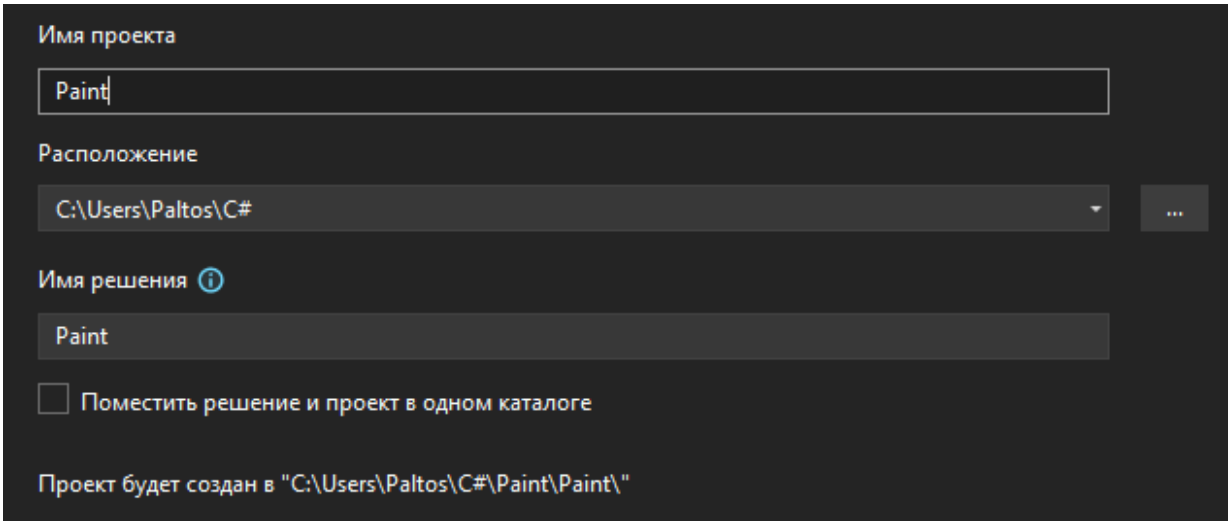
```
private void buttonZnak_Click(object sender, EventArgs e)
{
    if (operations == true)
    {
        textBox1.Text = "-" + textBox1.Text;
        operations = false;
    }
    else if (operations == false)
    {
        textBox1.Text = textBox1.Text.Replace("-", "");
        operations = true;
    }
}
```

