

Практическая работа №11

Цель работы: получение навыков написания простейшего десктопного приложения на языке C#.

Теоретическая часть

Шаг 1

Создайте новый проект с именем ImageEditor. Тип проекта – Windows Forms App.

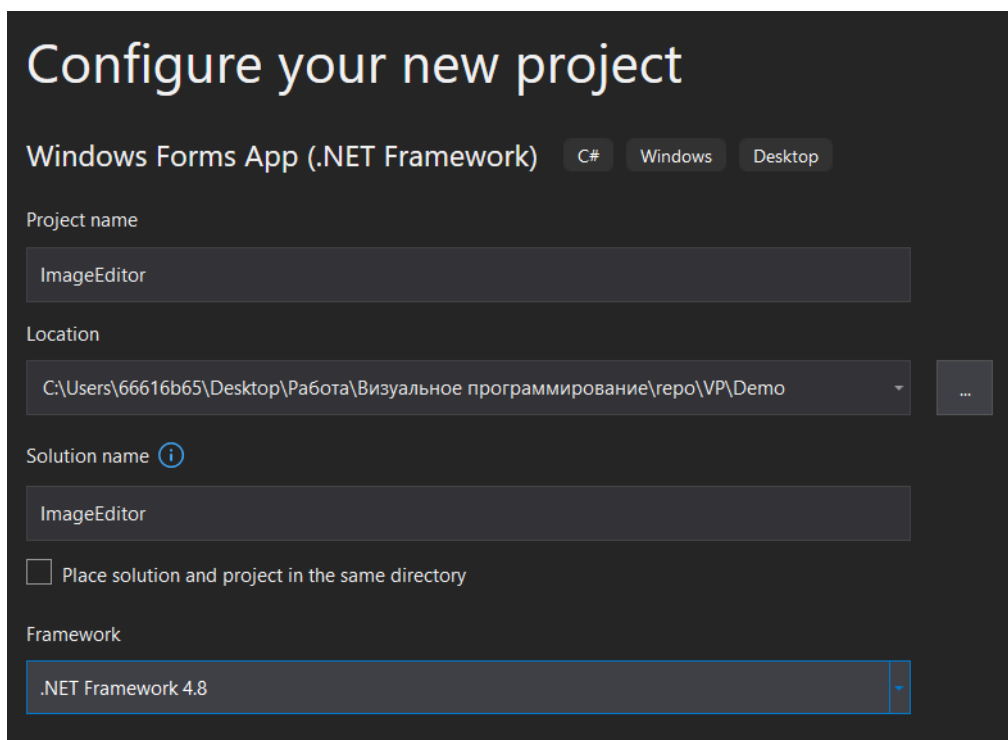


Рисунок 1 – Создание проекта

Шаг 2

После создания проекта откроется окно с формой. При разработке очень важно задавать имена элементам управления в зависимости от их назначения, а не оставлять имена по умолчанию. Поэтому переименуйте **Form1**, как это показано на картинке. За текст в левом верхнем углу отвечает свойство **Text**.

