

Лекция 6 МНОГОДОКУМЕНТНЫЕ ПРИЛОЖЕНИЯ

6.1 Создание «кнопочной» главной формы

Рассмотренные в предыдущих лекциях приложения были посвящены созданию однооконных приложений – все элементы управления и вывода результатов работы программ размещались в одном окне главной формы. В этой лекции мы рассмотрим технологию проектирования приложений с многооконным интерфейсом документов (Multiple-document interface, MDI) – многодокументных приложений.

Во многих многодокументных приложениях используется так называемая главная кнопочная форма – главная форма приложения, на которой меню реализовано в виде «больших» кнопок с необходимыми комментариями.

В каких ситуациях имеет смысл проектировать главную форму как главную кнопочную форму? Так поступают достаточно часто. Представьте себе, что создаваемый проект предоставляет конечному пользователю несколько различных сервисов, и пользователь, начиная работу с проектом, выбирает нужный ему сервис. Главная форма может иметь меню, команды которого и позволяют пользователю выбирать нужный ему сервис. Если каждый сервис достаточно сложен и требует собственного интерфейса, то в таких ситуациях вместо стандартного меню удобнее использовать главную кнопочную форму. Роль команд меню в ней играют расположенные в форме командные кнопки. Выбор командной кнопки на форме соответствует выбору команды меню.

Освоение материала новой технологии программирования всегда понятнее при решении некоторой учебной задачи – задачи, при решении которой упор делается на освоение новой технологии, а не на алгоритм решения самой задачи. В таких учебных задачах алгоритм ее решения должен быть очевидным или рассмотрен в предыдущих лекциях.

Задача 6.1 Разработать приложение, позволяющее создавать массив из 6 геометрических фигур типа прямоугольник. Прямоугольники задаются координатами двух противоположных вершин. Координаты вершин формируются случайным образом в диапазоне целых чисел от минус 100 до 100. Пользовательский интерфейс должен содержать 3 формы. Главная «кнопочная» форма, на которой должны располагаться кнопки режимов работы программы с соответствующей поясняющей информацией. Окно формы для табличного представления и редактирования информации. Окно формы для графического представления информации.

Таким образом, главное окно должно содержать 3 кнопки режимов работы программы:

- создание массива прямоугольников;
- переход на окно табличной формы представления и редактирования значений координат вершин прямоугольников;

- переход к окну с графической формой представления прямоугольников.

На главной форме можно поместить дополнительную кнопку для информации об авторе программы, которая может содержать имя автора, его контактный телефон и фото автора (в дополнительной форме).

Естественно, для задания режимов работы программы можно использовать меню, рассмотренное в предыдущих лекциях, а пространство окна главной формы использовать для отображения одного из режимов работы. Однако, в учебных целях, мы рассмотрим организацию «кнопочной» формы как альтернативу меню.

Создадим проект пока с одной главной формой. Для красоты разместим на ней некоторый рисунок или фотографию. Для этого необходимо в окне элементов управления проекта выбрать элемент PictureBox и разместить его в окне формы. В свойствах элемента PictureBox необходимо выбрать Image, в правой части которого открыть диалоговое окно выбора файла изображения. Найти необходимое изображение и подтвердить его выборку. После этого изображение появится в окне главной формы приложения (смотри рисунок 6.1).