Приложение 3. Paint

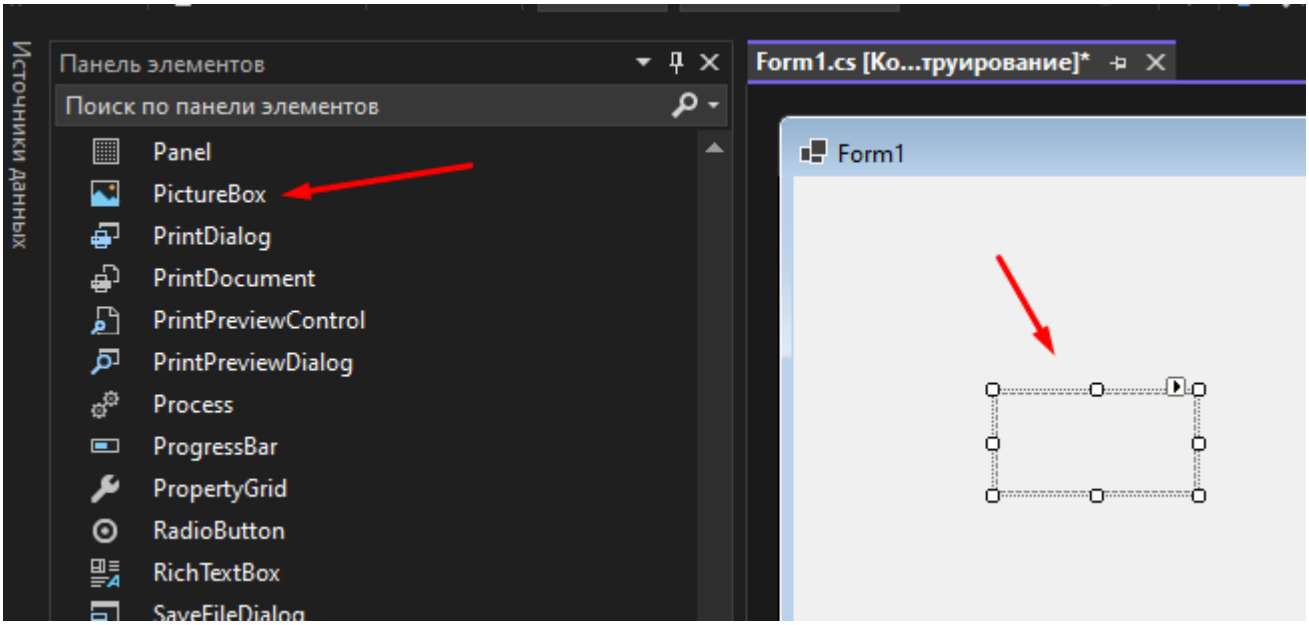
1. Создаём новый проект



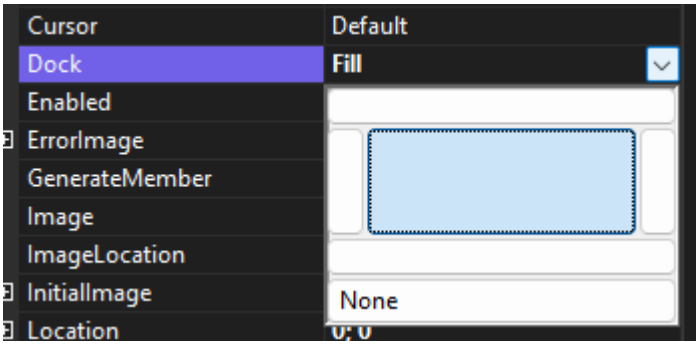
2. Называем его Paint:



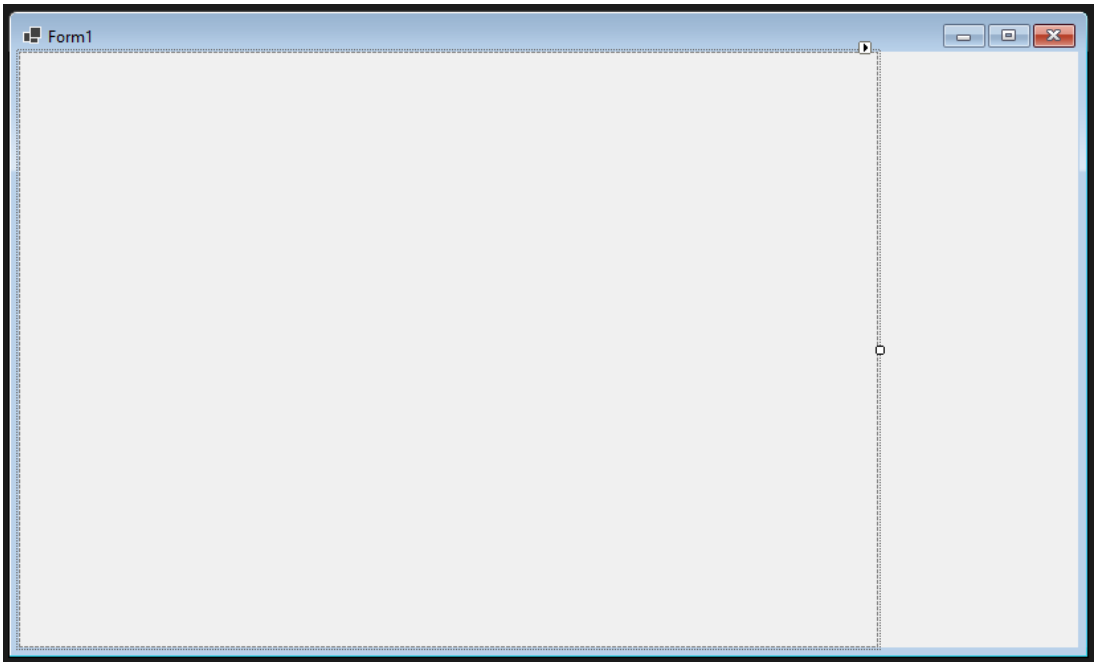
3. Размещаем на нашей форме **PictureBox**:



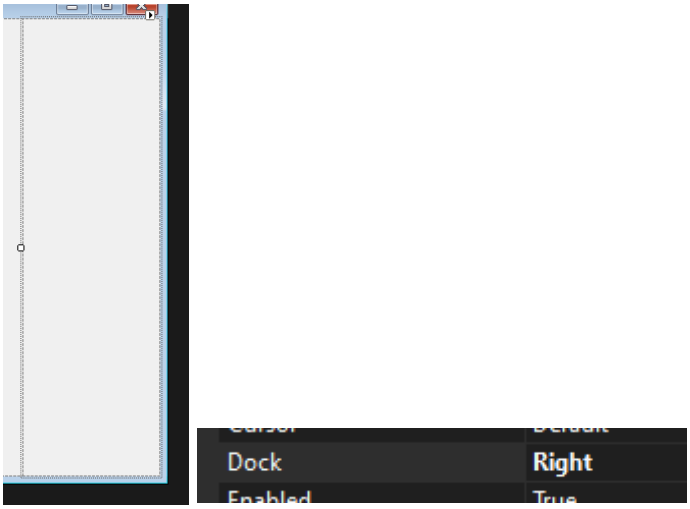
4. Привяжем его на весь экран - **Fill**:



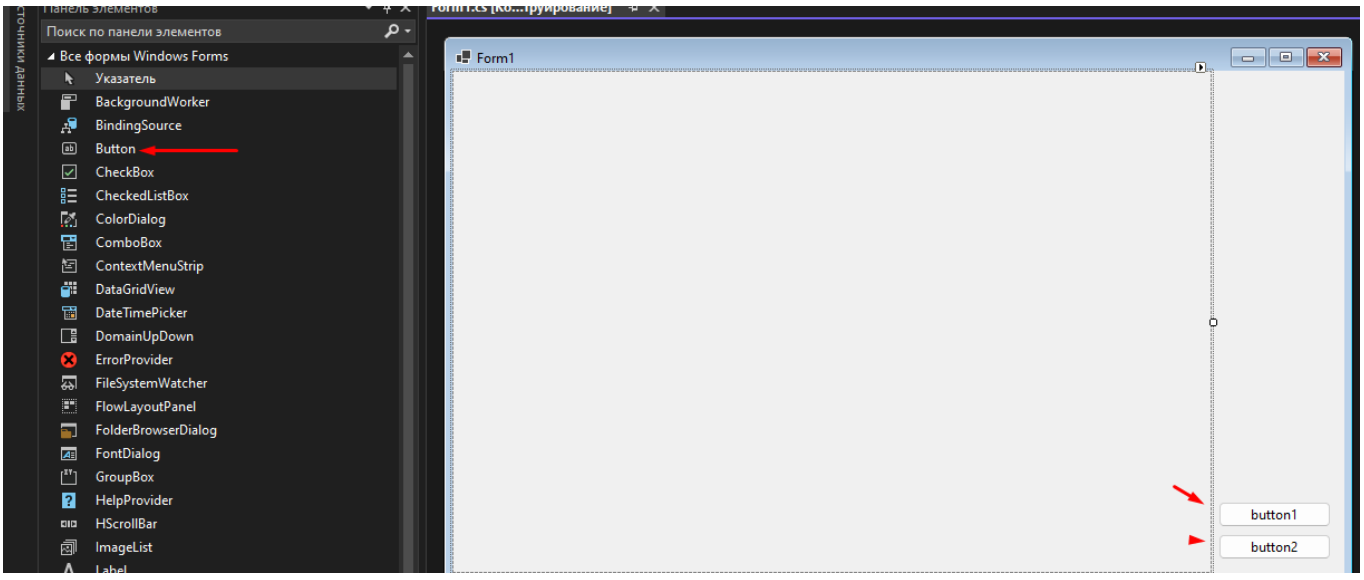
5. Растянем его побольше:



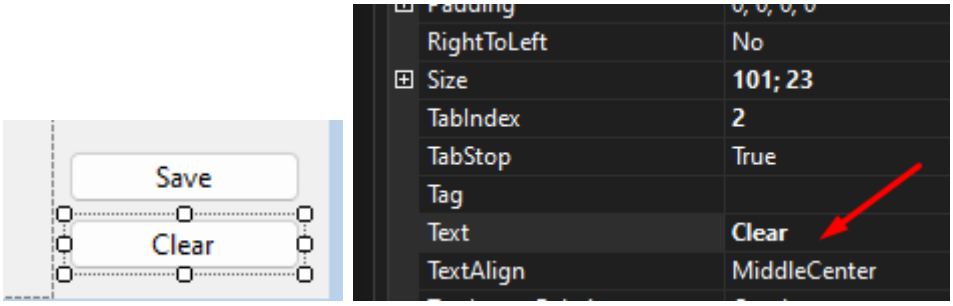
Теперь добавим новую панель **Panel**, и привяжем её к правому углу:



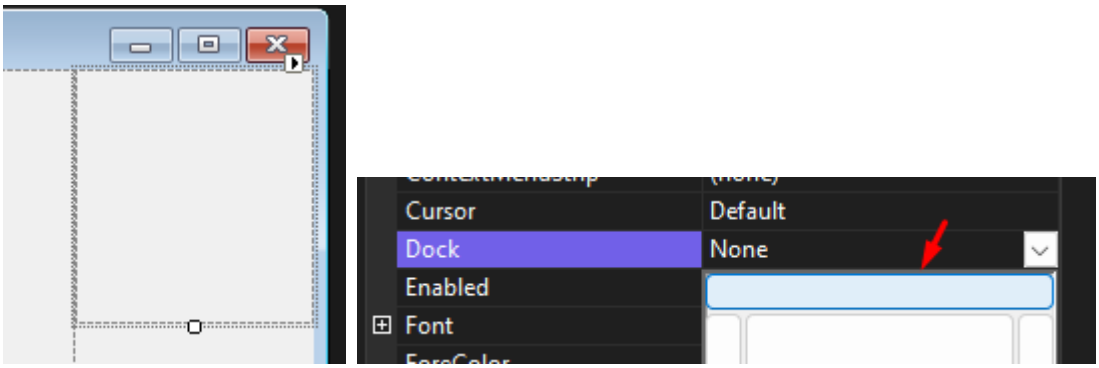
6. Добавим две кнопки, которые будут отвечать за сохранение и за очистку экрана:



7. Переименуем их как **Save** и **Clear** (в поле текст):

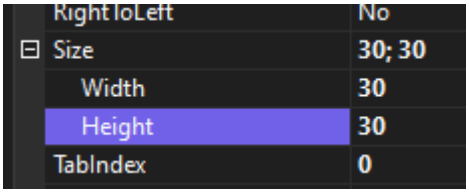


8. Добавим **Panel** и разместим по верхнему краю:

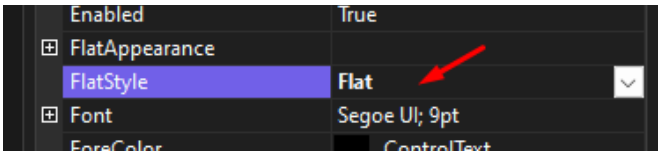


9. Внутрь добавим кнопки, отвечающие за смену цвета. Добавляем **Button**:

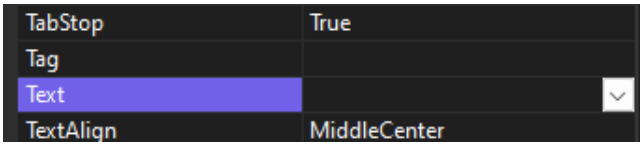
Меняем размер **30 x 30:**



FlatStyle поменяем на **Flat**:

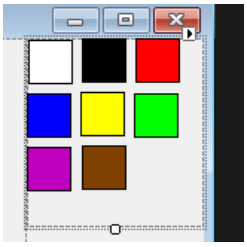


Убираем текст:

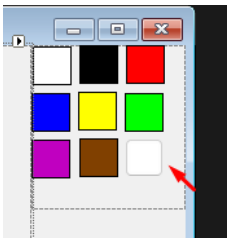


BackColor меняем на **White**:

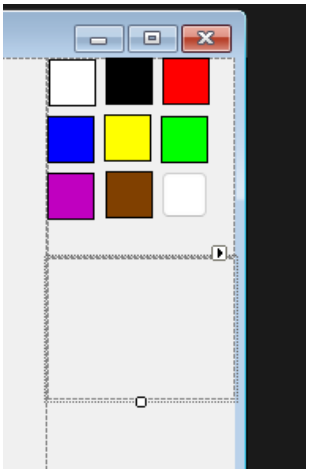
10. Через **Ctrl+C** скопируем, и через **Ctrl+V** вставим кнопку несколько раз, и поменяем новым кнопкам цвет:



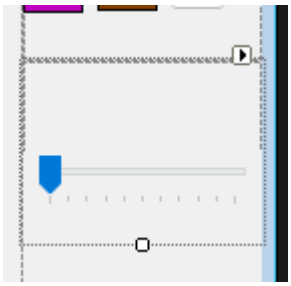
11. Разместим ещё одну кнопку, которая будет отвечать за открытие палитры цветов:



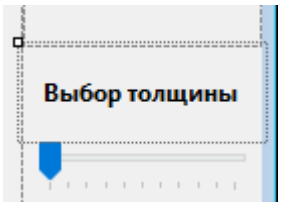
12. Добавим **Panel** и разместим по верхнему краю:



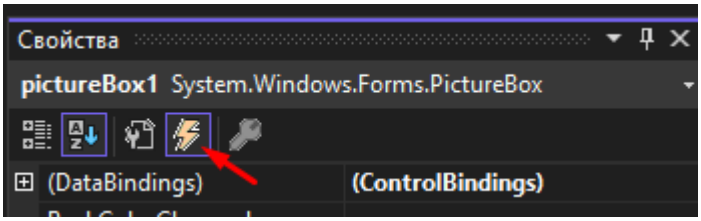
13. В неё добавим **TrackBar**, с помощью которого будем регулировать толщину карандаша**:**



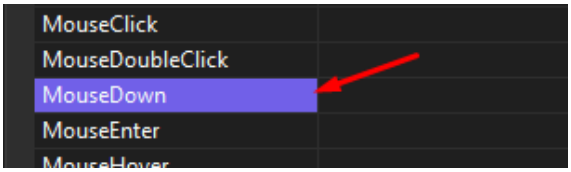
14. Также добавим **Label** в нашу панель, и напишем «**Выбор толщины**»:



15. Выбираем **PictureBox** и переходим во вкладку **События**:



Щёлкаем два раза по **MouseDown**:

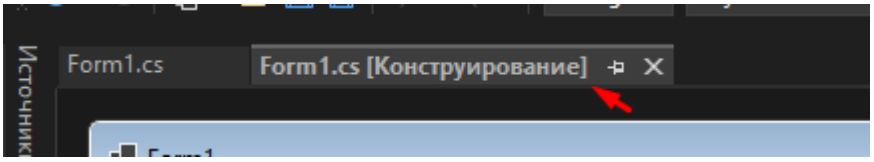


Для создания массива, нам нужно определить класс, который предназначен для хранения массива точечных объектов и управления им. Эти точечные объекты представляют координаты на двумерной поверхности, которые необходимы для рисования линий или фигур в приложении для рисования.

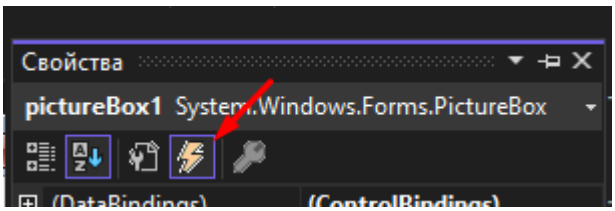
16. Добавляем булеву переменную **isMouse**, которая будет проверять зажата ли кнопка мыши в данный момент или нет. Она будет играть определенную роль в определении того, когда начинать или останавливать рисование. Когда мы нажимаем ЛКМ, то должны поменять переменную на **true**:

```
10          private bool isMouse = false;
            Ссылок: 1
11          private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
12          {
13              isMouse = true;
14          }
15      }
```

17. Теперь нужно добавить событие на отпускание мыши. Переходим в конструирование:



События:

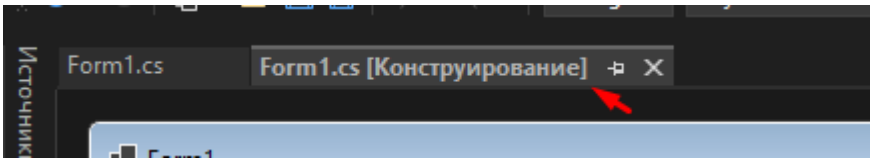


Щёлкаем два раза по **MouseUp**:

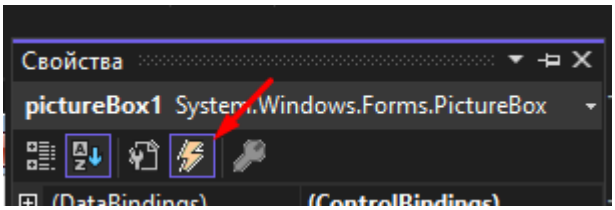
18. Меняем значение переменной **isMouse** на **false**:

```
Ссылок: 1
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    isMouse = false;
}
}
```

19. Также добавляем **MouseMove**:



События:



Щёлкаем два раза по **MouseMove**:

20. Для рисования мы будем использовать массив точек.

Массивы точек позволяют эффективно хранить и управлять координатами. Они используются для хранения координат линий, которые можно нарисовать в приложении. Использование класса позволяет инкапсулировать логику работы с точками и сделать код более организованным и легко поддерживаемым.

Создаём новый класс **ArrayPoints**. Этот класс будет инкапсулировать логику хранения массива точечных объектов и управления точками (координатами):

```
Ссылок: 0
private class ArrayPoints
{

}
```

Внутри класса **ArrayPoints** объявляем поля класса:

- **index**. Служит для отслеживания текущей позиции в массиве точек. Начальное значение равно 0.

- **points**. Массив точек, где каждая точка представлена объектом Point. В этом массиве будут храниться отдельные точечные объекты, представляющие координаты.

```
private class ArrayPoints
{
    private int index = 0;
    private Point[] points;
}
```

Создадим конструктор класса **ArrayPoints**. Инициализирует массив точек заданного размера. Для определения начальной емкости массива точек требуется целочисленный параметр **size**.

Если указанный размер меньше или равен 0, он устанавливается равным 2. Это предотвращает создание массива с недопустимым размером. Минимальный размер массива (2) позволяет хранить как минимум две точки, необходимые для рисования линии между двумя координатами.