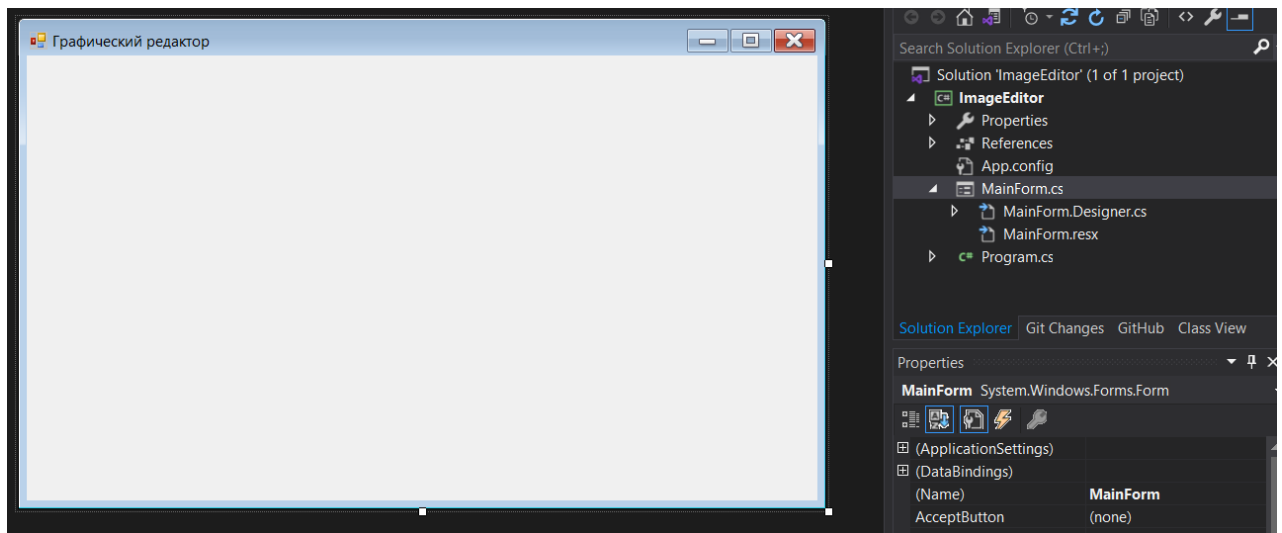


Рисунок 2 – Переименование формы

Шаг 3

Зададим значения следующих свойств формы:

- **Size: 664; 448** – для красоты
- **StartPosition: CenterScreen** – при загрузке форма будет отображаться по центру экрана

Все изменения свойств сохраняются в файле **MainForm.Designer.cs**, а именно внутри метода **InitializeComponent()**. Этот метод вызывается в конструкторе формы.

Обратите внимание на заголовок класса, а именно **partial class MainForm**. Этот класс является разделяемым, то есть его код лежит в разных файлах, но при этом это один единый класс. Это делается для удобства, чтобы разделить разработанную программистом логику от отрисовки и автоматической привязки обработчиков событий.

Если выбрать файл **MainForm.cs** и нажать F7 (или кнопку View Code в контекстном меню), то откроется другая часть класса **MainForm**, в котором мы будем писать логику работы.

```

partial class MainForm
{
    /// <summary> Required designer variable.
    private System.ComponentModel.IContainer components = null;

    /// <summary> Clean up any resources being used.
    0 references
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Windows Form Designer generated code

    /// <summary> Required method for Designer support - do not modify the contents ...
    1 reference
    private void InitializeComponent()
    {
        this.SuspendLayout();
        ///
        /// MainForm
        ///
        this.AutoScaleMode = new System.Drawing.SizeF(8F, 16F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(646, 401);
        this.Name = "MainForm";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "Графический редактор";
        this.ResumeLayout(false);
    }

    #endregion
}

```

Рисунок 3 – Файл MainForm.Designer.cs

```

public partial class MainForm : Form
{
    1 reference
    public MainForm()
    {
        InitializeComponent();
    }
}

```

Рисунок 4 – Файл MainForm.cs

Шаг 4

Добавим на форму меню, где будут располагаться кнопки для работы с изображением. Для этого в окне **Toolbox** нужно выбрать контрол (элемент управления) **MenuStrip** и перетащить его на форму. Свойство **Name** также зададим **menuStrip**.