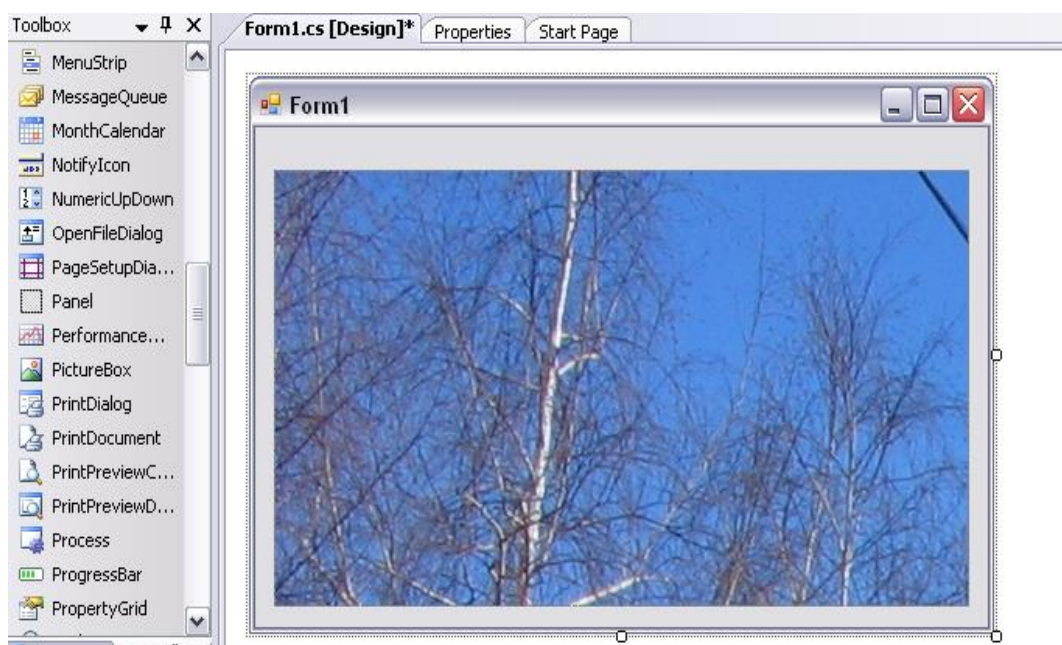


Рисунок 6.1 – Размещение рисунка в окне формы

Разместим на рисунке четыре кнопки – соответствующие режимам работы программы (смотри рисунок 6.2).

Реально в нашей программе три режима работы, но три кнопки в окне формы выглядят как-то неуютно, и я добавил еще одну кнопку – сведения об авторе. Очень часто в меню программы включаются дополнительные режимы работы, например, режим Help, в котором содержится инструкция по работе с программой и отображаться вся информация о нашем приложении.

В учебные программы часто включаются дополнительные теоретические сведения по алгоритму решения задачи или правильной организации режима диалога с программой.

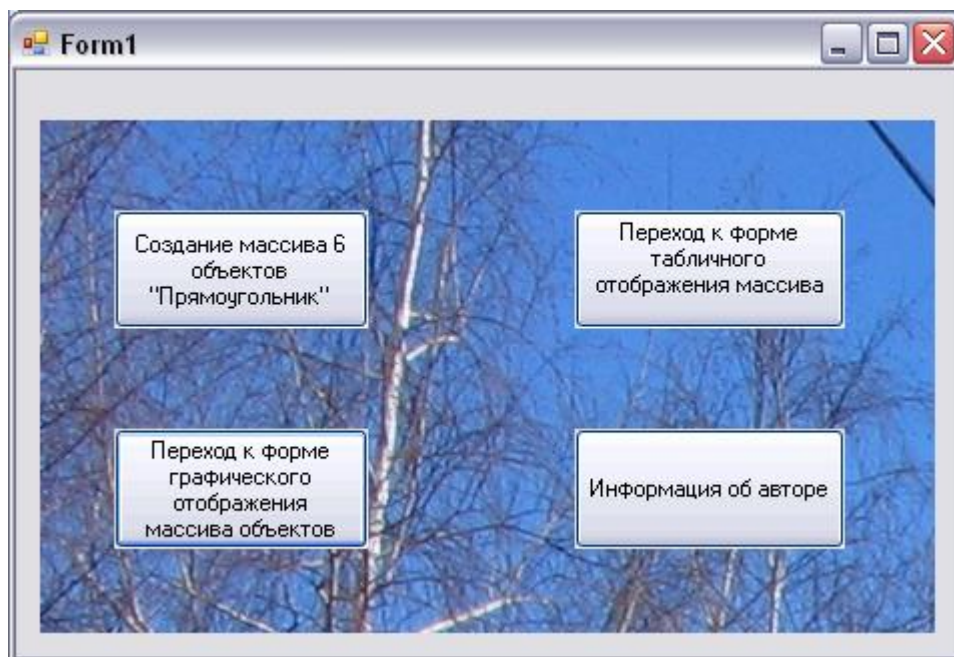


Рисунок 6.2 – Главная «Кнопочная форма» приложения

6.2 Добавление новых форм приложения

Существует несколько вариантов добавления в проект приложения новых форм. Рассмотрим два основных.

Для того чтобы добавить в проект новую форму, щелкните правой клавишей мыши строку названия проекта WindowsFormsApplication1 в окне Solution Explorer (см. рисунок 6.3).

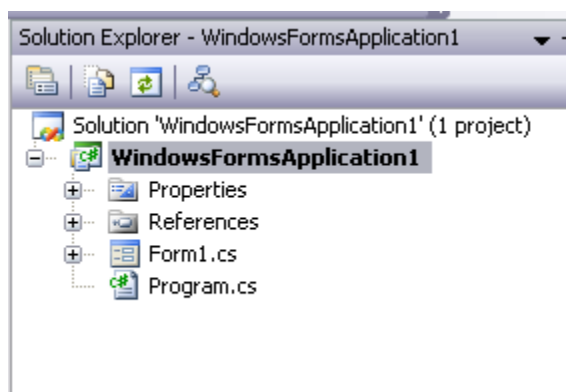


Рисунок 6.3 – Первый вариант добавления формы в проект

В появившемся меню режимов работы выберите режим Add и в нем команду Add Windows Form (см. рисунок 6.4).

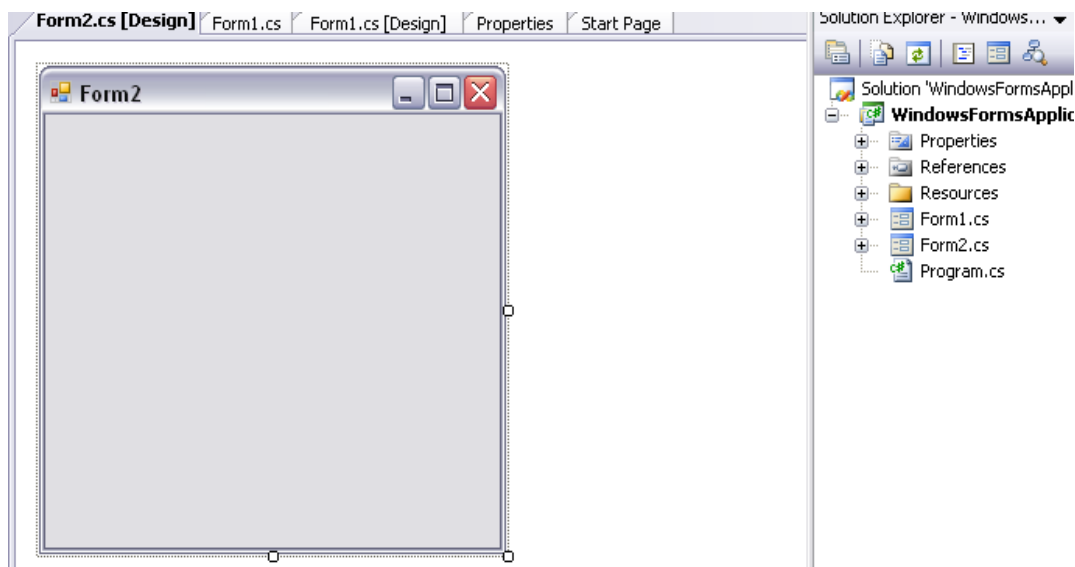


Рисунок 6.4 – Подключение новой формы к проекту приложения.

Обратите внимание запись Form2.cs появилась в окне Solution Explorer проекта WindowsFormsApplication1.

Во втором варианте Вам необходимо выбрать режим Project, а в нем команду Add Windows Form . . . После этого подтвердить название предлагаемой формы нажатием кнопки Add.

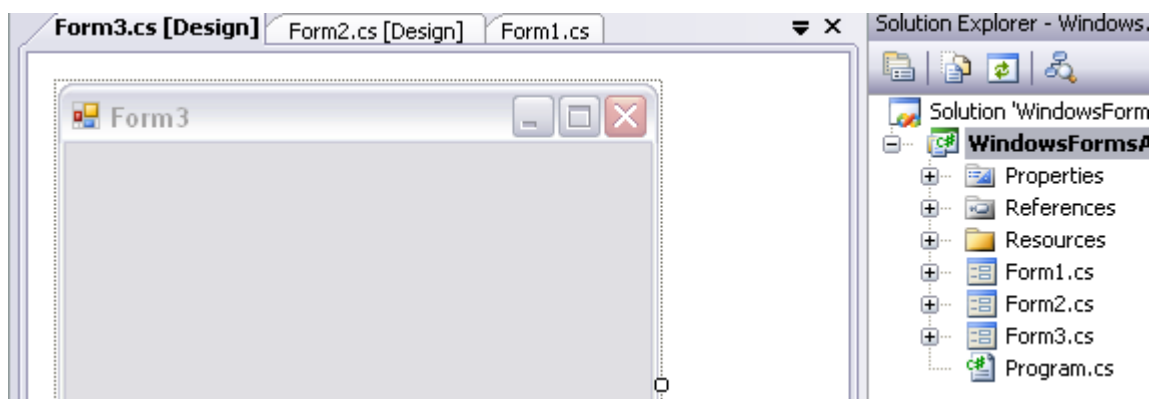


Рисунок 6.5 – Подключение третьей формы к проекту

Контроль подключения форм можно осуществлять как в окне Solution Explorer так и в окне редактирования форм.

6.3 Обработчики событий главной формы

Режим создания массива 6 прямоугольников можно реализовать в главной форме, а отображение результатов работы этого режима в формах 2 и 3.

Исходный код главной формы будет иметь следующий вид:

```
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 button1_Click(object sender, EventArgs e)
        {
            Random rnd = new Random();
            for (int i = 0; i < 6; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    a[i, j] = rnd.Next() % 201 - 100;
                }
            }
            for (int i = 0; i < 6; i++)
            {
                if(Math.Abs(a[i,0]-a[i, 2]) == Math.Abs(a[i, 1] - a[i, 3]))
                    a[i, 4] = 1; else a[i, 4] = 0;
                a[i,5]=Math.Abs(a[i,0]-a[i,2])*Math.Abs(a[i,1]-a[i,3]);
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            int y;
            Form2 f2 = new Form2();
            f2.ShowDialog();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            Form3 f3 = new Form3();
            f3.ShowDialog();
        }
    }
}
```

```

private void button4_Click(object sender, EventArgs e)
{
    Form4 f4 = new Form4();
    f4.Show();
}
}
}

```

При создании массива в 5 столбец будем записывать информацию о прямоугольнике 0 – квадрат, 1 – обычный прямоугольник, а 6 столбец значение площади прямоугольника.

В нашем проекте запланировано несколько форм. Естественно возникает вопрос можно ли открывать одновременно несколько форм и как переходить с одной формы на другую? Ответ на этот вопрос зависит от того как было открыто окно.

Каждое окно (форма) можно открыть как модальное (как "диалоговое окно") или как немодальное (как обычное окно).

Если окно открывается методом Show(), то она задает обычное окно; если окно открывается методом ShowDialog(), то она открывается как диалоговое окно. В чем разница? Из диалогового окна нельзя выйти, не закончив диалог и не закрыв форму. Открыв диалоговое окно, нельзя переключиться на работу с другой формой, не закончив диалог. Закрывать диалоговое окно можно разными способами. Можно щелкнуть по крестику, расположенному в правом верхнем углу формы или специальной кнопкой.

Если форма открыта методом Show, как не диалоговое окно, то, не закончив работу с открытой формой, можно перейти в главную форму или другую немодальную форму, поработать там, нажав какие-нибудь командные кнопки, получив нужную информацию, а затем снова вернуться к исходной форме.

В нашем приложении мы используем оба способа открытия окон форм.

Для закрытия окна можно также применять два метода – Hide() и Close(). Первый из этих методов скрывает форму, второй закрывает. Для диалоговых окон можно применять как метод Hide(), так и метод Close(): эффект будет одинаков – диалоговое окно будет закрыто.

Метод Hide() можно применять и для немодальных форм, открытых методом Show(). Окно, открытое не для диалога, можно временно скрыть, вызвав метод Hide(), а затем показать, вызвав метод Show().

Еще одно замечание, связанное с закрытием форм. Когда закрывается главная форма, закрываются все открытые к этому времени формы и приложение заканчивает свою работу. Когда же закрывается любая другая форма, закрывается только эта форма, остальные формы остаются открытыми.

6.4 Табличная форма представления и редактирования значений

Рассмотрим обработчики событий окна табличной формы представления и редактирования значений координат вершин прямоугольников.

Обработчики реализованы на форме 2, изображенной на рисунке 6.6. Для табличной формы представления значений массива использован элемент DataGridView. Рассмотрим его применение в интерфейсе, позволяющем пользователю вводить и отображать двумерные массивы.

	Точ. А.х	Точ. А.у	Точ. В.х	Точ. В.у	Тип объекта	Площадь
▶	-22	-32	55	-55	Прямоугольник	1771
	-24	60	-56	-31	Прямоугольник	2912
	54	52	41	65	Квадрат	169
	-33	23	13	62	Прямоугольник	1794
	55	11	76	-6	Прямоугольник	357
	-75	100	-21	-95	Прямоугольник	10530
*						

Рисунок 6.6 – Табличная форма представления значений массива

В свойствах элемента DataGridView выбираем Columns и запускаем редактор параметров столбцов таблицы, в котором мы добавляем необходимое количество столбцов и определяем название столбцов в окне формы и имя столбцов в программе (см. рисунок 6.7). С помощью редактора столбцов определяются их ширина в окне формы.

Рисунок 6.7 – Работа с редактором столбцов DataGridView

Исходный код формы 2 имеет следующий вид:

```
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 Form2 : Form
    {
        public static int i, j;
        public static string kop;
        public Form2()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            for (i = 0; i < 6; i++)
            {
                dataGridView1.Rows.Add();
                for (j = 0; j < 6; j++)
                {
                    dataGridView1.Rows[i].Cells[j].Value=Form1.a[i,j].ToString();
                }
                if (Form1.a[i,4]==0) dataGridView1.Rows[i].Cells[4].Value =
"Прямоугольник";
                else dataGridView1.Rows[i].Cells[4].Value = "Квадрат";
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            string elem = "";
            bool ok;
            int k;
            for (i = 0; i < 6; i++)
                for (j = 0; j < 4; j++)
                {
                    do
                    {
                        ok = true;
                        try
                        {
                            elem = dataGridView1.Rows[i].Cells[j].Value.ToString();
                        }
                    }
                }
            }
        }
    }
}
```

```

        Form1.a[i, j] = int.Parse(elem);
    }
    catch (Exception any)
    {
        Form5 f5 = new Form5();
        if (f5.ShowDialog() == DialogResult.OK) k = 0;
        dataGridView1.Rows[i].Cells[j].Value = kop;
        ok = false;
    }
    } while (!ok);
}
for (i = 0; i < 6; i++)
{
    if (Math.Abs(Form1.a[i, 0] - Form1.a[i, 2]) ==
Math.Abs(Form1.a[i, 1] - Form1.a[i, 3])) Form1.a[i, 4] = 1;
        else Form1.a[i, 4] = 0;
        Form1.a[i, 5] = Math.Abs(Form1.a[i, 0] - Form1.a[i, 2]) *
Math.Abs(Form1.a[i, 1] - Form1.a[i, 3]);
    }
}
}
}

```

На форме 2 реализованы 3 обработчика событий – вывод матрицы в таблицу элемента DataGridView, редактирование значений матрицы и возврат к 1 форме. Рассмотрим каждый фрагмент кода отдельно.

Вывод матрицы в таблицу элемента DataGridView содержит цикл перебора строк таблицы, в котором программно добавляется очередная строка и запускается цикл перебора столбцов таблицы.

Поскольку массив сформирован в виде глобальной переменной формы 1, то его использование в форме 2 требует следующей записи `Form1.a[i, j]`.

Значение столбца 4 двумерного массива анализируется на наличие 0 – признак равенства сторон прямоугольника (квадрат).

В последний столбец выводится значение площади прямоугольника.

Процесс редактирования элементов DataGridView включает непосредственное изменение значений на форме 2 и запуск обработчика «Редактирование объекта», в котором содержимое элементов DataGridView переписывается в массив формы 1. Процесс перезаписи находится в охраняемом блоке. Если вы вместо цифр случайно введете символ буквы или другой знак, то запускается обработчик ошибочной ситуации, который создаст и откроет 5 диалоговое окно для повторного ввода ошибочного значения (см. рисунок 6.8).

Исходный код формы 5:

```

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 Form5 : Form
    {
        public Form5()
        {
            InitializeComponent();
            label1.Text = "a[" + Form2.i.ToString() + "," +
Form2.j.ToString() + "]= ?";
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Form2.kop = textBox1.Text;
            Close();
        }
    }
}

```

Исправление ошибок ввода продолжается пока не будет введено правильное числовое значение.

Если в результате редактирования элементов DataGridView изменяется тип прямоугольника, например, получается квадрат, то это учитывается в 4 столбце массива. Одновременно перезаписывается новое значение площади нового прямоугольника.

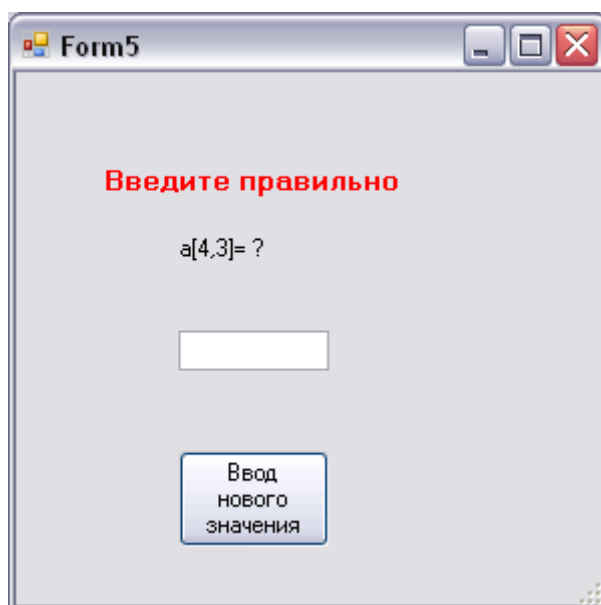


Рисунок 6.8 – Исправление ошибки ввода

В приведенном коде формы 2 программно добавляется очередная строка элемента DataGridView. Можно программно формировать и столбцы элемента DataGridView (обычно это делается при однотипных столбцах). Например, если нам необходимо подготовить элемент DataGridView на 6

столбцов, то можно использовать следующий фрагмент кода для программного добавления столбцов:

```
dataGridView1.Columns.Clear();  
DataGridViewColumn column;  
for (int i = 0; i < 6; i++)  
{  
    column = new DataGridViewTextBoxColumn();  
    column.DataPropertyName = "Column" + i.ToString();  
    column.Name = "Column" + i.ToString();  
    dataGridView1.Columns.Add(column);  
}
```

В первой строке кода мы удаляем предыдущее «состояние столбцов» элемента DataGridView, объявляем переменную `column` а в цикле создаем объект `column`, которому присваиваем имя для формы и имя для программы и добавляем созданный объект нашему элементу.

6.5 Графическая форма представления прямоугольников

Рассмотрим обработчики событий в окне графической формы представления прямоугольников.

Обработчики реализованы на форме 3, изображенной на рисунке 6.9.

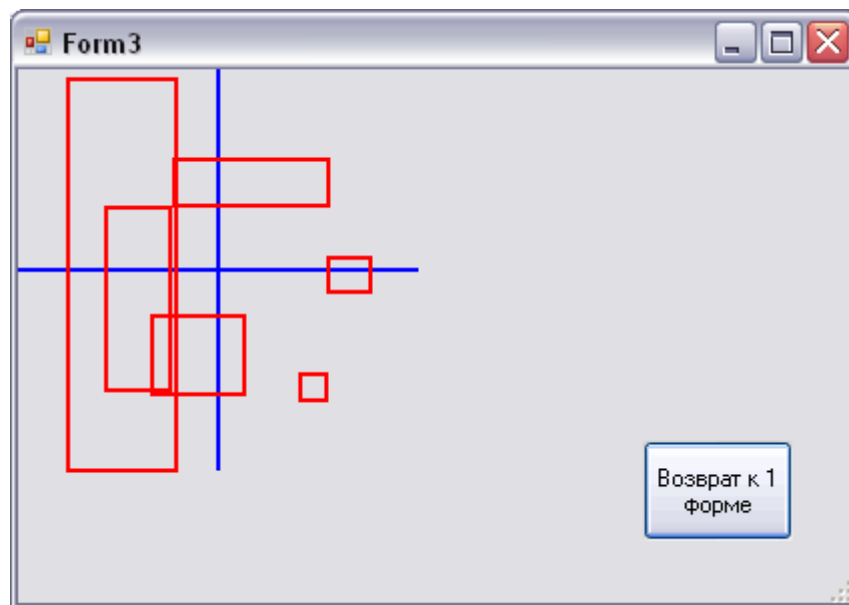


Рисунок 6.9 – Окно графической формы представления прямоугольников

Сразу хочу отметить, что дизайн представленной формы долек от совершенства. В коде формы 3 нет ничего того, что для Вас является новым (эта форма добавлена в проект для большего числа форм). Поэтому просто приведу исходный код формы:

```

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 Form3 : Form
    {
        public Form3()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void Form3_Paint(object sender, PaintEventArgs e)
        {
            int ax, ay, bx, by;
            Pen myPen = new Pen(Color.Blue, 2);
            Graphics g = e.Graphics;
            g.DrawLine(myPen, 0, 100, 200, 100);
            g.DrawLine(myPen, 100, 0, 100, 200);
            myPen = new Pen(Color.Red, 2);
            for (int i = 0; i < 6; i++)
            {
                if (Form1.a[i, 0] < Form1.a[i, 2]) ax = Form1.a[i, 0];
                else ax = Form1.a[i, 2];
                if (Form1.a[i, 1] < Form1.a[i, 3]) ay = Form1.a[i, 1];
                else ay = Form1.a[i, 3];
                bx = Math.Abs(Form1.a[i, 0] - Form1.a[i, 2]);
                by = Math.Abs(Form1.a[i, 1] - Form1.a[i, 3]);
                g.DrawRectangle(myPen, ax+100, ay+100, bx, by);
            }
        }
    }
}

```

Еще одна форма проекта «Информация об авторе» также добавлена для демонстрации возможностей много документного приложения.



Рисунок 6.10 – Окно режима программы «Информация об авторе»

Исходный код формы 4 содержит только обработчик кнопки возврата к 1 форме. Остальное реализовано с помощью свойств элементов формы 4:

```
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 Form4 : Form
    {
        public Form4()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
}
```