Затем массив точек инициализируется указанным размером.

```
private class ArrayPoints
{
    private int index = 0;
    private Point[] points;

    Ссылок: 0
    public ArrayPoints(int size)
    {
        if (size <= 0) { size = 2; }
        points = new Point[size];
    }
}
```

Добавим метод **SetPoint**. Устанавливает точку с координатами (x, y) в массиве точек.

```
private class ArrayPoints
{
    private int index = 0;
    private Point[] points;

    Ссылок: 0
    public ArrayPoints(int size)
    {
        if (size <= 0) { size = 2; }
        points = new Point[size];
    }

    Ссылок: 0
    public void SetPoint(int x, int y)
    {

    }
}
```

- Если индекс превышает длину массива, он сбрасывается до 0, что позволяет перезаписывать старые точки.
- points[index] = new Point(x, y): Создает новую точку с координатами (x, y) и сохраняет её в массиве.
- index++: Увеличивает индекс на 1, чтобы перейти к следующей позиции в массиве.

```
public void SetPoint(int x, int y)
{
    if (index >= points.Length)
        index = 0;

    points[index] = new Point(x, y);
    index++;
}
```

Продолжаем работу с классом **ArrayPoints**, создадим метод **ResetPoints**, который сбрасывает индекс до 0, что позволяет начать запись точек заново:

```
public void ResetPoints()
{
    index = 0;
}
```

Пример использования: Представь, что ты рисуешь линию на экране мышкой. Когда ты отпускаешь кнопку мыши, текущая линия завершена, и, если ты снова начнешь рисовать, тебе нужно начать запись новых точек с начала массива. Метод ResetPoint помогает сбросить текущие координаты и начать заново с начала массива, чтобы не смешивать старые и новые точки.

Создадим метод, который будет получать размер нашего массива **GetCountPoints**, она будет возвращать наш **index:**

```
public int GetCountPoints()
{
    return index;
}
```

Пример использования: Когда ты рисуешь линию, тебе нужно знать, сколько точек ты уже записал, чтобы понять, готов ли ты рисовать линию между ними. Этот

метод позволяет получить текущее количество точек, чтобы решить, надо ли продолжать запись или уже можно начать рисование.

Далее создаём метод для возвращения массива наших точек **GetPoints**:

```
public Point[] GetPoints()
{
    return points;
}
```

Пример использования: Когда ты хочешь нарисовать линию на экране, тебе нужно знать координаты всех точек, между которыми ты будешь рисовать линии. Метод GetPoints позволяет получить весь массив точек, чтобы передать его методу рисования и соединить точки линиями.

Пример использования всех методов вместе

1. **Начало рисования:**
   1.1. Пользователь нажимает кнопку мыши (**MouseDown**): **ResetPoint**() сбрасывает индекс до 0, чтобы начать запись новых точек.
2. **Процесс рисования:**
   2.1. Пользователь перемещает мышь с нажатой кнопкой (**MouseMove**): **SetPoint**(x, y) добавляет новые точки в массив.
   2.2. **GetCountsPoints**() проверяет, достаточно ли точек для рисования линии.
   2.3. Если точек достаточно, **GetPoints**() возвращает массив точек для метода рисования, который рисует линии между точками.
3. **Окончание рисования:**
   3.1. Пользователь отпускает кнопку мыши (**MouseUp**): **ResetPoint**() снова сбрасывает индекс для нового рисования.

21. Теперь мы закончили с созданием класса **ArrayPoints,** для удобства можем его свернуть:

```
10  >   private class ArrayPoints...
42
```

22. Далее нам нужно создать:

- **arrayPoints**: Хранение координат точек для рисования линий.
- **map**: Растровое изображение, на котором будут выполняться все графические операции.

- **graphics**: Объект для выполнения графических операций на растровом изображении.
- **pen**: Определяет цвет и толщину линий, используемых для рисования.

Создадим объект класса **ArrayPoints** с размером массива точек 2:

```
private ArrayPoints arrayPoints = new ArrayPoints(2);
```

Зачем нужно: Создание экземпляра класса ArrayPoints с массивом для хранения точек. Размер массива установлен на 2, что позволяет хранить минимум две точки, необходимые для рисования линии между ними.

Почему размер 2: Это минимальный размер, достаточный для хранения координат начала и конца линии. Если бы размер был меньше 2, нельзя было бы нарисовать линию.

23. Создадим переменную, которая будет отвечать за хранение нашего растрового изображения **map,** с размером 100x100 пикселей:

```
Bitmap map = new Bitmap(100, 100);
```

Создадим также переменную **graphics** типа **Graphics**, которая будет использоваться для рисования (этот объект предоставляет методы для рисования линий, фигур и других графических элементов):

```
Graphics graphics;
```

Создадим объект **Pen** черного цвета с толщиной линии 3 пикселя. Этот объект будет использоваться для рисования линий:

```
Pen pen = new Pen(Color.Black, 3f);
```