Добавьте пункты для работы с файлом:

- `fileMenuItem`
- `openFileMenuItem`
- `saveAsMenuItem`
- `exitMenuItem`

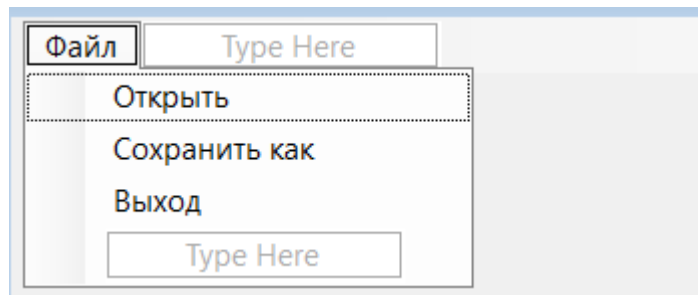


Рисунок 5 – Меню для действий с файлом

И также пункты меню для работы с изображением:

- `imageMenuItem`
- `filterMenuItem`

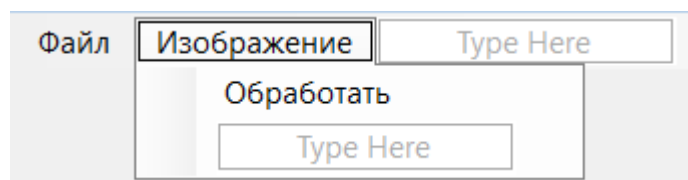


Рисунок 6 – Меню для действий с изображением

Шаг 5

Добавим контрол для отображения изображения `PictureBox`. Зададим значения свойств:

- `Name: pictureBox`
- `Location: 12; 42`

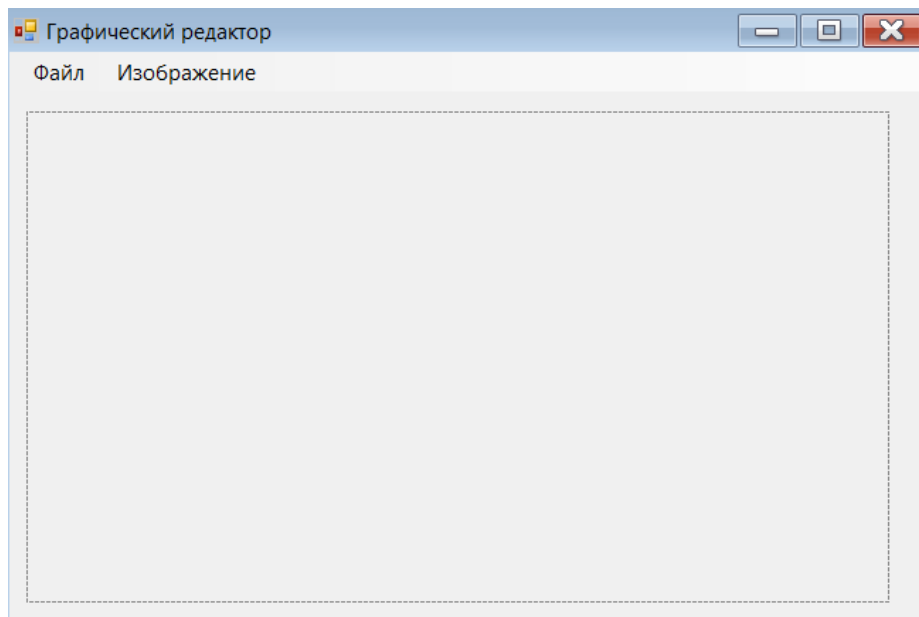


Рисунок 7 – Добавление PictureBox

Шаг 6

Добавим на форму диалоги для открытия и закрытия файла (контролы OpenFileDialog и SaveFileDialog). Они не будут отображаться на самой форме, но будут видны снизу. Также нужно задать имя для каждого контрола, как на картинке.

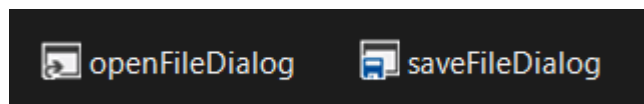


Рисунок 8 – Контролы для файлового диалогового окна

Шаг 7

Объявим поле для хранения изображения.

```
private Bitmap _sourceImage;
```

Также сделаем, чтобы при загрузке формы **pictureBox** и кнопка меню **Изображение** были недоступны.

```
public MainForm()  
{  
    InitializeComponent();  
    pictureBox.Visible = false;  
    imageMenuItem.Visible = false;  
}
```

Шаг 8

Теперь добавим обработчики для работы с кнопками меню. Это можно сделать вручную (написать метод и добавить в список обработчиков события **Click** нужной кнопки), а можно автоматически (дважды щёлкнуть по самой кнопке (автоматически добавится обработчик для **Click**) или выбрать нужное событие в списке **Events** конкретного контрола).

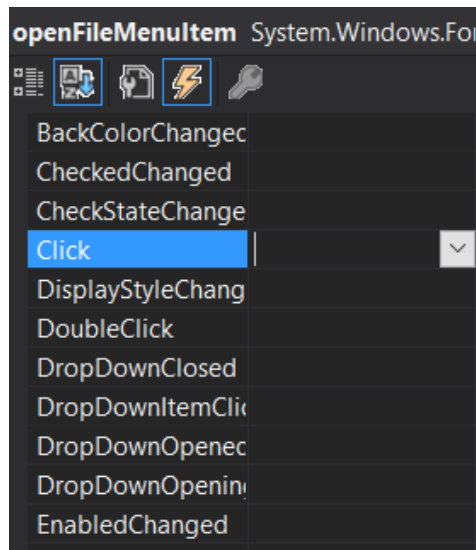


Рисунок 9 – События для контрола

При автоматическом создании метода в файле **MainForm.cs** появится метод, а в файле **MainForm.Designer.cs** – подписка на событие.

```
public partial class MainForm : Form
{
    1 reference
    ... public MainForm()
    ... {
    ...     InitializeComponent();
    ... }

    1 reference
    ... private void openFileMenuItem_Click(object sender, EventArgs e)
    ... {
    ... }
}
```

Рисунок 10 – Файл MainForm.cs

```
// openFileMenuItem
//
this.openFileMenuItem.Name = "openFileMenuItem";
this.openFileMenuItem.Size = new System.Drawing.Size(224, 26);
this.openFileMenuItem.Text = "Открыть";
this.openFileMenuItem.Click += new System.EventHandler(this.openFileMenuItem_Click);
```

Рисунок 11 – Файл MainForm.Designer.cs

Добавим код для загрузки изображения.

```
private void openFileMenuItem_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "Изображения(*.bmp;*.jpeg;*.jpg)|*.bmp;*.jpeg;*.jpg";
    openFileDialog.ShowDialog();
    _sourceImage = new Bitmap(openFileDialog.FileName);
    pictureBox.Size = _sourceImage.Size;
    this.Width = pictureBox.Width + 40;
    this.Height = pictureBox.Height + 77;
    this.CenterToScreen();
    pictureBox.Image = _sourceImage;
    pictureBox.Visible = true;
    imageMenuItem.Visible = true;
}
```

При открытии файлового диалога можно выбрать только файлы с определенным расширением. Изображение передается в `pictureBox`, размер формы автоматически изменяется, становятся доступными кнопки для работы с изображением.

Также добавим метод для сохранения файла. Для сохранения будут доступны только форматы `.bmp` и `.jpg`. Необходимо подключить namespace `System.IO` и `System.Drawing.Imaging`.

```
private void saveAsMenuItem_Click(object sender, EventArgs e)
{
    saveFileDialog.Filter = "Изображение BMP|*.bmp|Изображение JPEG|*.jpeg|Изображение JPG|*.jpg";
    saveFileDialog.ShowDialog();
    if (saveFileDialog.FileName != "")
    {
        using (FileStream fs = (FileStream)saveFileDialog.OpenFile())
        {
            switch (saveFileDialog.FilterIndex)
            {
                case 1:
                    pictureBox.Image.Save(fs, ImageFormat.Bmp);
                    break;
                case 2:
                case 3:
                    pictureBox.Image.Save(fs, ImageFormat.Jpeg);
                    break;
            }
        }
    }
}
```

```
}
```

И добавим обработчик для кнопки Выйти. В данном случае приложение просто закрывается.

```
private void exitMenuItem_Click(object sender, EventArgs e)
{
    Close();
}
```

В обработчике для кнопки Обработать будем вызывать метод, который разработаем на следующем шаге.

```
private void filterMenuItem_Click(object sender, EventArgs e)
{
    var resultBitmap = ImageProcess.FilterImage(_sourceImage);
    pictureBox.Image = resultBitmap;
}
```

Шаг 9

Добавим в приложение статический класс **ImageProcess**. Подключим пространство имен **System.Drawing**.

Далее над изображением будут выполняться различные математические преобразования. В данном примере рассмотрим фильтр для повышения резкости изображения. Для этого нам нужно преобразовать матрицу изображения в одномерный массив, произвести математические преобразования и снова перевести изображение в двумерный вид.

Следующий метод будет преобразовывать изображение в одномерный массив. Причём отдельно выделяются red, green и blue компоненты пикселя.

```
static byte[] BmpToArray(Bitmap source)
{
    var result = new byte[source.Width * source.Height * 3];
    for (int i = 0; i < source.Width; i++)
    {
        for (int j = 0; j < source.Height; j++)
        {
            Color pixel = source.GetPixel(i, j);
            result[3 * (j * source.Width + i) + 0] = pixel.R;
            result[3 * (j * source.Width + i) + 1] = pixel.G;
            result[3 * (j * source.Width + i) + 2] = pixel.B;
        }
    }
    return result;
}
```


Теперь разработаем метод для обратного преобразования одномерного массива в изображение.

```
static Bitmap ArrayToBmp(byte[] source, int width, int height)
{
    var result = new Bitmap(width, height);
    for (int i = 0; i < width; i++)
    {
        for (int j = 0; j < height; j++)
        {
            int position = 3 * (j * width + i);
            result.SetPixel(i, j, Color.FromArgb(source[position + 0],
                                                    source[position + 1],
                                                    source[position + 2]));
        }
    }
    return result;
}
```

В ходе математических преобразований значение пикселя может выходить за пределы допустимого значения (0..255), поэтому для предотвращения этого разработаем специальный метод.

```
static int RgbRange(int value)
{
    if (value < 0)
    {
        value = 0;
    }
    else
    {
        if (value > 255)
        {
            value = 255;
        }
    }
    return value;
}
```

Для фильтрации с целью повышения резкости будем использовать следующее ядро фильтра:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Рисунок 12 – Ядро фильтра

Метод **FilterImage** выполняет математические преобразования над исходным изображением и возвращает изображение обработанное.

```
public static Bitmap FilterImage(Bitmap source)
{
    byte[] src = BmpToArray(source);
    byte[] res = new byte[src.Length];

    int[,] matrix = new int[3, 3] { { 0, -1, 0 }, { -1, 5, -1 }, { 0, -1, 0 } };
    for (int i = 0; i < source.Width; i++)
    {
        for (int j = 0; j < source.Height; j++)
        {
            var r = 0;
            var g = 0;
            var b = 0;
            for (int n = 0; n < 3; n++)
            {
                for (int m = 0; m < 3; m++)
                {
                    if (((j - 1 + m) < 0) || ((j - 1 + m) == source.Height)
                        || ((i - 1 + n) < 0) || ((i - 1 + n) == source.Width))
                    {
                        continue;
                    }

                    int position = 3 * (source.Width * (j - 1 + m) + (i - 1 +
n));

                    int matrixValue = matrix[n, m];

                    r += src[position + 0] * matrixValue;
                    g += src[position + 1] * matrixValue;
                    b += src[position + 2] * matrixValue;
                }
            }
            r = RgbRange(r);
            g = RgbRange(g);
            b = RgbRange(b);

            int pixelPosition = 3 * (source.Width * j + i);
            res[pixelPosition + 0] = (byte)r;
            res[pixelPosition + 1] = (byte)g;
            res[pixelPosition + 2] = (byte)b;
        }
    }
    return ArrayToBmp(res, source.Width, source.Height);
}
```

Шаг 10

Проверим работу приложения.

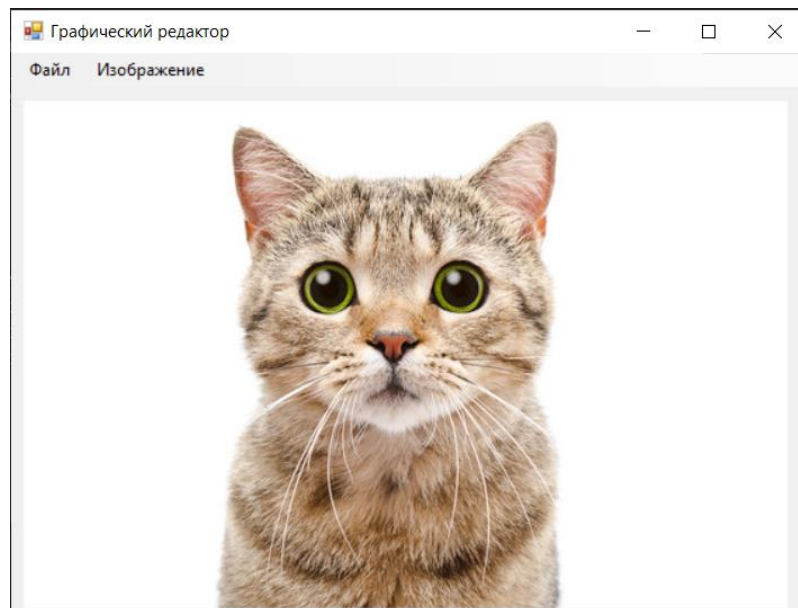


Рисунок 13 – Исходное изображение

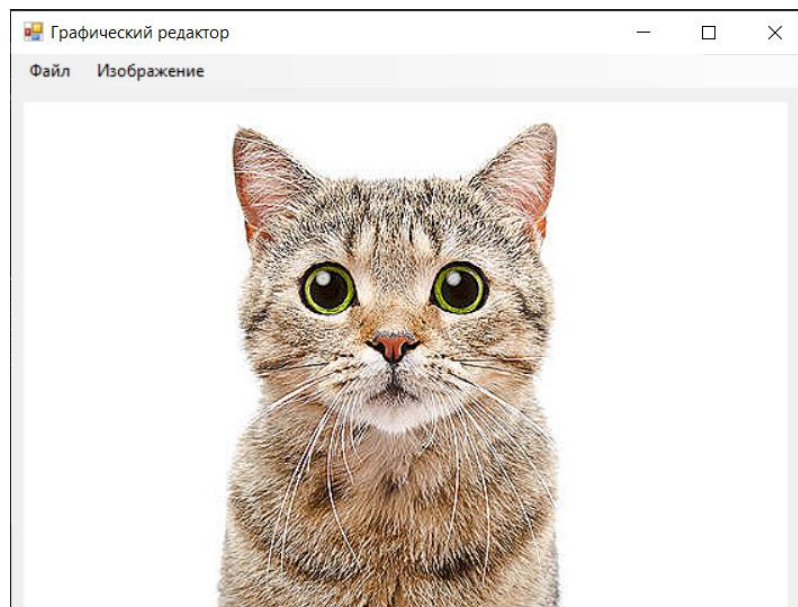


Рисунок 14 – Обработанное изображение

Задание

Требуется добавить в редактор функции:

- сделать эффект блюра (Гауссово размытие);
- усилить край изображения (по Лапласу);
- отменить или вернуть изменения (кнопки undo и redo, Ctrl+Z, Ctrl+Y);

- сохранить изменения в текущем изображении (кнопка Save, Ctrl+S).