

Лекция 5 ИСПОЛЬЗОВАНИЕ ДИАЛОГОВАХ МЕНЮ

5.1 Обработчики событий для работы с матрицей

Продолжая разработку приложения с меню и инструментальной панелью, нам необходимо написать код обработчиков сообщений для команд создания матрицы 6*6 и вывод (печать) матрицы в клиентскую область нашего приложения.

Создание матрицы необходимо заканчивать выводом на экран сообщения об успешном окончании работы обработчика, например, «Матрица создана».

Результатом метода «Печать матрицы» является сама матрица, поэтому дополнительных сообщений не требуется.

Исходный код программы будет рассматриваться фрагментами по мере наполнения обработчиков событий.

Начальный исходный код программы:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public static int[,] a = new int[6, 6];
        public Form1()
        {
            InitializeComponent();
        }

        private void sozMToolStripMenuItem_Click(object sender,
            EventArgs e)
        {
            Random rnd = new Random();
            for (int i = 0; i < 6; i++)
                for (int j = 0; j < 6; j++)
                    a[i, j] = rnd.Next() % 90 + 10;
            richTextBox1.AppendText("Матрица создана \n");
        }

        private void printToolStripMenuItem_Click(object sender,
            EventArgs e)
        {
            string ss;
            richTextBox1.AppendText("\n");
        }
    }
}
```

```

for (int i = 0; i < 6; i++)
{
    ss = "";
    for (int j = 0; j < 6; j++)
        ss = ss + Convert.ToString(a[i, j]) + "\t";
    richTextBox1.AppendText(ss + "\n");
}
}

```

Код остальных обработчиков событий пока «пустой».

5.2 Обработчик событий для открытия файла

Следующей по очереди инструментальной панели находится кнопка открытия файла – команда LoadT. Рассмотрим порядок действий при написании этого обработчика события.

Для того чтобы реализовать в нашем приложении обработчика открытия файла, нам потребуется элемент OpenFileDialog. Перетащим значок этого элемента из окна Toolbox в окно нашей формы. Значок элемента OpenFileDialog1 появится внизу на панели под нашей формой (смотри рисунок 5.1).

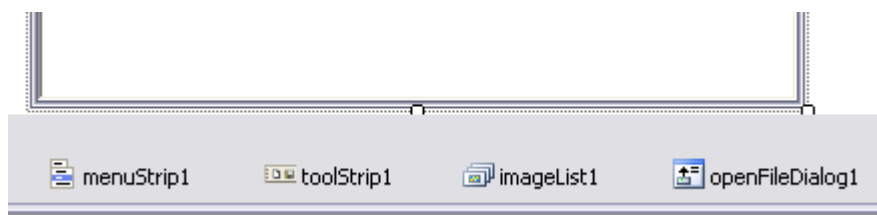


Рисунок 5.1 – Подключение элемента OpenFileDialog

Использование элемента OpenFileDialog (особенно в части диалога) это целая технология программирования, позволяющая просто обращаться к дискам, папкам и файлам нашего компьютера. Как и любой элемент окна Toolbox он представлен классом, имеющим множество методов для работы с файлами, например, метод openFileDialog1.ShowDialog, отображает на экране стандартное диалоговое окно выбора файла (смотри рисунок 5.2), в котором можно «путешествовать» по компьютеру в поисках нужного файла.

Настраивая свойства элемента openFileDialog1, можно задать фильтр имен открываемых файлов. Для этого необходимо элементу openFileDialog1 в окне Properties изменить свойство Filter. Присвойте этому свойству следующую текстовую строку:

Text files|*.txt|RTF files|*.rtf| All files|*.*

Строка фильтра состоит из блоков, разделенных символом |. Первый блок задает название типа файла Text files, а второе — маску для имен файлов. Для текстовых файлов применяется маска *.txt.

Далее следует название формата RTF files. Для RTF используется маска *.rtf.

И, наконец, чтобы приложение могло открывать файлы любых типов (All files), используется маска *.* (пример на рисунке 5.2).

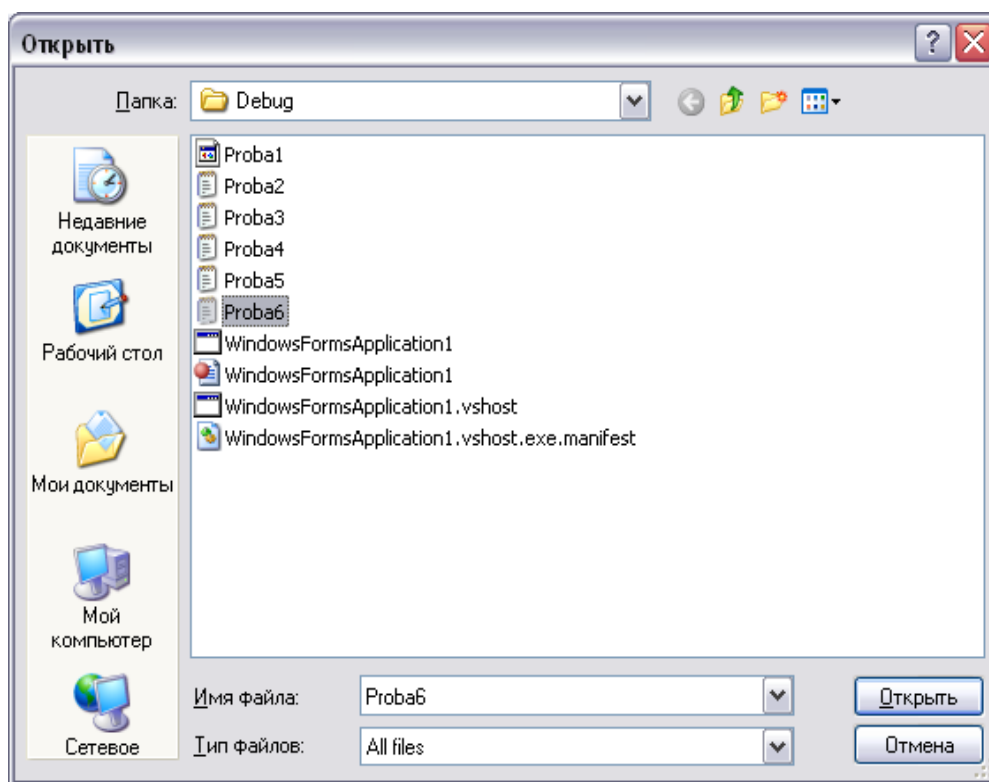


Рисунок 5.2 – Работа программы при открытии диалогового окна

По условию задачи мы будем работать с текстовыми файлами, но можно посмотреть, что читается при выборе файлов другого типа.

Дополнение к исходному коду программы:

```
private void loadToolStripMenuItem_Click(object sender,
EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK &&
        openFileDialog1.FileName.Length > 0)
    {
        try
        {
            richTextBox1.LoadFile(openFileDialog1.FileName,
                RichTextBoxStreamType.PlainText);
        }
        catch (System.ArgumentException ex)
        {
            richTextBox1.LoadFile(openFileDialog1.FileName,
                RichTextBoxStreamType.RichText);
        }
        this.Text = "Файл [" + openFileDialog1.FileName + "];
    }
}
```

Если в окне выбора файла пользователь щелкнул кнопку Открыть, метод ShowDialog возвращает значение DialogResult.OK. В условие включена дополнительная проверка, что файл был выбран (длина строки полного пути к выбранному файлу openFileDialog1.FileName.Length должна быть больше нуля).

При открытии файла методом richTextBox1.LoadFile ему передается два параметра. В качестве первого параметра этому методу передается путь к файлу, а в качестве второго — тип файла.

Поскольку мы должны работать с текстовыми файлами, то основным типом является RichTextBoxStreamType.PlainText.

В том случае, если выбранный файл имеет формат, отличный от PlainText в методе LoadFile возникает исключение System.ArgumentException и наш обработчик выполняет повторную попытку загрузить файл, но на этот раз уже как файл типа RichText. Этот тип соответствует формату RTF.

Дополнительно для информации полный путь к открытому файлу будет отображен в заголовке главного окна нашего приложения.

5.3 Обработчик событий для записи в файл

Следующей по очереди инструментальной панели находится кнопка записи клиентской области приложения в файл – команда SaveT. Рассмотрим порядок действий при написании этого обработчика события.

Для того чтобы реализовать в нашем приложении обработчика записи в файл, нам потребуется элемент SaveFileDialog. Перетащим значок этого элемента из окна Toolbox в окно нашей формы. Значок элемента SaveFileDialog1 появится внизу на панели под нашей формой.

Настраивая свойства элемента SaveFileDialog1, нужно отредактируйте его свойства Filter и FileName.

Свойству Filter необходимо указать тип тестового файла – Text files|*.txt, а свойству FileName шаблон имени документа – doc1.txt. При этом по умолчанию документы будут сохраняться в файле с этим именем.

Дополнение к исходному коду программы:

```
private void saveTToolStripMenuItem_Click(object sender,
    EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK &&
        saveFileDialog1.FileName.Length > 0)
    {
        richTextBox1.SaveFile(saveFileDialog1.FileName,
            RichTextBoxStreamType.PlainText);
        this.Text = "Файл [" + saveFileDialog1.FileName + "];"
    }
}
```

Дополнительно для информации полный путь к записываемому файлу будет отображен в заголовке главного окна нашего приложения.

Второй параметр метода `richTextBox1.SaveFile` позволяет выбрать тип сохраняемого файла. В нашем примере программа настроена на работу только с текстовыми файлами. Если передать через этот параметр значение `RichTextBoxStreamType.RichText`, то документ будет сохранен в формате RTF.

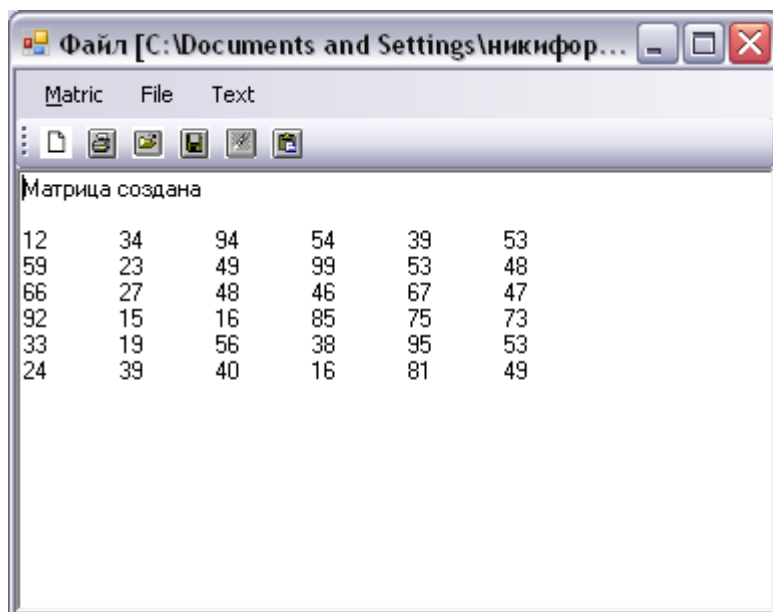


Рисунок 5.3 – Пример работы приложения с текстом

5.4 Обработчик событий для работы с текстом

Для работы с текстом клиентской области нашего приложения предусмотрено два обработчика событий, соответствующих командам меню очистка и преобразование.

Обработчик события очистка текста содержит один метод редактора текстов `richTextBox1`, который удаляет все строки текста:

```
private void clearToolStripMenuItem_Click(object sender,
    EventArgs e)
{
    richTextBox1.Clear();
}
```

Обработчик события преобразования текста предназначен для выделения из текста клиентской области значений матрицы 6*6, обычно считанной из файла.

Исходный код метода обработки текста имеет следующий вид:

```
private void obraboToolStripMenuItem_Click(object sender,
    EventArgs e)
{
    string ss;
    string[] clova;
```

```

int k, n;
int[] masi = new int[100];
n = 0;
ss = richTextBox1.Text;
clova = ss.Split('\t', ' ');
for (int i = 0; i < clova.Length; i++)
{
    k = clova[i].Length;
    if (k == 2 || k == 3)
    {
        masi[n] = Int32.Parse(clova[i]); n++; }
        richTextBox1.AppendText(i.ToString() + " = " + clova[i] +
            " k= " + k.ToString() + "\n");
    }
}
int j1, j2; j1 = j2 = 0;
for (int i = 0; i < 36; i++)
{
    a[j1, j2] = masi[i];
    j2++;
    if (j2 == 6) { j1++; j2 = 0; }
    if (j1 == 6 && j2 == 6) break;
}
}

```

Здесь необходимы некоторые комментарии.

Объявлен массив `clova` типа `string`, в который из текста клиентской области приложения с помощью метода `Split` заносятся любые сочетания символов выделенных знаком табуляции, пробелом или возвратом :

```

ss = richTextBox1.Text;
clova = ss.Split('\t', ' ', '\n');

```

Определяется общее количество слов текста `clova.Length`.

В цикле все слова длиной 2 или 3 символа преобразуются в целые числа и записываются в массив целых чисел `masi`. Одновременно все слова и их длина выводятся на экран в клиентскую область (для контроля).

В последней части обработчика события выделенные целые числа из массива переписываются в матрицу 6*6.

Примерный вид содержимого клиентской области:

Матрица создана

12	34	94	54	39	53
59	23	49	99	53	48
66	27	48	46	67	47
92	15	16	85	75	73
33	19	56	38	95	53
24	39	40	16	81	49

0 = Матрица k= 7

1 = создана k= 7

2 = k= 0

3 = k= 0

4 = 12 k= 2

5 = 34 k= 2
6 = 94 k= 2
7 = 54 k= 2
8 = 39 k= 2
9 = 53 k= 2
10 = k= 0
11 = 59 k= 2
12 = 23 k= 2
13 = 49 k= 2
14 = 99 k= 2
15 = 53 k= 2
16 = 48 k= 2
17 = k= 0
18 = 66 k= 2
19 = 27 k= 2
20 = 48 k= 2
21 = 46 k= 2
22 = 67 k= 2
23 = 47 k= 2
24 = k= 0
25 = 92 k= 2
26 = 15 k= 2
27 = 16 k= 2
28 = 85 k= 2
29 = 75 k= 2
30 = 73 k= 2
31 = k= 0
32 = 33 k= 2
33 = 19 k= 2
34 = 56 k= 2
35 = 38 k= 2
36 = 95 k= 2
37 = 53 k= 2
38 = k= 0
39 = 24 k= 2
40 = 39 k= 2
41 = 40 k= 2
42 = 16 k= 2
43 = 81 k= 2
44 = 49 k= 2
45 = k= 0
46 = k= 0

После визуальной проверки клиентскую область можно очистить.

Далее запускаем режим печати матрицы (смотри рисунок 5.4):

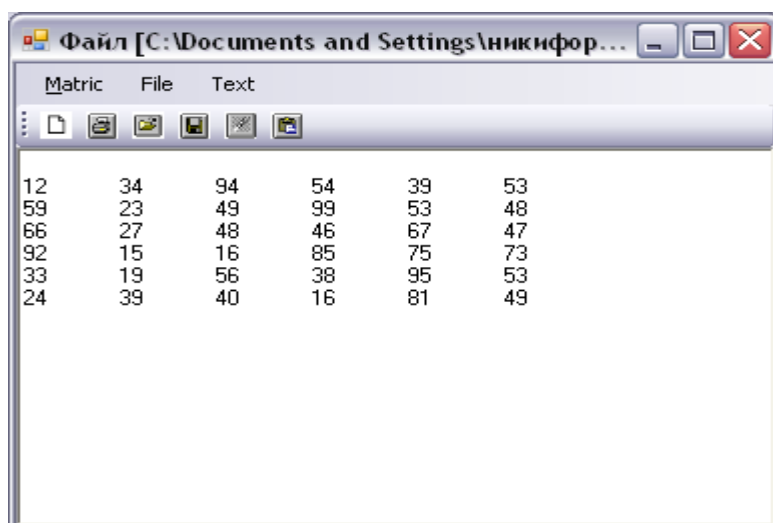


Рисунок 5.4 – Печать матрицы после преобразования клиентской области текста.

Видно, что нет привычного сообщения «Матрица создана».