24. В методе **pictureBox1_MouseMove** выполним проверку на зажатие левой кнопки мыши.

```
private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
{
    if (!isMouse) { return; } // Проверка состояния мыши: Если
переменная isMouse равна false (мышь не нажата), метод возвращает
управление без выполнения дальнейших действий.

    arrayPoints.SetPoint(e.X, e.Y); // Запись точки: Добавляет
текущие координаты мыши e.X (горизонтальная координата) и e.Y
```

```
(вертикальная координата) в массив точек. e — это объект типа
MouseEventArgs, который передается обработчику событий MouseMove.

    if (arrayPoints.GetCountsPoints() >= 2) // Проверка количества
точек: Если количество точек в массиве больше или равно 2,
продолжается выполнение кода внутри блока if.
    {
        graphics.DrawLines(pen, arrayPoints.GetPoints()); //
Рисование линий: Рисует линии между точками в массиве, используя
объект graphics и pen.
        pictureBox1.Image = map; // Обновление изображения:
Устанавливает изображение PictureBox в объект Bitmap, на котором
были нарисованы линии.
        arrayPoints.SetPoint(e.X, e.Y); // Добавление новой точки:
Снова записывает текущие координаты мыши, чтобы продолжить
рисование линии в следующем событии MouseMove.
    }
}
```

25. Реализуем метод **SetSize**, который будет отвечать за установку размера нашего изображения:

```
73      private void SetSize()
74      {
75
76      }
```

Размер возьмём из разрешения, установленного у пользователя. Также инициализируем наше изображение и графику:

```
        private void SetSize()
        {
            Rectangle rectangle = Screen.PrimaryScreen.Bounds; //
Возвращает размеры (границы) основного экрана. Это значение
хранится в объекте Rectangle под именем rectangle.
            map = new Bitmap(rectangle.Width, rectangle.Height); //
Создает новое растровое изображение (Bitmap) размером с основной
экран. Это изображение будет использоваться для рисования.
            graphics = Graphics.FromImage(map); // Создает объект
Graphics, который используется для рисования на объекте Bitmap
(map). Это позволяет выполнять графические операции (рисование,
заливка, и т.д.) на этом изображении.
        }
```

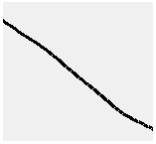
Добавим его вызов при инициализации нашей формы:

```
5       public Form1()
6       {
7           InitializeComponent();
8           SetSize();
9       }
```

26. В метод **pictureBox1_MouseUp** добавим вызов метода **ResetPoints** (для сброса массива точек)**:**

```
private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
{
    isMouse = false;
    arrayPoints.ResetPoints();
}
```

27. Теперь если вы запустите программу, то увидите, что рисование работает, но линия выглядит немного прерывистой:



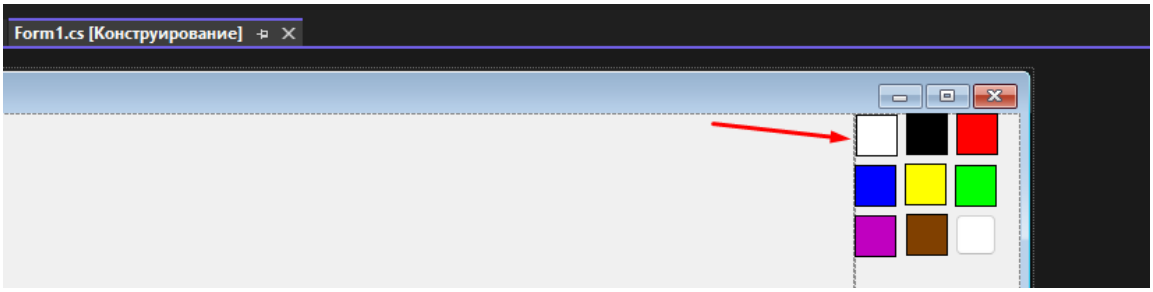
Решим эту проблему. В методе **SetSize** допишем код для округления линии:

```
private void SetSize()
{
    Rectangle rectangle = Screen.PrimaryScreen.Bounds;
    map = new Bitmap(rectangle.Width, rectangle.Height);
    graphics = Graphics.FromImage(map);

    pen.StartCap = System.Drawing.Drawing2D.LineCap.Round;
    pen.EndCap = System.Drawing.Drawing2D.LineCap.Round;
}
```

Устанавливает форму конца линии кончика пера и начала пера в Round (круглая форма).

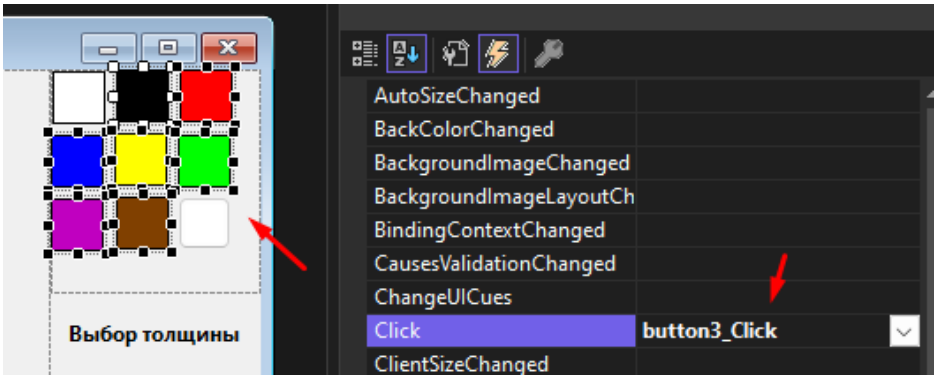
28. Реализуем возможность смены цвета пера. В нашей форме кликаем на цвет:



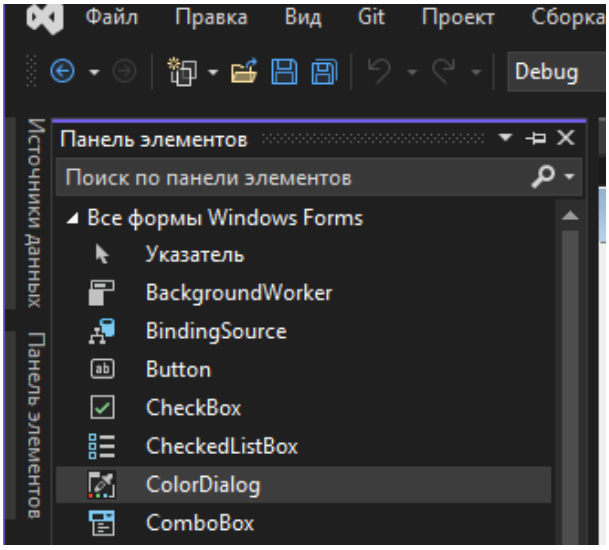
Напишем:

```
private void button3_Click(object sender, EventArgs e)
{
    pen.Color = ((Button)sender).BackColor;
}
```

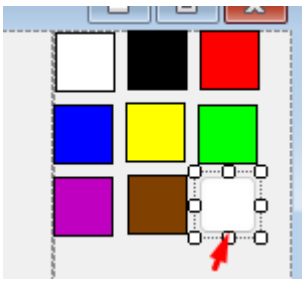
Затем нужно выбрать все кнопки с цветами, и на них добавить созданный нами метод нажатия:



29. Сделаем для последней кнопки выбор палитры. Для этого добавим на нашу форму **ColorDialog** перетаскиванием:



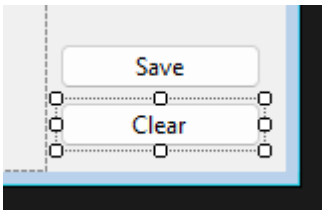
Щёлкаем по нашей кнопке:



Пропишем код, который будет вызывать диалоговое окно, и оставлять выбранный пользователем цвет:

```
private void button11_Click(object sender, EventArgs e)
{
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        pen.Color = colorDialog1.Color;
        ((Button)sender).BackColor = colorDialog1.Color;
    }
}
```

30. Реализуем кнопку **Clear**. Дважды щёлкаем по ней:



Очистим его цветом, который был по умолчанию в **pictureBox1** и присвоим ему изображение:

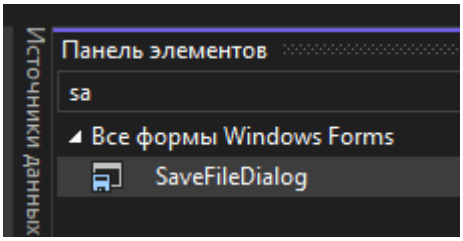
```
private void button2_Click(object sender, EventArgs e)
{
    graphics.Clear(pictureBox1.BackColor);
    pictureBox1.Image = map;
}
```

31. Сделаем изменение размера линии. Щёлкаем по нему:

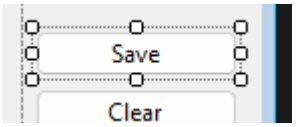
Пропишем:

```
Ссылок: 1
private void trackBar1_Scroll(object sender, EventArgs e)
{
    pen.Width = trackBar1.Value;
}
```

32. Реализуем кнопку **Save**. Добавим на нашу форму **SaveFileDialog**:



Дважды щёлкаем по кнопке:



Создадим фильтр, который будет позволять сохранять нашу картинку в формате **jpg**:

```
private void button1_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "JPG(*.JPG)|*.jpg";
}
```

Пропишем условие, что при нажатии ОК, будет сохраняться наша картинка:

```
private void button1_Click(object sender, EventArgs e)
{
    saveFileDialog1.Filter = "JPG(*.JPG)|*.jpg";
    if(saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        if (pictureBox1.Image == null)
            pictureBox1.Image.Save(saveFileDialog1.FileName);
    }
}
```

33. Сохраните вашу работу.

Самостоятельные задания

Создайте модернизированный кликер.

Пример реализации: