Н. Культин

bhv

Microsoft®

Visual C#

в задачах и примерах

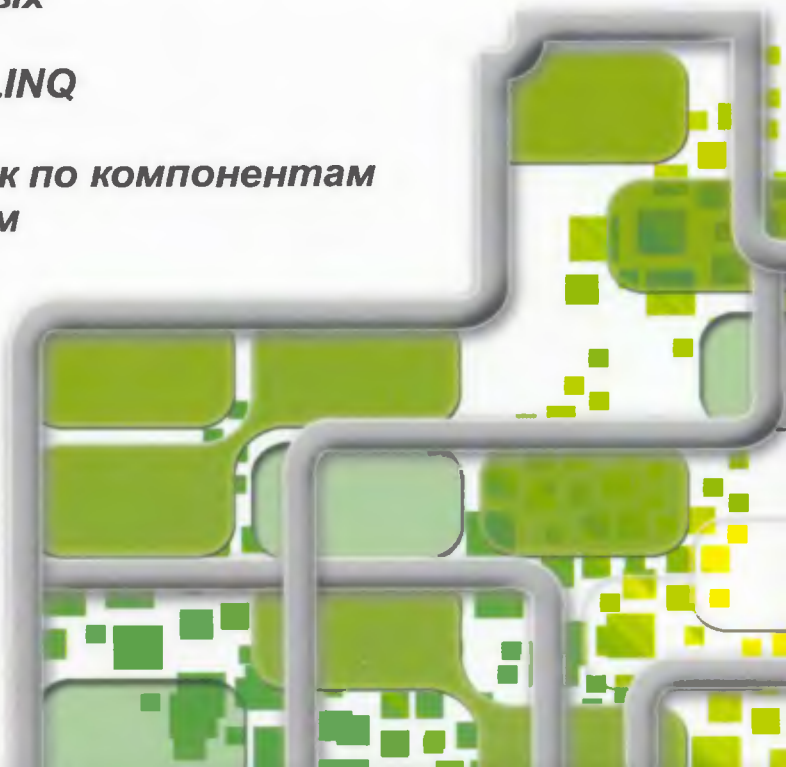
В ногу со временем!

Базовые компоненты

Программирование графики и баз данных

Работа с LINQ

Справочник по компонентам и функциям



+ CD

Никита Культин

Microsoft© Visual C#

в задачах и примерах

Санкт-Петербург
«БХВ-Петербург»
2009

УДК 681.3.068
ББК 32.973.26-018.1
К90

Культин Н. Б.

К90 Microsoft Visual C# в задачах и примерах. — СПб.: БХВ-Петербург, 2009. — 320 с.: ил. + CD-ROM

ISBN 978-5-9775-0410-2

Книга представляет собой сборник программ и задач для самостоятельного решения. Примеры различной степени сложности – от простейших до приложений работы с графикой и базами данных, в том числе Microsoft Access и Microsoft SQL Server Compact Edition – демонстрируют назначение базовых компонентов, раскрывают тонкости разработки приложений Windows Forms в Microsoft Visual C#. Уделено внимание использованию технологии LINQ. Справочник, входящий в книгу, содержит описание базовых компонентов, событий, исключений и наиболее часто используемых функций.

На прилагаемом компакт-диске находятся проекты, представленные в книге.

Для начинающих программистов

УДК 681.3.068
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн серии	*Игоря Цырульникова*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.03.09.
Формат 60×90 1/16. Печать офсетная. Усл. печ. л. 20.
Тираж 2500 экз. Заказ № 978
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.003650.04.08 от 14.04.2008 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0410-2

Оглавление

Предисловие

В последнее время в общем объеме вновь создаваемого программного обеспечения различного назначения увеличивается доля .NET-приложений — программ, ориентированных на платформу Microsoft .NET. Это объясняется, прежде всего, новыми возможностями, которые предоставляет платформа прикладным программам, а также тем, что технология .NET поддерживается новейшими операционными системами.

Microsoft .NET — это технология, в основе которой лежит идея универсального программного кода, который может быть выполнен любым компьютером, вне зависимости от используемой операционной системы. Универсальность программного кода обеспечивается за счет предварительной (выполняемой на этапе разработки) компиляции исходной программы в *универсальный промежуточный код* (CIL-код, Common Intermediate Language), который во время загрузки транслируется в *выполняемый*. Преобразование промежуточного кода в выполняемый осуществляет JIT-компилятор (от Jast In Time — в тот же момент, "на лету"), являющийся элементом виртуальной выполняющей системы (Virtual Execution System, VES). Работу .NET-приложений в операционной системе Windows обеспечивает Microsoft .NET Framework.

Чтобы понять, что такое .NET, и какие возможности она предоставляет программисту, необходимо опробовать ее в деле. Для этого нужно изучить среду и технологию разработки, назначение и возможности компонентов, их свойства и методы. И здесь хорошим подспорьем могут стать примеры, программы, разработанные другими программистами.

Среда разработки **Microsoft Visual C#** является инструментом разработки .NET-приложений для Windows. В ней интегрированы удобный дизайнер форм, специализированный редактор кода,

отладчик, мастер публикации и другие инструменты, необходимые программисту.

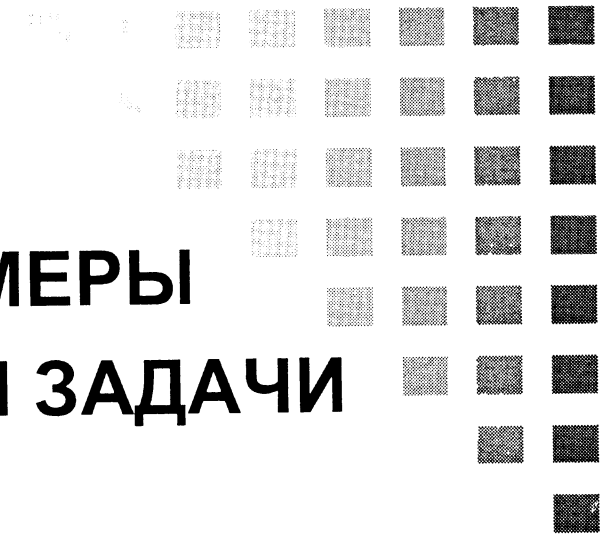
Книга, которую вы держите в руках, посвящена практике программирования в Microsoft Visual C#. В ней собраны разнообразные примеры, демонстрирующие назначение базовых компонентов, технологии работы с файлами, графикой, и базами данных. Следует обратить внимание, что большинство примеров не являются учебными в общепринятом смысле, это — вполне работоспособные, законченные программы.

Книга состоит из двух частей. Первая часть содержит примеры программ. Примеры представлены в виде краткого описания, диалоговых окон и хорошо документированных листингов.

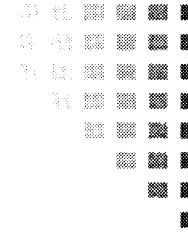
Вторая часть книги — это краткий справочник. В нем можно найти описание базовых компонентов и наиболее часто используемых функций.

Научиться программировать можно, только программируя, решая конкретные задачи. Поэтому, чтобы получить максимальную пользу от книги, вы должны работать с ней активно. Изучайте листинги, старайтесь понять, как работают программы. Не бойтесь экспериментировать — совершенствуйте программы, вносите в них изменения. Чем больше вы сделаете самостоятельно, тем большему научитесь!

Часть 1



Примеры и задачи



Часть 1

Примеры и задачи

Базовые компоненты

В этом разделе приведены примеры, демонстрирующие назначение и технологию работы с базовыми компонентами.

Общие замечания

- Процесс создания программы состоит из двух шагов: сначала создается форма, затем — функции обработки *событий*.
- Форма создается путем добавления необходимых компонентов и последующей их настройки.
- В форме практически любого приложения есть компоненты, обеспечивающие взаимодействие программы с пользователем. Такие компоненты называют базовыми.
- К базовым компонентам можно отнести:
  - `Label` — поле отображения текста;
  - `TextBox` — поле редактирования текста;
  - `Button` — командную кнопку;
  - `CheckBox` — флажок;

- `RadioButton` — селектор (радиокнопка);
- `ListBox` — список выбора;
- `ComboBox` — поле редактирования со списком выбора.

- Вид и поведение компонента определяют значения его *свойств* (характеристик) (описание свойств базовых компонентов можно найти в справочнике во второй части книги).
- Основную работу в программе выполняют функции обработки *событий* (описание основных событий можно найти в справочнике во второй части книги).
- Исходную информацию программа может получить из полей редактирования (компонент `TextBox`), списка (компонент `ListBox`), комбинированного списка (компонент `ComboBox`).
- Ввести значения логического типа можно с помощью компонентов `CheckBox` и `RadioButton`.
- Результат программа может вывести в поле отображения текста (компонент `Label`), в поле редактирования или в окно сообщения (метод `MessageBox.Show( )`).
- Для преобразования текста, находящегося, например, в поле редактирования, в целое число нужно использовать функцию `Convert.ToInt32()`, в дробное число — `Convert.ToDouble()`.
- Преобразовать численное значение в строку позволяет метод `ToString()`. В качестве параметра метода можно указать формат отображения: `"C"` — денежный с разделителями групп разрядов и обозначением валюты (`currency`); `"N"` — числовой с разделителями групп разрядов (`numeric`); `"F"` — числовой без разделителей групп разрядов (`fixed`).

Мили-километры

Программа **Мили-километры** (рис. 1.1, листинг 1.1) пересчитывает расстояние из миль в километры. Демонстрирует использование компонента `TextBox` для ввода данных и компонента `Label`

для отображения числовой информации. Программа спроектирована таким образом, что в поле редактирования можно ввести только дробное число. Значения свойств формы приведены в табл. 1.1.

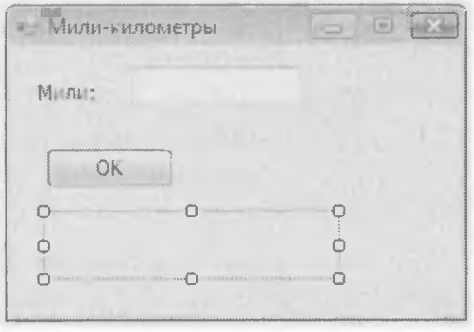


Рис. 1.1. Форма программы

Таблица 1.1. *Значения свойств формы*

Свойство	Значение	Комментарий
`Text`	Мили-километры	Текст заголовка
`StartPosition`	`CenterScreen`	Начальное положение окна — в центре экрана
`FormBorderStyle`	`FixedSingle`	Тонкая граница окна. Пользователь не сможет изменить размер окна путем перемещения его границы
`MaximizeBox`	`False`	Кнопка **Развернуть окно** недоступна. Пользователь не сможет развернуть окно программы на весь экран
`Font`	`Tahoma; 9pt`	Шрифт, наследуемый компонентами формы

Листинг 1.1. Модуль формы программы *Мили-километры*

```
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 m2k
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // нажатие клавиши в поле редактирования
        private void textBox1_KeyPress(object sender,
                                KeyPressEventArgs e)
        {
            // Правильными символами считаются цифры,
            // запятая, <Enter> и <Backspace>.
            // Будем считать правильным символом
            // также точку, но заменим ее запятой.
            // Остальные символы запрещены.
            // Чтобы запрещенный символ не отображался
            // в поле редактирования, присвоим
            // значение true свойству Handled параметра e

            if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
            {
```

```
        // цифра
        return;
    }

    if (e.KeyChar == '.')
    {
        // точку заменим запятой
        e.KeyChar = ',';
    }

    if (e.KeyChar == ',')
    {
        if (textBox1.Text.IndexOf(',') != -1)
        {
            // запятая уже есть в поле редактирования
            e.Handled = true;
        }
        return;
    }

    if (  Char.IsControl (e.KeyChar) )
    {
        // <Enter>, <Backspace>, <Esc>
        if ( e.KeyChar == (char) Keys.Enter)
            // нажата клавиша <Enter>
            // установить курсор на кнопку OK
            button1.Focus();
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}
```

```
        // щелчок на кнопке OK
        private void button1_Click(object sender, EventArgs e)
        {
            double mile; // расстояние в милях
            double km;   // расстояние в километрах

            // Если в поле редактирования нет данных,
            // то при попытке преобразовать пустую
            // строку в число возникает исключение.
            try
            {
                mile = Convert.ToDouble(textBox1.Text);

                km = mile * 1.609344;

                label2.Text = km.ToString("n")
                              + " км.";
            }
            catch
            {
                // обработка исключения:
                // переместить курсор в поле редактирования
                textBox1.Focus();
            }
        }
    }
}
```

Фунты-килограммы

Программа **Фунты-килограммы** (рис. 1.2, листинг 1.2) пересчитывает вес из фунтов в килограммы. Показывает, как можно управлять доступностью командной кнопки (компонент `Button`) в зависимости от наличия данных в поле редактирования.

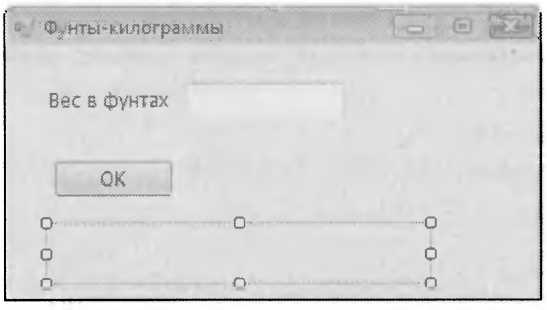


Рис. 1.2. Форма программы **Фунты-килограммы**

Листинг 1.2. Модуль формы программы *Фунты-килограммы*

```
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 Form1()
        {
            InitializeComponent();

            // сделать кнопку OK недоступной
            button1.Enabled = false;
```

```
// нажатие клавиши в поле редактирования
private void textBox1_KeyPress(object sender,
                        KeyPressEventArgs e)
{
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        return;

    if (e.KeyChar == '.') e.KeyChar = ',';

    if (e.KeyChar == ',')
    {
        // в поле редактирования не может
        // быть больше одной запятой и запятая
        // не может быть первым символом
        if ( (textBox1.Text.IndexOf(',') != -1) ||
              ( textBox1.Text.Length == 0))
        {
            e.Handled = true;
        }
        return;
    }

    if (  Char.IsControl (e.KeyChar) )
    {
        // <Enter>, <Backspace>, <Esc>
        if ( e.KeyChar == (char) Keys.Enter)
            // установить курсор на кнопку OK
            button1.Focus();
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}
```

```
        // текст в поле редактирования изменился
        private void textBox1_TextChanged(object sender,
                                          EventArgs e)
        {
            label2.Text = ""; // очистить поле отображения
                              // результата расчета

            if (textBox1.Text.Length == 0)
                // в поле редактирования нет данных
                // сделать кнопку OK недоступной
                button1.Enabled = false;
            else
                // сделать кнопку OK доступной
                button1.Enabled = true;
        }

        // щелчок на кнопке OK
        private void button1_Click(object sender, EventArgs e)
        {
            double funt; // вес в фунтах
            double kg;   // вес в килограммах

            funt = Convert.ToDouble(textBox1.Text);

            // 1 фунт = 409,5 грамма
            kg = funt * 0.4095;

            label2.Text = funt.ToString("N") + " ф. = "
                          + kg.ToString("N") + " кг.";
        }
    }
}
```

Конвертор

Программа **Конвертор** (рис. 1.3, листинг 1.3) демонстрирует обработку одной функцией событий от нескольких компонентов.

Рис. 1.3. Форма программы **Конвертор**

Листинг 1.3. Модуль формы программы *Конвертор*

```
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 Form1()
        {
```

```
        InitializeComponent();
    }

    // Обрабатывает нажатие клавиши в полях
    // редактирования Курс и Цена.
    // Сначала надо обычным образом создать функцию
    // обработки события KeyPress для компонента
    // textBox1, затем - указать ее в качестве
    // обработчика этого же события для компонента
    // textBox2
    private void textBox1_KeyPress(object sender,
                                   KeyPressEventArgs e)
    {
        if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
            return;

        if (e.KeyChar == '.') e.KeyChar = ',';

        if (e.KeyChar == ',')
        {
            if ((textBox1.Text.IndexOf(',') != -1) ||
                  (textBox1.Text.Length == 0))
            {
                e.Handled = true;
            }
            return;
        }

        if (Char.IsControl(e.KeyChar))
        {
            if (e.KeyChar == (char)Keys.Enter)
            {
                if (sender.Equals(textBox1))
                    // клавиша <Enter> нажата в поле Курс
                    // переместить курсор в поле Цена
```

```
                textBox2.Focus();
            else
                // клавиша <Enter> нажата в поле Цена
                //
                button1.Focus();
        }
        return;
    }

    // остальные символы запрещены
    e.Handled = true;
}

// изменился текст в поле редактирования
// textBox1 или textBox2
private void textBox1_TextChanged(object sender,
                                  EventArgs e)
{
    label3.Text = "";
    if ((textBox1.Text.Length == 0) ||
        (textBox2.Text.Length == 0))
        // если какое-либо из полей не содержит
        // данных, то сделать недоступной кнопку OK
        button1.Enabled = false;
    else
        button1.Enabled = true;
}

// щелчок на кнопке OK
private void button1_Click(object sender,
EventArgs e)
{
    double usd; // цена в долларах
    double k;   // курс
    double rub; // цена в рублях
```

```
            usd = Convert.ToDouble(textBox1.Text);
            k = Convert.ToDouble(textBox2.Text);

            rub = usd * k;

            label3.Text =
                  rub.ToString("C"); // финансовый формат
        }
    }
}
```

Фото

Программа **Фото** (рис. 1.4, листинг 1.4) позволяет рассчитать стоимость печати фотографий. Демонстрирует использование компонента `RadioButton`.

Рис. 1.4. Форма программы **Фото**

Листинг 1.4. Модуль формы программы *Фото*

```
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 Form1()
        {
            InitializeComponent();

            // настройка компонентов
            radioButton1.Checked = true;
            button1.Enabled = false;
        }

        // щелчок на кнопке OK
        private void button1_Click(object sender, EventArgs e)
        {
            double cena = 0 ; // цена
            int n;            // кол-во фотографий
            double sum;       // сумма

            if (radioButton1.Checked)
                cena = 8.50;
            if (radioButton2.Checked)
                cena = 10;
            if (radioButton3.Checked)
                cena = 15.5;
```

```
    n = Convert.ToInt32(textBox1.Text);
    sum = n * cena;

    label2.Text = "Цена: " + cena.ToString("c") +
        "\nКоличество: " + n.ToString() + "шт.\n" +
        "Сумма заказа: " + sum.ToString("C");
}

// В поле Количество можно ввести только целое число
private void textBox1_KeyPress(object sender,
                               KeyPressEventArgs e)
{
    if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
        return;

    if (Char.IsControl(e.KeyChar))
    {
        if (e.KeyChar == (char)Keys.Enter)
        {
            // нажата клавиша <Enter>
            button1.Focus();
        }
        return;
    }
    // остальные символы запрещены
    e.Handled = true;
}

private void textBox1_TextChanged(object sender,
                                  EventArgs e)
{
    if (textBox1.Text.Length == 0)
        button1.Enabled = false;
    else
        button1.Enabled = true;
```

```
            label2.Text = "";
        }

        // щелчок на radioButton
        private void radioButton1_Click(object sender,
                                        EventArgs e)
        {
            label2.Text = "";
            // установить курсор в поле Количество
            textBox1.Focus();
        }
    }
}
```

Комплектация автомобиля

Программа **Комплектация** (рис. 1.5, листинг 1.5) позволяет рассчитать стоимость автомобиля в зависимости от выбранной комплектации. Демонстрирует использование компонента `CheckBox`. Отображение картинки обеспечивает компонент `PictureBox`.

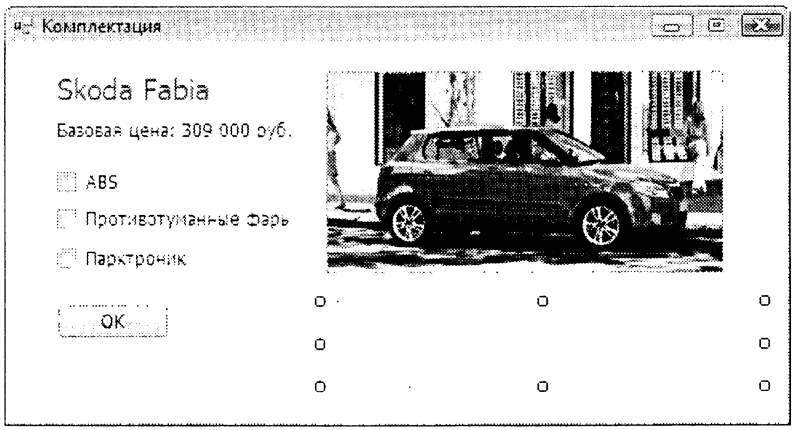


Рис. 1.5. Форма программы **Комплектация**

Листинг 1.5. Модуль формы программы *Комплектация*

```
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 Form1()
        {
            InitializeComponent();
        }

        // щелчок на кнопке OK
        private void button1_Click(object sender, EventArgs e)
        {
            double sum;       // сумма
            double discount; // скидка
            double total;     // общая сумма

            sum = 309000;
            discount = 0;

            if (checkBox1.Checked)
            {
                // ABS
                sum += 8390;
            }
```

```
        if (checkBox2.Checked)
        {
            // противотуманные фары
            sum += 5990;
        }

        if (checkBox3.Checked)
        {
            // парктроник
            sum += 7590;
        }

        total = sum;

        string st;
        st = "Цена в выбранной комплектации: " +
              sum.ToString("C");

        if ((checkBox1.Checked) && (checkBox2.Checked) &&
             (checkBox3.Checked))
        {
            // скидка предоставляется, если
            // выбраны все опции
            discount = sum * 0.01;
            total = total - discount;
            st += "\nСкидка (1%): " +
                   discount.ToString("C") +
                  "\nИтого: " + total.ToString("C");
        }
        label3.Text = st;
    }

    // пользователь изменил состояние переключателя
    // функция обрабатывает событие CheckedChanged
    // компонентов checkBox1 - checkBox3
```

```
        private void checkBox1_CheckedChanged(object sender,
                                              EventArgs e)
        {
            label3.Text = "";
        }
    }
}
```

Жалюзи

Программа **Жалюзи** (рис. 1.6, листинг 1.6) демонстрирует использование компонента `ComboBox`, который служит для выбора материала (пластик, алюминий, соломка, текстиль).

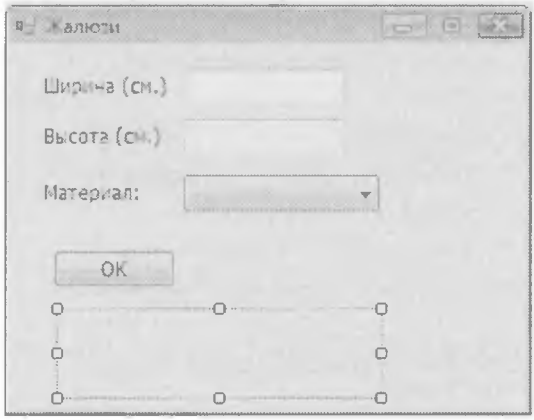


Рис. 1.6. Форма программы **Жалюзи**

Листинг 1.6. Модуль формы программы *Жалюзи*

```
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 Form1()
        {
            InitializeComponent();

            // настройка компонентов
            comboBox1.DropDownStyle =
                ComboBoxStyle.DropDownList;

            // comboBox1.Items.Add("пластик");
            // comboBox1.Items.Add("алюминий");
            // comboBox1.Items.Add("бамбук");
            // comboBox1.Items.Add("соломка");
            // comboBox1.Items.Add("текстиль");

            comboBox1.SelectedIndex = 0;
        }

        // нажатие клавиши в поле редактирования
        // функция обрабатывает событие KeyPress
        // компонентов textBox1 и textBox2
        private void textBox1_KeyPress(object sender,
                                       KeyPressEventArgs e)
        {
```

```
    {
        if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
            return;
        if (Char.IsControl(e.KeyChar))
        {
            if (e.KeyChar == (char)Keys.Enter)
            {
              if (sender.Equals(textBox1))
               // клавиша <Enter> нажата в поле Ширина
               // переместить курсор в поле Высота
                    textBox2.Focus();
              else
               // клавиша <Enter> нажата в поле Высота
               // переместить фокус на comboBox1
               comboBox1.Focus();
            }
            return;
        }

        // остальные символы запрещены
        e.Handled = true;
    }
}

private void textBox1_TextChanged(object sender,
                                  EventArgs e)
{
    if ((textBox1.Text.Length == 0) ||
        (textBox2.Text.Length == 0))

        button1.Enabled = false;
    else
        button1.Enabled = true;
```

```
        label4.Text = "";
    }

    // щелчок на кнопке OK
    private void button1_Click(object sender, EventArgs e)
    {
        double w;
        double h;
        double cena = 0; // цена за 1 кв.м.
        double sum;

        w = Convert.ToDouble(textBox1.Text);
        h = Convert.ToDouble(textBox2.Text);

        switch (comboBox1.SelectedIndex)
        {
            case 0: cena = 50;  break; // пластик
            case 1: cena = 100; break; // алюминий
            case 2: cena = 75;  break; // бамбук
            case 3: cena = 70;  break; // соломка
            case 4: cena = 60;  break; // текстиль
        }

        sum = (w * h) / 10000 * cena;
        label4.Text =
            "Размер: " + w + "x" + h + " см.\n" +
            "Цена (р./м.кв.): " + cena.ToString("c") +
            "\nСумма: " + sum.ToString("c");
    }

    // в списке Материал пользователь
    // выбрал другой элемент
    private void comboBox1_SelectedIndexChanged
                            (object sender, EventArgs e)
```

```
        {
            label4.Text = "";
        }
    }
}
```

Калькулятор

Программа **Калькулятор** (листинг 1.7) демонстрирует создание компонентов в "коде". Кнопки калькулятора — объединенные в массив `btn` компоненты `Button` — создает и настраивает конструктор формы. Он же назначает кнопкам функции обработки события `Click`. Форма и окно программы приведены на рис. 1.7.

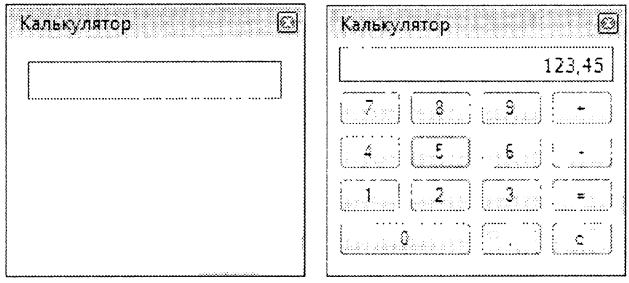


Рис. 1.7. Форма и окно программы **Калькулятор**

Листинг 1.7. Модуль формы программы *Калькулятор*

```
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
    {
        private const int bw = 40, bh = 22; // размер кнопки
        // расстояние между кнопками
        private const int dx = 5, dy = 5;

        // кнопки
        private Button[] btn = new Button[15];

        // текст на кнопках
        char[] btnText = {'7','8','9','+',
                          '4','5','6','-',
                          '1','2','3','=',
                          '0',',','c'};

        // кнопку будем идентифицировать
        // по значению свойства Tag
        int[] btnTag = {7,8,9,-3,
                        4,5,6,-4,
                        1,2,3,-2,
                        0,-1,-5};

        private double ac = 0; // аккумулятор
        private int op = 0 ;   // код операции

        private Boolean fd = true;
        // fd == true - ждем первую цифру числа,
        // например, после нажатия кнопки "+"
        // fd == false - ждем следующую цифру
```

```
public Form1()
{
    InitializeComponent();

    // установить размер клиентской области формы
    this.ClientSize =
        new Size(4*bw + 5*dx, 5*bh + 7*dy);

    // задать размер и положение индикатора
    label1.SetBounds(dx, dy, 4*bw + 3*dx, bh);
    label1.Text = "0";

    y = label1.Bottom + dy ;

    // создать кнопки
    int k =0;     // номер кнопки
    int x, y;     // координаты кнопки

    for ( int i=0; i < 4; i++)
    {
        x = dx;
        for (int j = 0; j < 4; j++)
        {
            if ( !((i == 3 ) && (j == 0)))
            {
                // создать и настроить кнопку
                btn[k] = new Button();
                btn[k].SetBounds(x, y, bw, bh);
                btn[k].Tag = btnTag[k];
                btn[k].Text = btnText[k].ToString();

                // задать функцию обработки
                // события Click
                this.btn[k].Click +=
          new System.EventHandler(this.Button_Click);
```

```
            if (btnTag[k] < 0)
            {
                // кнопка операции
                btn[k].BackColor =
                    SystemColors.ControlLight;
            }

            // поместить кнопку на форму
            this.Controls.Add(this.btn[k]);

            x = x + bw + dx;
            k++;
        }
        else // кнопка ноль
        {
          // создать и настроить кнопку
          btn[k] = new Button();
          btn[k].SetBounds(x, y, bw * 2 + dx, bh);
          btn[k].Tag = btnTag[k];
          btn[k].Text = btnText[k].ToString();

          // задать функцию обработки
          // события Click
          this.btn[k].Click +=
        new
        System.EventHandler(this.Button_Click);

          // поместить кнопку на форму
          this.Controls.Add(this.btn[k]);

          x = x + 2*bw + 2*dx;
          k++;
          j++;
        }
    }
```

```
            y = y + bh + dy;
        }
    }

    // щелчок на кнопке
    private void Button_Click(object sender,
                              System.EventArgs e)
    {
        // нажатая кнопка
        Button btn = (Button)sender;

        // кнопки 1..9
        if ( Convert.ToInt32(btn.Tag) > 0)
        {
            if ( fd )
            {
                // на индикаторе ноль, т.е.
                // это первая цифра
                label1.Text = btn.Text;
                fd = false;
            }
            else
                label1.Text += btn.Text;
            return;
        }

        // ноль
        if (Convert.ToInt32(btn.Tag) == 0)
        {
             if (fd) label1.Text = btn.Text;
             if (label1.Text != "0")
                    label1.Text += btn.Text;
             return;
        }
```

```
// запятая
if ( Convert.ToInt32(btn.Tag) == -1 )
{
    if (fd)
    {
        label1.Text = "0,";
        fd = false;
    }
    else
        if (label1.Text.IndexOf(",") == -1)
            label1.Text += btn.Text;
    return;
}

// "C" - очистить
if (Convert.ToInt32(btn.Tag) == -5 )
{
    ac = 0; // очистить аккумулятор
    op = 0;
    label1.Text = "0";

    fd = true; // снова ждем первую цифру
    return;
}

// кнопки "+","-","="
if (Convert.ToInt32(btn.Tag) < -1)
{
    double n; // число на индикаторе

    n = Convert.ToDouble(label1.Text);

    // Нажатие клавиши операции означает, что
    // пользователь ввел операнд. Если в
```

```
                // аккумуляторе есть число, то выполним
                // операцию. Если это не так, запомним
                // операцию, чтобы выполнить ее при следующем
                // нажатии клавиши операции.
                if (ac != 0)
                {
                    switch (op)
                    {
                        case -3: ac += n;
                            break;
                        case -4: ac -= n;
                            break;
                        case -2: ac = n;
                            break;
                    }
                    label1.Text = ac.ToString("N");
                }
                else {
                    ac = n;
                }

                op = Convert.ToInt32(btn.Tag);
                fd = true; // ждем следующее число
            }
        }
    }
}
```

Просмотр иллюстраций

Программа **Просмотр иллюстраций** (рис. 1.8, листинг 1.8) демонстрирует использование компонентов `ListBox`, `PictureBox` и `FolderBrowserDialog`. Выбор папки, в которой находятся иллюстрации, выполняется в стандартном окне **Обзор папок** (`FolderBrowserDialog`).

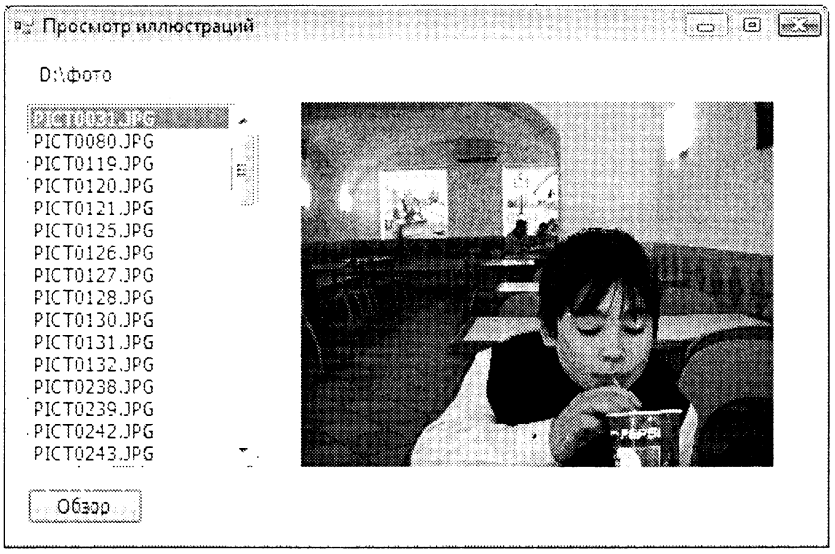


Рис. 1.8. Окно программы **Просмотр иллюстраций**

Листинг 1.8. Модуль формы программы *Просмотр иллюстраций*

```
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;

using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
```

```
int pbw, pbh,      // первоначальный размер
    pbX, pbY;      // и положение pictureBox

string aPath;      // путь к файлам картинок

public Form1()
{
    InitializeComponent();

    // запомнить размер
    // и положение pictureBox1
    pbh = pictureBox1.Height;
    pbw = pictureBox1.Width;
    pbX = pictureBox1.Location.X;
    pbY = pictureBox1.Location.Y;

    // элементы listBox1 сортируются в
    // алфавитном порядке
    listBox1.Sorted = true;

    DirectoryInfo di; // каталог
    // получить имя каталога "Мои рисунки"
    di = new DirectoryInfo(Environment.GetFolderPath(

    Environment.SpecialFolder.MyPictures));
    aPath = di.FullName;
    label1.Text = aPath;

    // сформировать список иллюстраций
    FillListBox(aPath);
}

// формирует список иллюстраций
// aPath - путь к файлам иллюстраций
private Boolean FillListBox(string aPath)
```

```
{
    // информация о каталоге
    DirectoryInfo di =
        new DirectoryInfo(aPath);

    // информация о файлах
    FileInfo[] fi = di.GetFiles("*.jpg");

    // очистить список listBox
    listBox1.Items.Clear();

    // добавляем в listBox1 имена jpg-файлов,
    // содержащихся в каталоге aPath
    foreach (FileInfo fc in fi)
    {
        listBox1.Items.Add(fc.Name);
    }

    label1.Text = aPath;

    if (fi.Length == 0) return false;
    else
    {
        // выбираем первый файл из полученного списка
        listBox1.SelectedIndex = 0;
        return true;
    }
}

// пользователь выбрал другой
// элемент списка щелчком кнопкой
// мыши или перемещением по списку
// при помощи клавиатуры
```

```
private void listBox1_SelectedIndexChanged
                    (object sender, EventArgs e)
{
    double mh, mw;    // коэффициенты масштабирования

    pictureBox1.Visible = false;
    pictureBox1.Left = pbX;

    // загружаем изображение в pictureBox1
    pictureBox1.SizeMode =
                PictureBoxSizeMode.AutoSize;
    pictureBox1.Image =
        new Bitmap( aPath + "\\" +
                    listBox1.SelectedItem.ToString());

    // масштабируем, если нужно
    if ((pictureBox1.Image.Width > pbw) ||
        (pictureBox1.Image.Height > pbh))
    {
        pictureBox1.SizeMode =
            PictureBoxSizeMode.StretchImage;

        mh = (double)pbh /
                (double)pictureBox1.Image.Height;
        mw = (double)pbw /
                (double)pictureBox1.Image.Width;

        if (mh < mw)
        {
            // масштабируем по ширине
            pictureBox1.Width = Convert.ToInt16(
                    pictureBox1.Image.Width * mh);
            pictureBox1.Height = pbh;
        }
```

```
            else
            {
                // масштабируем по высоте
                pictureBox1.Width = pbw;
                pictureBox1.Height = Convert.ToInt16(
                        pictureBox1.Image.Height * mw);
            }
        }

        // разместить картинку в центре области
        // отображения иллюстраций
        pictureBox1.Left =
                pbX + (pbw - pictureBox1.Width) / 2;
        pictureBox1.Top =
                pbY + (pbh - pictureBox1.Height) / 2;

        pictureBox1.Visible = true;
    }

    // щелчок на кнопке Обзор
    private void button1_Click(object sender, EventArgs e)
    {
        // FolderBrowserDialog - окно Обзор папок
        FolderBrowserDialog fb
                        = new FolderBrowserDialog();

        fb.Description =
             "Выберите папку,\n" +
             "в которой находятся иллюстрации";
        fb.ShowNewFolderButton = false;

        // отображаем диалоговое окно
        if (fb.ShowDialog() == DialogResult.OK)
        {
```

```
                // пользователь выбрал каталог и
                // щелкнул на кнопке OK
                aPath = fb.SelectedPath;
                label1.Text = aPath;

                if ( !FillListBox(fb.SelectedPath))
                    // в каталоге нет файлов иллюстраций
                    pictureBox1.Image = null;
            }
        }
    }
}
```

Просмотр иллюстраций-2

Программа **Просмотр иллюстраций-2** (рис. 1.9, листинг 1.9) позволяет просмотреть jpg-файлы.

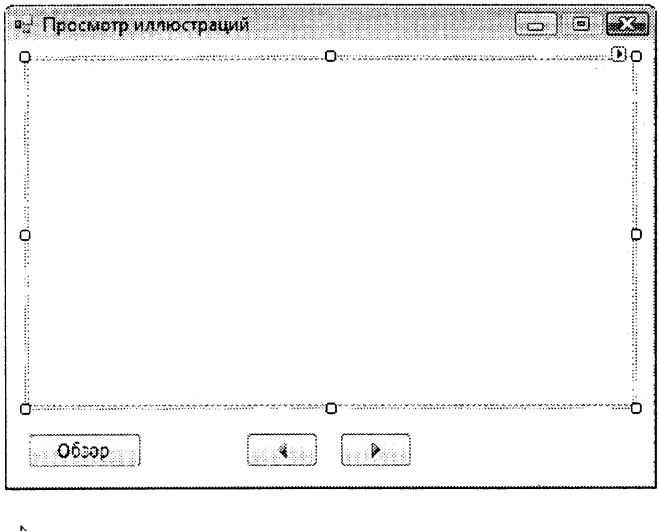


Рис. 1.9. Форма программы **Просмотр иллюстраций**

Она демонстрирует использование компонента PictureBox, показывает, как превратить командную кнопку в графическую, а также как при помощи компонента ToolTip обеспечить отображение подсказки при позиционировании указателя мыши на командной кнопке. Чтобы на кнопке появилась картинка, нужно присвоить значение свойству Image — указать png-файл с "прозрачным" фоном. Значения свойств, обеспечивающих отображение подсказок, приведены в табл. 1.2.

Таблица 1.2. *Свойства, обеспечивающие отображение подсказок*

Компонент	Свойство	Значение
button1	ToolTip on toolTip1	Выбор папки
button2	ToolTip on toolTip1	Назад
button3	ToolTip on toolTip1	Далее

Листинг 1.9. Модуль формы программы *Просмотр иллюстраций*

```
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;

using System.IO;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
```

```
// список jpg-файлов
List<string> imgList = new List<string>();
int nImg = 0;  // номер отображаемой иллюстрации

int pbw, pbh,      // первоначальный размер
    pbX, pbY;      // и положение pictureBox

string aPath;      // путь к файлам

public Form1()
{
    InitializeComponent();

    // запомнить размер
    // и положение pictureBox1
    pbh = pictureBox1.Height;
    pbw = pictureBox1.Width;
    pbX = pictureBox1.Location.X;
    pbY = pictureBox1.Location.Y;

    DirectoryInfo di; // каталог
    // получить имя каталога "Мои рисунки"
    di = new DirectoryInfo(Environment.GetFolderPath(
           Environment.SpecialFolder.MyPictures));
    aPath = di.FullName;

    // сформировать список иллюстраций
    FillListBox(aPath);
}

// формирует список иллюстраций
// aPath - путь к файлам иллюстраций
private Boolean FillListBox(string aPath)
{
```

```
        // информация о каталоге
        DirectoryInfo di =
            new DirectoryInfo(aPath);

        // информация о файлах
        FileInfo[] fi = di.GetFiles("*.jpg");

        // очистить список иллюстраций
        imgList.Clear();

        // добавляем в imgList имена jpg-файлов
        // каталога aPath
        foreach (FileInfo fc in fi)
        {
            imgList.Add(fc.Name);
        }

        if (fi.Length == 0) return false;
        else
        {
            nImg = 0;
            ShowPicture(aPath + "\\" + imgList[nImg]);

            // сделать недоступной кнопку Предыдущая
            button2.Enabled = false;

            // если в каталоге один jpg-файл,
            // сделать недоступной кнопку Следующая
            if (imgList.Count == 1)
                button3.Enabled = false;

            this.Text = aPath;

            return true;
        }
    }
```

```
// выводит иллюстрацию в поле компонента pictureBox1
private void ShowPicture(string aPicture)
{
    double mh, mw;    // коэффициенты масштабирования

    pictureBox1.Visible = false;
    pictureBox1.Left = pbX;

    // загружаем изображение в pictureBox1
    pictureBox1.SizeMode =
          PictureBoxSizeMode.AutoSize;
    pictureBox1.Image =
        new Bitmap(aPicture);

    // масштабируем, если нужно
    if ((pictureBox1.Image.Width > pbw) ||
        (pictureBox1.Image.Height > pbh))
    {
        pictureBox1.SizeMode =
            PictureBoxSizeMode.StretchImage;

        mh = (double)pbh /
              (double)pictureBox1.Image.Height;
        mw = (double)pbw /
              (double)pictureBox1.Image.Width;

        if (mh < mw)
        {
            // масштабируем по ширине
            pictureBox1.Width = Convert.ToInt16(
                    pictureBox1.Image.Width * mh);
            pictureBox1.Height = pbh;
        }
```

```
            else
            {
                // масштабируем по высоте
                pictureBox1.Width = pbw;
                pictureBox1.Height = Convert.ToInt16(
                        pictureBox1.Image.Height * mw);
            }
        }

        // разместить картинку в центре области
        // отображения иллюстраций
        pictureBox1.Left =
                 pbX + (pbw - pictureBox1.Width) / 2;
        pictureBox1.Top =
                 pbY + (pbh - pictureBox1.Height) / 2;

        pictureBox1.Visible = true;
    }

    // предыдущая картинка
    private void button2_Click(object sender, EventArgs e)
    {
        // если кнопка "Следующая" недоступна,
        // сделаем ее доступной
        if (!button3.Enabled)
            button3.Enabled = true;

        if (nImg > 0)
        {
            nImg--;
            ShowPicture(aPath + "\\" + imgList[nImg]);

            // отображается первая иллюстрация
            if (nImg == 0)
```

```
        {
            // теперь кнопка Предыдущая недоступна
            button2.Enabled = false;
        }
    }
}

// следующая картинка
private void button3_Click(object sender, EventArgs e)
{
    if (!button2.Enabled)
        button2.Enabled = true;

    if (nImg < imgList.Count)
    {
        nImg++;
        ShowPicture(aPath + "\\" + imgList[nImg]);
        if (nImg == imgList.Count-1)
        {
            button3.Enabled = false;
        }
    }
}

// щелчок на кнопке Обзор
private void button1_Click(object sender, EventArgs e)
{
    // FolderBrowserDialog - окно Обзор папок
    FolderBrowserDialog fb
                    = new FolderBrowserDialog();

    fb.Description =
          "Выберите папку,\n" +
          "в которой находятся иллюстрации";
```

```
            // кнопка Создать папку недоступна
            fb.ShowNewFolderButton = false;

            // "стартовая" папка
            fb.SelectedPath = aPath;

            // отображаем диалоговое окно
            if (fb.ShowDialog() == DialogResult.OK)
            {
                // пользователь выбрал каталог и
                // щелкнул на кнопке OK
                aPath = fb.SelectedPath;


                if (!FillListBox(fb.SelectedPath))
                    // в каталоге нет файлов иллюстраций
                    pictureBox1.Image = null;
            }
        }
    }
}
```

Секундомер

Программа **Секундомер** (рис. 1.10, листинг 1.10) демонстрирует использование компонента `Timer`.

Рис. 1.10. Форма программы **Секундомер**

Листинг 1.10. Модуль формы программы *Секундомер*

```
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 sek
{
    public partial class Form1 : Form
    {
        // минуты, секунды, миллисекунды
        int m, s, ms;

        public Form1()
        {
            InitializeComponent();

            // настройка таймера:
            // сигнал от таймера - каждые 0.5 секунды
            timer1.Interval = 500;

            // обнуление показаний
            m = 0; s = 0; ms = 0;

            label1.Text = "00";
            label2.Text = "00";
            label3.Visible = true;
        }
```

```
// щелчок на кнопке Пуск/Стоп
private void button1_Click(object sender, EventArgs e)
{
    if (timer1.Enabled)
    {
        // таймер работает,
        // остановить таймер
        timer1.Enabled = false;

        // изменить текст на кнопке
        // и сделать ее доступной
        button1.Text = "Пуск";
        button2.Enabled = true;
    }
    else
    {
        // таймер не работает,
        // запускаем таймер
        timer1.Enabled = true;

        // изменить текст на кнопке
        // и сделать ее недоступной
        button1.Text = "Стоп";
        button2.Enabled = false;
    }
}

// щелчок на кнопке Сброс
private void button2_Click(object sender, EventArgs e)
{
    m = 0; s = 0; ms = 0;

    label1.Text = "00";
    label2.Text = "00";
}
```

```
// сигнал от таймера
private void timer1_Tick(object sender, EventArgs e)
{
    // двоеточие мигает с периодом 0.5 сек
    if (label3.Visible)
    {
        if (s < 59)
        {
            s++;
            if (s < 10)
                label2.Text = "0" + s.ToString();
            else
                label2.Text = s.ToString();
        }
        else
        {
            if (m < 59)
            {
                m++;
                if (m < 10)
                    label1.Text = "0" + m.ToString();
                else
                    label1.Text = m.ToString();
                s = 0;
                label2.Text = "00";
            }
            else
            {
                m = 0;
                label1.Text = "00";
            }
        }
        label3.Visible = false;
    }
```

```
            else
                label3.Visible = true;
        }
    }
}
```

Таймер

Программа **Таймер** (рис. 1.11, листинг 1.11) демонстрирует использование компонентов `NumericUpDown` и `Timer`.

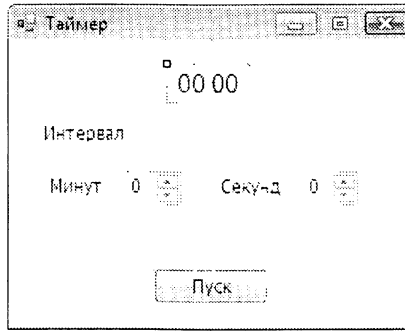


Рис. 1.11. Форма программы **Таймер**

Листинг 1.11. Модуль формы программы *Таймер*

```
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
    {
        private DateTime t1; // время запуска таймера
        private DateTime t2; // время срабатывания таймера

        public Form1()
        {
            InitializeComponent();

            // настройка компонентов numericUpDown
            numericUpDown1.Maximum = 59;
            numericUpDown1.Minimum = 0;
            // чтобы при появлении окна
            // курсор не мигал в поле редактирования
            numericUpDown1.TabStop = false;

            numericUpDown2.Maximum = 59;
            numericUpDown2.Minimum = 0;
            numericUpDown2.TabStop = false;

            // кнопка Пуск/Стоп недоступна
            button1.Enabled = false;
        }

        // обрабатывает событие ValueChanged от компонентов
        // numericUpDown1 и numericUpDown1
        private void numericUpDown1_ValueChanged
                                (object sender, EventArgs e)
        {
            if ((numericUpDown1.Value == 0) &&
```

```
                (numericUpDown2.Value == 0))
            button1.Enabled = false;
        else
            button1.Enabled = true;
    }

    // щелчок на кнопке Пуск/Стоп
    private void button1_Click(object sender, EventArgs e)
    {
        if (!timer1.Enabled)
        {
            // таймер не работает

            // t1 - текущее время
            // t2 = t1 + интервал
            t1 = new DateTime(DateTime.Now.Year,
                  DateTime.Now.Month, DateTime.Now.Day);
            t2 =
            t1.AddMinutes((double)numericUpDown1.Value);
            t2 =
            t2.AddSeconds((double)numericUpDown2.Value);

            groupBox1.Enabled = false;
            button1.Text = "Стоп";

            if (t2.Minute < 9)
               label1.Text =
               "0"+t2.Minute.ToString()+":";
            else
               label1.Text = t2.Minute.ToString() + ":";

            if (t2.Second < 9)
                label1.Text += "0" +
                t2.Second.ToString();
```

```
            else
                label1.Text += t2.Second.ToString();

            // сигнал от таймера поступает каждую секунду
            timer1.Interval = 1000;

            // пуск таймера
            timer1.Enabled = true;

            groupBox1.Visible = false;
        }
        else
        {
            // таймер работает, останавливаем
            timer1.Enabled = false;
            button1.Text = "Пуск";
            groupBox1.Enabled = true;
            numericUpDown1.Value = t2.Minute;
            numericUpDown2.Value = t2.Second;
        }
    }

    // сигнал от таймера
    private void timer1_Tick(object sender, EventArgs e)
    {
        t2 = t2.AddSeconds(-1);

        if (t2.Minute < 9)
           label1.Text = "0" + t2.Minute.ToString() + ":";
        else
           label1.Text = t2.Minute.ToString() + ":";

        if (t2.Second < 9)
            label1.Text += "0" + t2.Second.ToString();
```

```
            else
                label1.Text += t2.Second.ToString();

            if (Equals(t1, t2))
            {
                timer1.Enabled = false;
                MessageBox.Show(
                    "Заданный интервал времени истек",
                    "Таймер",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Information);

                button1.Text = "Пуск";
                groupBox1.Enabled = true;
                numericUpDown1.Value = 0;
                numericUpDown2.Value = 0;
            }
        }
    }
}
```

Параметры пользователя

Программа **Параметры пользователя** (рис. 1.12, листинг 1.12) показывает, как можно сохранить, а затем, при следующем запуске программы, загрузить значения параметров программы.

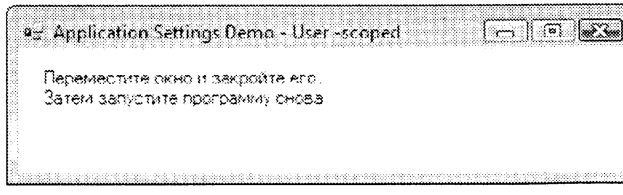


Рис. 1.12. Форма программы **Параметры пользователя**

Листинг 1.12. Модуль формы программы *Параметры пользователя*

```
/* Программа демонстрирует, как сохранить параметры программы
   в файле конфигурации. Файл конфигурации создает среда
   разработки в результате формирования списка параметров
   на вкладке Setteings (команда Project>Properties) */

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;

/* Пространство имен WindowsFormsApplication1.Properties
   определено в файле Settings.Desiner.cs, который создается
   автоматически в результате формирования списка параметров
   на вкладке Settings (команда Project>Properties).

  Список параметров программы:
  ---------------------------
  Name | Type | Scope |Value
  ---------------------------
  Left | int  | User  | 10
  Top  | int  | User  | 10
  ---------------------------
  Параметры используются для сохранения
  информации о положении формы.

  User-scoped параметры сохраняются в файле User.config,
  который находится в каталоге пользователя
```

```
   (C:\Users\User\AppData\Local\Application, где: User - имя
   пользователя; Application - имя приложения, которое
   сохраняет параметры). Файл конфигурации создает приложение
   в момент сохранения параметров. Для каждого пользователя
   создается свой файл конфигурации.
   Application-scoped параметры сохраняются каталоге
   приложения, в файле application.exe.config (где: application -
   имя выполняемого файла приложения). Файл Application-scoped
   параметров один для всех пользователей.
*/

using WindowsFormsApplication1.Properties;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // загрузить параметры из файла конфигурации
            this.Left = Settings.Default.Left;
            this.Top = Settings.Default.Top;
            this.label2.Text = Settings.Default.User;
        }

        private void Form1_FormClosed(object sender,
                                      FormClosedEventArgs e)
```

```
        {
            // сохранить текущие значения параметров в файле
            // конфигурации
            Settings.Default.Left = this.Left;
            Settings.Default.Top = this.Top;
            Settings.Default.Save();
        }
    }
}
```

Справочная информация

Программа **Чистый дисконтированный доход** (рис. 1.13, листинг 1.13) демонстрирует различные способы отображения справочной информации в CHM-формате. Окно справочной информации (рис. 1.14) появляется на экране в результате нажатия клавиши <F1> или щелчка на кнопке **Справка**.

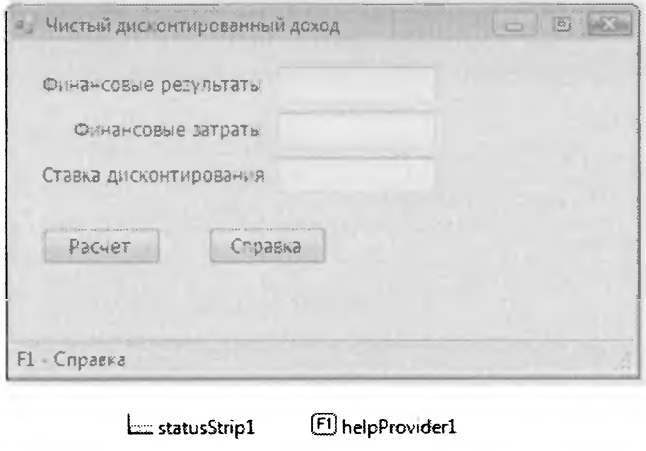


Рис. 1.13. Форма программы
Чистый дисконтированный доход

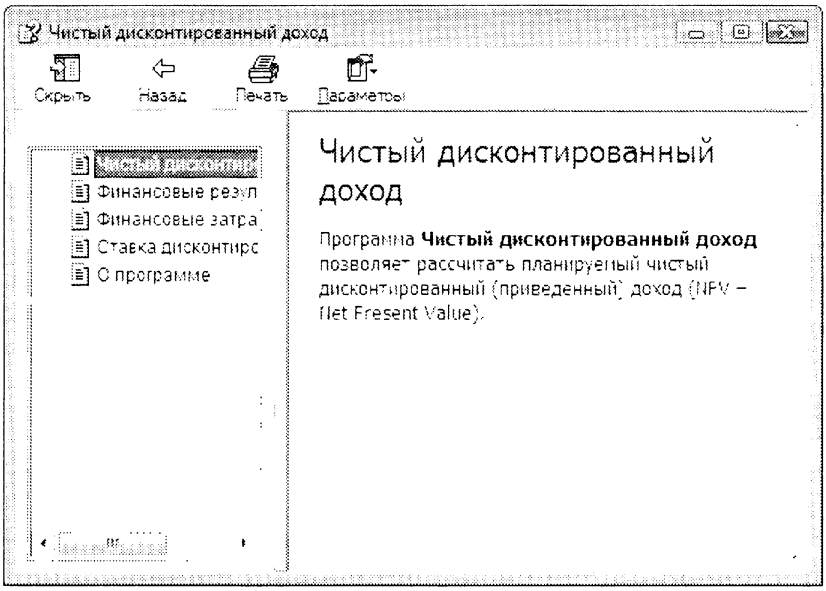


Рис. 1.14. Окно справочной информации программы **Чистый дисконтированный доход**

Листинг 1.13. Модуль формы программы *Чистый дисконтированный доход*

```
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 Form1()
    {
        InitializeComponent();

        // chm-файл получается путем компиляции htm-файлов,
        // в которых находится справочная информация.
        // Обычно каждый раздел справочной информации
        // помещают в отдельный файл. В дальнейшем
        // имя htm-файла используется в качестве
        // раздела идентификатора справочной информации.

        // файл справки приложения
        helpProvider1.HelpNamespace = "npv.chm";
        helpProvider1.SetHelpNavigator(this,
                    HelpNavigator.Topic);
        helpProvider1.SetShowHelp(this, true);

        // задать раздел справки
        // для textBox1 - Финансовые результаты
        helpProvider1.SetHelpKeyword(textBox1,
                                    "npv_02.htm");
        helpProvider1.SetHelpNavigator(textBox1,
                              HelpNavigator.Topic);
        helpProvider1.SetShowHelp(textBox1, true);

        // задать раздел справки
        // для textBox2 - Финансовые затраты
        helpProvider1.SetHelpKeyword(textBox2,
                                  "npv_03.htm");
        helpProvider1.SetHelpNavigator(textBox2,
                               HelpNavigator.Topic);
        helpProvider1.SetShowHelp(textBox2, true);
```

```
        // задать раздел справки
        // для textBox3 - Ставка дисконтирования
        helpProvider1.SetHelpKeyword(textBox3,
                                     "npv_04.htm");
        helpProvider1.SetHelpNavigator(textBox3,
                                HelpNavigator.Topic);
        helpProvider1.SetShowHelp(textBox3, true);
    }

    // щелчок на кнопке Справка
    private void button2_Click(object sender, EventArgs e)
    {
        Help.ShowHelp(this, helpProvider1.HelpNamespace,
              "npv_01.htm");
    }

    // щелчок на кнопке Расчет
    private void button1_Click(object sender, EventArgs e)
    {
        double p = 0; // поступления от продаж
        double r = 0; // расходы
        double d = 0; // ставка дисконтирования

        double npv = 0; // чистый дисконтированный доход

        try
        {
            p = Convert.ToDouble(textBox1.Text);
            r = Convert.ToDouble(textBox2.Text);
            d = Convert.ToDouble(textBox3.Text) / 100;

            npv = (p - r) / (1.0 + d);

            label4.Text =
               "Чистый дисконтированный доход (NPV) =  " +
```

```
                    npv.ToString("c");
                }
                catch
                {
                }
            }
        }
    }
```

Файлы

В этом разделе приведены программы, демонстрирующие выполнение операций с файлами.

Курс

Программа **Курс** (рис. 1.15, листинг 1.14) добавляет в базу данных, представляющую собой текстовый файл, информацию о текущем курсе доллара. Если файла данных в каталоге приложения нет, программа его создаст. Кнопка **Добавить** становится доступной после того как пользователь введет данные в поле редактирования. Для ввода даты используется компонент `DateTimePicker`.

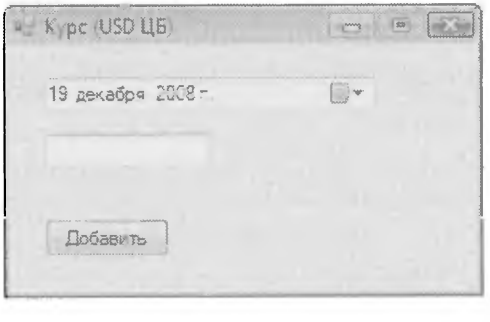


Рис. 1.15. Форма программы **Курс**

Листинг 1.14. Модуль формы программы *Курс*

```
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 Form1()
        {
            InitializeComponent();

            button1.Enabled = false;
        }

        // нажатие клавиши в поле редактирования
        private void textBox1_KeyPress(object sender,
                                       KeyPressEventArgs e)
        {
            if ((e.KeyChar >= '0') && (e.KeyChar <= '9'))
                return;

            if (e.KeyChar == '.')

                e.KeyChar = ',';

            if (e.KeyChar == ',')
```

```
    {
        // в поле редактирования не может
        // быть больше одной запятой и запятая
        // не может быть первым символом
        if ((textBox1.Text.IndexOf(',') != -1) ||
              (textBox1.Text.Length == 0))
        {
            e.Handled = true;
        }
        return;
    }

    if (Char.IsControl(e.KeyChar))
    {
        // <Enter>, <Backspace>, <Esc>
        if (e.KeyChar == (char)Keys.Enter)
            // установить курсор на кнопку OK
            button1.Focus();
        return;
    }
    // остальные символы запрещены
    e.Handled = true;
}

// изменилось содержимое поля редактирования
private void textBox1_TextChanged(object sender,
                                  EventArgs e)
{
    if (textBox1.Text.Length == 0)
        button1.Enabled = false;
    else
        button1.Enabled = true;
}
```

```
// щелчок на кнопке Добавить
private void button1_Click(object sender, EventArgs e)
{
    double kurs;   // курс
    DateTime date; // дата

    date = dateTimePicker1.Value;
    kurs =  System.Convert.ToDouble(textBox1.Text);

    // получить информацию о файле
    System.IO.FileInfo fi =
        new System.IO.FileInfo(
             Application.StartupPath + "\\usd.txt");

    // поток для записи
    System.IO.StreamWriter sw;

    if (fi.Exists) // файл данных существует?
        // откроем поток для добавления
        sw = fi.AppendText();
    else
        // создать файл и открыть поток для записи
        sw = fi.CreateText();

    // запись в файл
    sw.WriteLine(date.ToShortDateString());
    sw.WriteLine(kurs.ToString("N"));

    // закрыть поток
    sw.Close();

    // чтобы по ошибке не записать данные
    // второй раз, сделаем недоступным поле
    // ввода и кнопку
```

```
            button1.Enabled = false;
            textBox1.Enabled = false;
        }

        // пользователь выбрал другую дату
        private void dateTimePicker1_ValueChanged
                            (object sender, EventArgs e)
        {
            // очистить и сделать доступным
            // поле ввода
            textBox1.Enabled = true;
            textBox1.Clear();
            // установить курсор в поле ввода
            textBox1.Focus();
        }
    }
}
```

Котировки

Программа **Котировки** (рис. 1.16, листинг 1.15) демонстрирует чтение данных из текстового файла, а также использование компонентов `MonthCalendar` и `ListBox`.

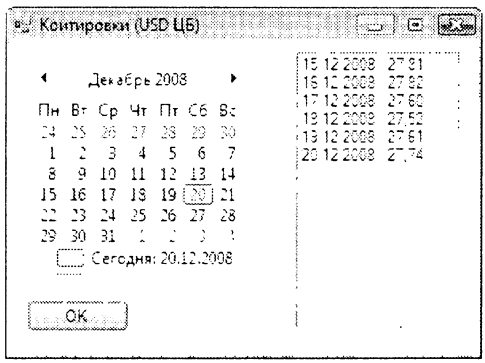


Рис. 1.16. Окно программы **Котировки**

Данные, удовлетворяющие критерию запроса (относящиеся к диапазону дат, выделенному в календаре) считываются из файла, сформированного программой **Курс**.

Листинг 1.15. Модуль формы программы *Котировки*

```
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 Form1()
        {
            InitializeComponent();
        }

        // щелчок на кнопке OK
        private void button1_Click(object sender, EventArgs e)
        {
            System.IO.StreamReader sr; // поток для чтения
            try
            {
                // создать поток для чтения
                sr = new System.IO.StreamReader(
                   Application.StartupPath + "\\usd.txt",
                     System.Text.Encoding.GetEncoding(1251));
```

```
        // начало и конец интервала, выделенного
        // на календаре
        DateTime dateStart =
                monthCalendar1.SelectionStart;
        DateTime dateEnd =
                monthCalendar1.SelectionEnd;

        string st1,st2 = "";
        DateTime date;

        listBox1.Items.Clear();

         // читаем данные из файла
        while ( ! sr.EndOfStream )
        {
            st1 = sr.ReadLine(); // дата как строка
            date = System.Convert.ToDateTime(st1);
            st2 = sr.ReadLine();

            if ((date >= dateStart) &&
                        (date <= dateEnd))
            {
                listBox1.Items.Add(st1 + "   " + st2);
            }
        }
        sr.Close();

        if (listBox1.Items.Count == 0)
        {
            listBox1.Items.Add("--- нет данных ---");
        }
    }

    catch(Exception exc)
    {
```

```
                MessageBox.Show
                    ("Ошибка доступа к файлу данных\n" +
                     exc.ToString(),
                     "Котировки",
                     MessageBoxButtons.OK,
                     MessageBoxIcon.Error);
                button1.Enabled = false;
            }
        }
    }
}
```

Редактор текста

Программа **NkEdit** (Редактор текста) (листинги 1.16, 1.17) демонстрирует использование компонентов `MenuStrip`, `TollStrip`, `OpenFileDialog`, `SaveFileDialog`, `FontDialog`, `PrintDialog`, а также отображение дочернего окна — диалога **О программе**.

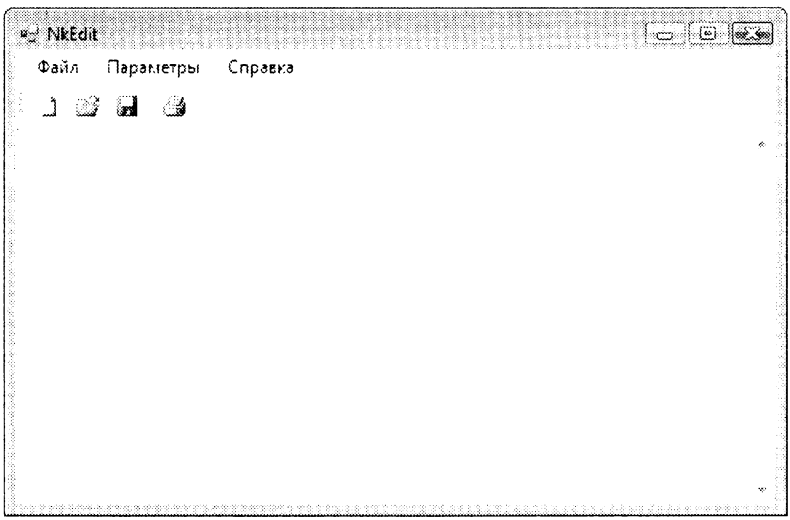


Рис. 1.17. Главная форма программы **NkEdit**

Информация о программе отображается в окне, которое появляется в результате выбора соответствующей команды в меню **Справка**. Главная форма приложения приведена на рис. 1.17, форма **О программе** — на рис. 1.18. Следует обратить внимание на то, что свойству `DialogResult` кнопки **OK** нужно присвоить значение `OK`.

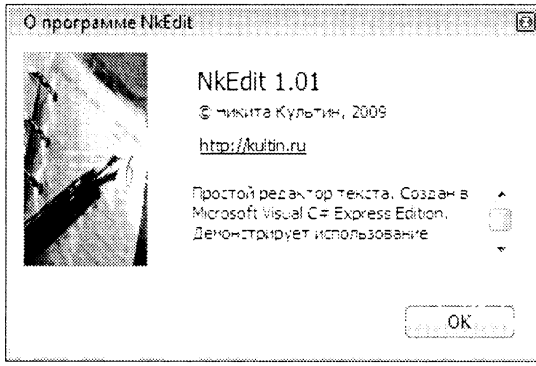


Рис. 1.18. Форма **О программе**

Листинг 1.16. Модуль главной формы программы *NkEdit*

```
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
```

```
{
    // имя файла
    private string fn = string.Empty;

    private bool docChanged = false;

    public Form1()
    {
        InitializeComponent();

        textBox1.ScrollBars = ScrollBars.Vertical;
        textBox1.Text = string.Empty;

        this.Text = "NkEdit - Новый документ";

        // отобразить панель инструментов
        toolStrip1.Visible = true;
        ParamToolStripMenuItem.Checked = true;

        // назначаем файл справки
        // helpProvider1.HelpNamespace = "nkedit.chm";

        // настройка компонента openDialog1
        openFileDialog1.DefaultExt = "txt";
        openFileDialog1.Filter = "текст|*.txt";
        openFileDialog1.Title = "Открыть документ";
        openFileDialog1.Multiselect = false;

        // настройка компонента saveDialog1
        saveFileDialog1.DefaultExt = "txt";
        saveFileDialog1.Filter = "текст|*.txt";
        saveFileDialog1.Title = "Сохранить документ";
    }
```

```
// открывает документ
private void OpenDocument()
{
  openFileDialog1.FileName = string.Empty;

  // отобразить диалог Открыть
  if (openFileDialog1.ShowDialog() ==DialogResult.OK)
  {
      fn = openFileDialog1.FileName;

      // отобразить имя файла в заголовке окна
      this.Text = fn;

      try
      {
         // считываем данные из файла
         System.IO.StreamReader sr =
               new System.IO.StreamReader(fn);

         textBox1.Text = sr.ReadToEnd();
         textBox1.SelectionStart =
                     textBox1.TextLength;

            sr.Close();
        }
        catch (Exception exc)
        {
          MessageBox.Show("Ошибка доступа к файлу\n" +
                exc.ToString(), "NkEdit",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }
}
```

```
// сохранить документ
private int SaveDocument()
{
    int result = 0;

    if (fn == string.Empty)
    {
       // отобразить диалог Сохранить
       if (saveFileDialog1.ShowDialog() ==
                        DialogResult.OK)
       {
          // отобразить имя файла в заголовке окна
          fn = saveFileDialog1.FileName;
                this.Text = fn;
       }
       else result = -1;
      }

      // сохранить файл
      if (fn != string.Empty)
      {
            try
            {
              // получим информацию о файле fn
              System.IO.FileInfo fi =
                    new System.IO.FileInfo(fn);

              // поток для записи
              System.IO.StreamWriter sw =
                     fi.CreateText();
              sw.Write(textBox1.Text);

              sw.Close(); // закрываем поток
              result = 0;
```

```
                }
                catch (Exception exc)
                {
                  MessageBox.Show(exc.ToString(),
                        "NkEdit",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
                }
            }
            return result;
}

    // выбор в меню Файл команды Создать
    private void FileCreateToolStripMenuItem_Click
                (object sender, EventArgs e)
    {
        if (docChanged)
        {
            DialogResult dr;
            dr = MessageBox.Show(
                  "Сохранить изменения ?", "NkEdit",
                  MessageBoxButtons.YesNoCancel,
                  MessageBoxIcon.Warning);
            switch (dr)
            {
                case DialogResult.Yes:
                    if (SaveDocument() == 0)
                    {
                        textBox1.Clear();
                        docChanged = false;
                    }
                    break;
                case DialogResult.No:
                    textBox1.Clear();
```

```
                docChanged = false;
                break;
            case DialogResult.Cancel:
                //
                break;
        };
    }
}

// выбор в меню Файл команды Открыть
private void FileOpenToolStripMenuItem_Click
                (object sender, EventArgs e)
{
    openFileDialog1.FileName = string.Empty;

    // отобразить диалог Открыть
    if (openFileDialog1.ShowDialog() ==
                        DialogResult.OK)
    {
        fn = openFileDialog1.FileName;

        // отобразить имя файла в заголовке окна
        this.Text = fn;

        try
        {
            // считываем данные из файла
            System.IO.StreamReader sr =
                new System.IO.StreamReader(fn);

            textBox1.Text = sr.ReadToEnd();
            textBox1.SelectionStart =
                  textBox1.TextLength;
            sr.Close();
```

```
            }
            catch (Exception exc)
            {
                MessageBox.Show("Ошибка чтения файла.\n" +
                    exc.ToString(), "MEdit",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }
        }
    }

    // выбор в меню Файл команды Сохранить
    private void FileSaveToolStripMenuItem_Click
                (object sender, EventArgs e)
    {
        SaveDocument();
    }

    // выбор в меню Файл команды Выход
    private void FileExitToolStripMenuItem_Click
                (object sender, EventArgs e)
    {
        this.Close();
    }

    // выбор в меню Параметры команды Панель инструментов
    private void ParamToolStripMenuItem_Click
            (object sender, EventArgs e)
    {
        // отобразить/скрыть панель инструментов
        toolStrip1.Visible = ! toolStrip1.Visible;
        ParamToolStripMenuItem.Checked =
                    ! ParamToolStripMenuItem.Checked;
    }
```

```
// выбор в меню Параметры команды Шрифт
private void ParamFontToolStripMenuItem_Click
          (object sender, EventArgs e)
{
    fontDialog1.Font = textBox1.Font;
    if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
        textBox1.Font = fontDialog1.Font;
    }
}

private void textBox1_TextChanged(object sender,
                              EventArgs e)
{
    docChanged = true;
}

// пользователь сделал щелчок на кнопке "Закрыть окно"
private void Form1_FormClosing(object sender,
                  FormClosingEventArgs e)
{
    if (docChanged )
    {
        DialogResult dr;
        dr = MessageBox.Show("Сохранить изменения?",
                        "NkEdit",

 MessageBoxButtons.YesNoCancel,
                        MessageBoxIcon.Warning);
        switch ( dr )
        {
            case DialogResult.Yes :
                if ( SaveDocument() != 0)
                // пользователь отменил операцию
                //  сохранения файла
```

```
                    e.Cancel = true; // отменить закрытие
                                     // окна программы
                    break;
                case DialogResult.No: ;
                    break;
                case DialogResult.Cancel:
                    // отменить закрытие окна программы
                    e.Cancel = true;
                    break;
            };
        }
    }

    // выбор в меню Справка команды О программе
    private void оПрограммаToolStripMenuItem_Click
                        (object sender, EventArgs e)
    {
        Form2 about = new Form2();

        about.ShowDialog();
    }

    // выбор в меню Файл команды Печать
    private void печатьToolStripMenuItem_Click
                (object sender, EventArgs e)
    {
        printDialog1.ShowDialog();
    }
}
}
```

Листинг 1.17. Модуль формы *О программе*

```
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 Form2()
        {
            InitializeComponent();
        }

        // щелчок на WEB-ссылке
        private void linkLabel1_LinkClicked(object sender,
                        LinkLabelLinkClickedEventArgs e)
        {
          // System.Diagnostics.Process.Start("IEXPLORE.EXE",
          //                                  "http://kultin.ru");
          string webRef = linkLabel1.Text;
          System.Diagnostics.Process.Start(webRef);
        }

    }
}
```

TaskDialog

В приложениях, работающих в Windows Vista, для вывода сообщений можно использовать диалог **TaskDialog** (рис. 1.19). Сле-

дующая программа, ее форма приведена на рис. 1.20, а текст — в листинге 1.18, показывает, как это сделать.

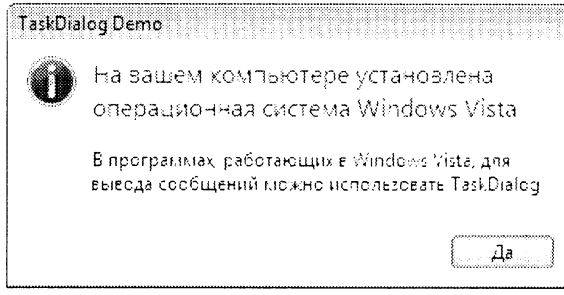


Рис. 1.19. Пример сообщения, выведенного при помощи **TaskDialog**

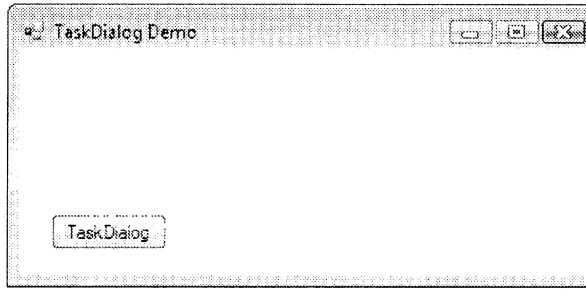


Рис. 1.20. Форма программы **TaskDialog Demo**

Листинг 1.18. Модуль формы программы *TaskDialog Demo*

```
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;

using System.Runtime.InteropServices;

namespace WindowsFormsApplication1
{
    class SimpleTaskDialog
    {
        [DllImport("comctl32.dll", CharSet = CharSet.Unicode,
                    EntryPoint = "TaskDialog")]
       static extern int _TaskDialog(IntPtr hWndParent,
       IntPtr hInstance, String pszWindowTitle,
       String pszMainInstruction, String pszContent,
       int dwCommonButtons, IntPtr pszIcon, out int
       pnButton);

        [Flags]
        public enum TaskDialogButtons
        {
            OK = 0x0001,
            Cancel = 0x0008,
            Yes = 0x0002,
            No = 0x0004,
            Retry = 0x0010,
            Close = 0x0020
        }

        public enum TaskDialogIcon
        {
            Information = UInt16.MaxValue - 2,
            Warning = UInt16.MaxValue,
            Stop = UInt16.MaxValue - 1,
            Question = 0,
            SecurityWarning = UInt16.MaxValue - 5,
```

```
        SecurityError = UInt16.MaxValue - 6,
        SecuritySuccess = UInt16.MaxValue - 7,
        SecurityShield = UInt16.MaxValue - 3,
        SecurityShieldBlue = UInt16.MaxValue - 4,
        SecurityShieldGray = UInt16.MaxValue - 6
    }

    public enum TaskDialogResult
    {
        None, OK, Cancel, Yes, No, Retry, Close
    }

    private static TaskDialogResult ShowInternal(
      IntPtr owner, string text, string instruction,
      string caption, TaskDialogButtons buttons,
      TaskDialogIcon icon)
    {
        int p;
        if (_TaskDialog(owner, IntPtr.Zero, caption,
             instruction, text, (int)buttons,
             new IntPtr((int)icon), out p) != 0)
            throw new InvalidOperationException
               ("Something weird has happened.");

        switch (p)
        {
            case 1: return TaskDialogResult.OK;
            case 2: return TaskDialogResult.Cancel;
            case 4: return TaskDialogResult.Retry;
            case 6: return TaskDialogResult.Yes;
            case 7: return TaskDialogResult.No;
            case 8: return TaskDialogResult.Close;
            default: return TaskDialogResult.None;
        }
    }
```

```
public static TaskDialogResult Show
       (IWin32Window owner, string text)
{
    return  Show(owner, text, null, null,
                  TaskDialogButtons.OK);
}

public static TaskDialogResult Show
(IWin32Window owner, string text, string instruction)
{
    return Show(owner, text, instruction, null,
                TaskDialogButtons.OK, 0);
}

public static TaskDialogResult Show
(IWin32Window owner, string text, string instruction,
 string caption)
{
    return Show(owner, text, instruction, caption,
                TaskDialogButtons.OK, 0);
}

public static TaskDialogResult Show
 (IWin32Window owner, string text, string
  instruction,
   string caption, TaskDialogButtons buttons)
{
    return Show(owner, text, instruction, caption,
                 buttons, 0);
}

public static TaskDialogResult Show
(IWin32Window owner, string text, string instruction,
  string caption, TaskDialogButtons buttons,
```

```
    TaskDialogIcon icon)
{
    return ShowInternal(owner.Handle, text,
        instruction, caption, buttons, icon);
}

public static TaskDialogResult Show(string text)
{
    return Show(text, null, null,
                TaskDialogButtons.OK);
}

public static TaskDialogResult Show
        (string text, string instruction)
{
    return Show(text, instruction, null,
                TaskDialogButtons.OK, 0);
}

public static TaskDialogResult Show
 (string text, string instruction, string caption)
{
    return Show(text, instruction, caption,
            TaskDialogButtons.OK, 0);
}

public static TaskDialogResult Show(string text,
              string instruction, string caption,
              TaskDialogButtons buttons)
{
  return Show(text, instruction, caption, buttons, 0);
}

public static TaskDialogResult Show(string text,
     string instruction, string caption,
```

```
            TaskDialogButtons buttons, TaskDialogIcon icon)
        {
            return ShowInternal(IntPtr.Zero, text,
                instruction, caption, buttons, icon);
        }
    }

    class OSVersion
    {
        [StructLayout(LayoutKind.Sequential)]
        public struct OSVERSIONINFO
        {
            public int dwOSVersionInfoSize;
            public int dwMajorVersion;
            public int dwMinorVersion;
            public int dwBuildNumber;
            public int dwPlatformId;
            [MarshalAs(UnmanagedType.ByValTStr,
            SizeConst = 128)]
            public string szCSDVersion;
        }

        // ----------------------- -------------
        //     OS                  Major.Minor
        // ------- ---------------------------------
        // Windows Server 2008      6.0
        // Windows Vista            6.0
        // Windows Server 2003 R2   5.2
        // Windows Server 2003      5.2
        // Windows XP               5.1
        // Windows 2000             5.0
        // ------- - ------------- -------------
```

```
    [DllImport("kernel32.Dll")]

    public static extern short GetVersionEx
                            (ref OSVERSIONINFO o);

    static public string GetOSVersionInfo()
    {
        OSVERSIONINFO os = new OSVERSIONINFO();
        os.dwOSVersionInfoSize =
                    Marshal.SizeOf(typeof(OSVERSIONINFO));
        GetVersionEx(ref os);
        return os.dwMajorVersion.ToString() + "." +
                   os.dwMinorVersion.ToString();
    }

    static public int GetMajorVersion()
    {
        OSVERSIONINFO os = new OSVERSIONINFO();
        os.dwOSVersionInfoSize =
                    Marshal.SizeOf(typeof(OSVERSIONINFO));
        GetVersionEx(ref os);
        return os.dwMajorVersion;
    }

    static public int GetMinorVersion()
    {
        OSVERSIONINFO os = new OSVERSIONINFO();
        os.dwOSVersionInfoSize =
                Marshal.SizeOf(typeof(OSVERSIONINFO));
        GetVersionEx(ref os);
        return os.dwMinorVersion;
    }
}
```

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if ( OSVersion.GetMajorVersion() == 6)
                SimpleTaskDialog.Show(this,
          "В программах, работающих в Windows Vista, для " +
          " вывода сообщений можно использовать TaskDialog",
          "На вашем компьютере установлена ОС Windows Vista",
          "TaskDialog Demo",
          SimpleTaskDialog.TaskDialogButtons.Yes,
          SimpleTaskDialog.TaskDialogIcon.Information);
            else
                MessageBox.Show(
        "TaskDialog можно использовать только в программах," +
        "работающих в Windows Vista",
        "TaskDialog Demo", MessageBoxButtons.OK,
         MessageBoxIcon.Information);
        }
    }
}
```

Графика

В этом разделе собраны программы, демонстрирующие работу с графикой.

Общие замечания

- Программа может вывести графику на поверхность формы, компонента или сформировать ее на поверхности объекта `Graphics`.
- Основную работу по формированию графики должна выполнять функция обработки события `Paint`.
- Для того чтобы на графической поверхности объекта (например, формы) появилась линия, окружность, прямоугольник, а так же текст или загруженная из файла иллюстрация, необходимо применить к свойству `Graphics` этого объекта соответствующий метод.
- Цвет линии, рисуемой методом, задается путем указания в качестве его параметра *карандаша* (объект `Pen`), цвет заливки контура — кисти (объект `Brush`).
- В качестве параметра `Pen` можно указать стандартный карандаш (в распоряжении программиста есть набор из более чем 100 цветных карандашей) или карандаш, созданный программистом.
- В качестве параметра `Brush` можно указать стандартную кисть (в распоряжении программиста есть набор из более чем 100 цветных кистей) или кисть, созданную программистом.
- Вид шрифта, выводимого методом `DrawString`, определяется значением параметра `Font`, цвет — значением параметра `Brush`.

Вывод текста

Программа **DrawString Demo** (рис. 1.21, листинг 1.19) демонстрирует вывод текста на графическую поверхность формы.

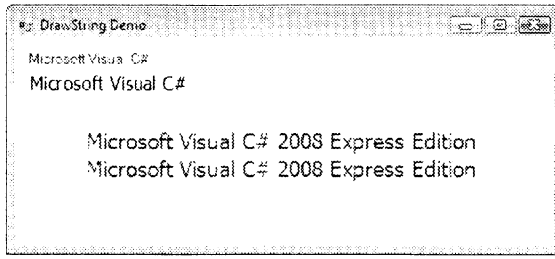


Рис. 1.21. Окно программы **DrawString Demo**

Листинг 1.19. Модуль формы программы *DrawString Demo*

```
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 Form1()
        {
            InitializeComponent();
        }

        // обработка события Paint формы
        private void Form1_Paint(object sender,
                                 PaintEventArgs e)
        {
```

```
// вывод текста шрифтом, заданным значением
// свойства Font формы
e.Graphics.DrawString("Microsoft Visual C#",
            this.Font, Brushes.DarkGreen, 10, 10);

// вывод текста шрифтом, созданным программистом:
Font aFont = new Font("Tahoma", 12,
                      FontStyle.Regular);
e.Graphics.DrawString("Microsoft Visual C#",
                     aFont, Brushes.Black, 10, 30);

// вывод текста в центре формы
Font hFont = new Font("Tahoma", 14,
                      FontStyle.Regular);
string header =
       "Microsoft Visual C# 2008 Express Edition";

// размер области отображения текста зависит от
// характеристик шрифта, которым он отображается

// определить размер области отображения текста
int w = (int)e.Graphics.
            MeasureString(header, hFont).Width;
int h = (int)e.Graphics.
            MeasureString(header, hFont).Height;

// вычислить координаты левого верхнего угла
// области отображения текста, так чтобы текст был
// размещен в центре формы по горизонтали и
// вертикали. ClientSize.Width - размер внутренней
// области формы
int x = (this.ClientSize.Width - w) / 2;
int y = (this.ClientSize.Height - h) / 2;
```

```
            // вывести текст в центре формы
            e.Graphics.DrawString(header, hFont,
                System.Drawing.Brushes.DarkGreen, x, y);

            // выведем текст еще раз
            e.Graphics.DrawString(header, hFont,
                System.Drawing.Brushes.DarkGreen, x, y + h);
        }

        // изменился размер окна
        private void Form1_Resize(object sender, EventArgs e)
        {
            // сообщить системе о необходимости
            // перерисовать окно. В результате будет
            // сгенерировано событие Paint.
            this.Refresh();
        }
    }
}
```

Диаграмма

В окне программы **Диаграмма** (рис. 1.22) отображается диаграмма изменения курса доллара (данные загружаются из файла).

Рис. 1.22. Окно программы **Диаграмма**

Программа (листинг 1.20) демонстрирует использование методов DrawString, DrawRectangle, FillRectangle, а также работу с массивами и чтение данных из файла.

Листинг 1.20. Модуль формы программы *Диаграмма*

```
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 Diagram
{
    public partial class Form1 : Form
    {
        // данные
        private double[] d;

        // строит график
        private void drawDiagram(object sender,
              PaintEventArgs e)
        {
            // графическая поверхность
            Graphics g = e.Graphics;

            // шрифт подписей данных
            Font dFont = new Font("Tahoma", 9);

            // ** заголовок **
            // шрифт заголовка
```

```
Font hFont = new Font("Tahoma", 14,
                      FontStyle.Regular);
string header = "Изменение курса доллара";

// ширина области отображения текста
int wh =(int)g.MeasureString(header,
hFont).Width;

int x = (this.ClientSize.Width - wh) / 2;

g.DrawString(header,
    hFont, System.Drawing.Brushes.DarkGreen,
    x, 5);

/*
 * область построения диаграммы:
 * - отступ сверху - 100;
 * - отступ снизу - 20;
 * - отступ слева - 20;
 * - отступ справа - 20.
 *
 * ClientSize - размер внутренней области окна
 *
 * график строим в отклонениях от минимального
 * значения ряда данных, так, чтобы он занимал
 * всю область построения
 */

double max = d[0]; // максимальный эл-т массива
double min = d[0]; // минимальный эл-т массива

for (int i = 1; i < d.Length; i++)
{
    if (d[i] > max) max = d[i];
```

```
        if (d[i] < min) min = d[i];
    }

    // рисуем диаграмму
    int x1, y1; // координаты левого верхнего
                //  угла столбика
    int w, h;   // размер столбца

    // ширина столбиков диаграммы
    // 5 - расстояние между столбиками
    // d.Length - кол-во рядов данных (столбиков)
    w = (ClientSize.Width - 40 - 5 * (d.Length - 1))
                                / d.Length;
    x1 = 20;
    for (int i = 0; i < d.Length; i++)
    {
        y1 = this.ClientSize.Height - 20 -
        (int)((this.ClientSize.Height - 100) *
        (d[i] - min) / (max - min));

        // подпись численного значения первой точки
        g.DrawString(Convert.ToString(d[i]),
            dFont, System.Drawing.Brushes.Black,
            x1, y1 - 20);

        // рисуем столбик
        h = ClientSize.Height - y1 - 20 + 1;

        // зеленый прямоугольник
        g.FillRectangle(Brushes.ForestGreen,
                        x1, y1, w, h);

        // контур прямоугольника
        g.DrawRectangle(System.Drawing.Pens.Black,
                        x1, y1, w, h);
```

```
            x1 = x1 + w +5;
        }
    }

    public Form1()
    {
        InitializeComponent();

        // чтение данных из файла в массив
        System.IO.StreamReader sr; // поток для чтения
        try
        {
            sr = new System.IO.StreamReader(
                Application.StartupPath + "\\usd.dat");

            // создаем массив
            d = new double[10];

            // читаем данные из файла
            int i = 0;
            string t = sr.ReadLine();
            while ((t != null) && (i < d.Length))
            {
                // записываем считанное число в массив
                d[i++] = Convert.ToDouble(t);
                t = sr.ReadLine();
            }

            // закрываем поток
            sr.Close();

            // задаем функцию обработки события Paint
            this.Paint
                  += new PaintEventHandler(drawDiagram);
```

```
            }
            // обработка исключений:
            // - файл данных не найден
            catch (System.IO.FileNotFoundException ex)
            {
                MessageBox.Show( ex.Message + "\n" +
                    "("+ ex.GetType().ToString() +")",
                    "График",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
            }

            // - другие исключения
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "",

                MessageBoxButtons.OK,MessageBoxIcon.Stop);
            }
        }

        private void Form1_SizeChanged(object sender,
                                       EventArgs e)
        {
            this.Refresh();
        }
    }
}
```

График

В окне программы **График** (рис. 1.23) отображается график изменения курса доллара. Данные загружаются из файла. Текст программы приведен в листинге 1.21.

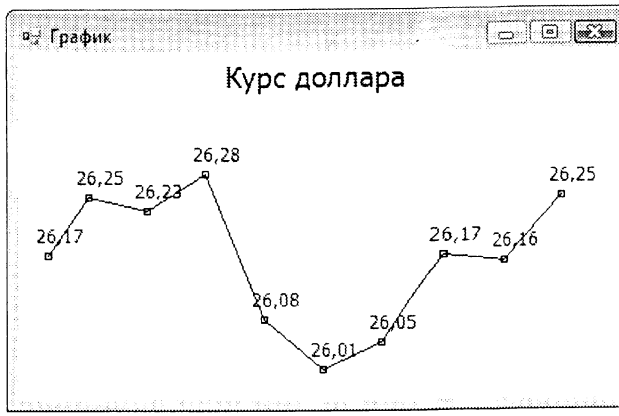


Рис. 1.23. Окно программы **График**

Листинг 1.21. Модуль формы программы *График*

```
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 graph
{
    public partial class Form1 : Form
    {
        // данные
        private double[] d;

        // строит график
        private void drawDiagram(object sender,
```

```
                                PaintEventArgs e)
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт подписей данных
    Font dFont = new Font("Tahoma", 9);

    // ** заголовок **
    // шрифт заголовка
    Font hFont = new Font("Tahoma", 14,
                          FontStyle.Regular);
    string header = "Курс доллара";

    // ширина области отображения текста
    int w = (int)g.MeasureString(header, hFont).Width;

    int x = (this.ClientSize.Width - w) / 2;

    g.DrawString(header,
        hFont, System.Drawing.Brushes.Black,
        x, 5);

    /*
     * область построения графика:
     * - отступ сверху - 100;
     * - отступ снизу - 20;
     * - отступ слева - 20;
     * - отступ справа - 20.
     *
     * ClientSize - размер внутренней области окна
     *
     * график строим в отклонениях от минимального
     * значения ряда данных, так, чтобы он занимал
```

```
 *  всю область построения
 */

// расстояние между точками графика (шаг по X )
int sw = (int)((this.ClientSize.Width - 40) /
                (d.Length - 1));

double max = d[0]; // максимальный эл-т массива
double min = d[0]; // минимальный эл-т массива

for (int i = 1; i < d.Length; i++)
{
    if (d[i] > max) max = d[i];
    if (d[i] < min) min = d[i];
}

// рисуем график
int x1, y1, x2, y2;

// первая точка
x1 = 20;
y1 = this.ClientSize.Height - 20 -
    (int)((this.ClientSize.Height - 100) *
    (d[0] - min) / (max - min));

// маркер первой точки
g.DrawRectangle(System.Drawing.Pens.Black,
    x1 - 2, y1 - 2, 4, 4);

// подпись численного значения первой точки
g.DrawString(Convert.ToString(d[0]),
    dFont, System.Drawing.Brushes.Black,
    x1 - 10, y1 - 20);
```

```
        // остальные точки
        for (int i = 1; i < d.Length; i++)
        {

            x2 = 8 + i * sw;
            y2 = this.ClientSize.Height - 20 -
                (int)((this.ClientSize.Height - 100) *
                (d[i] - min) / (max - min));

            // маркер точки
            g.DrawRectangle(System.Drawing.Pens.Black,
                  x2 - 2, y2 - 2, 4, 4);

            // соединим текущую точку с предыдущей
            g.DrawLine(System.Drawing.Pens.Black,
                x1, y1, x2, y2);

            // подпись численного значения
            g.DrawString(Convert.ToString(d[i]),
                dFont, System.Drawing.Brushes.Black,
                x2 - 10, y2 - 20);

            x1 = x2;
            y1 = y2;
        }
    }

    public Form1()
    {
        InitializeComponent();

        // чтение данных из файла в массив
        System.IO.StreamReader sr; // поток для чтения
        try
```

```
{
    // создаем поток для чтения
    sr = new System.IO.StreamReader(
        Application.StartupPath + "\\usd.dat");

    // создаем массив
    d = new double[10];

    // читаем данные из файла
    int i = 0;
    string t = sr.ReadLine();
    while ((t != null) && (i < d.Length))
    {
        // записываем считанное число в массив
        d[i++] = Convert.ToDouble(t);
        t = sr.ReadLine();
    }

    // закрываем поток
    sr.Close();

    // задаем функцию обработки события Paint
    this.Paint +=
        new PaintEventHandler(drawDiagram);
}
// обработка исключений:
// - файл данных не найден
catch (System.IO.FileNotFoundException ex)
{
    MessageBox.Show( ex.Message + "\n" +
        "("+ ex.GetType().ToString() +")",
        "График",
        MessageBoxButtons.OK,
        MessageBoxIcon.Error);
}
```

```
            // - другие исключения
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "",

                MessageBoxButtons.OK,MessageBoxIcon.Stop);
            }
        }

        private void Form1_SizeChanged(object sender,
                                       EventArgs e)
        {
            this.Refresh();
        }
    }
}
```

Круговая диаграмма

В окне программы **Круговая диаграмма** (рис. 1.24) отображаются результаты социологического опроса.

Рис. 1.24. Окно программы **Круговая диаграмма**

Исходные данные (вопрос, варианты ответа и количество ответов) загружаются из файла. Программа (листинг 1.22) обрабатывает данные (вычисляет долю каждой категории в общей сумме) и строит диаграмму.

Листинг 1.22. Модуль формы программы *Круговая диаграмма*

```
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
    {
        // заголовок диаграммы
        string header;

        // количество элементов данных
        int N = 0;

        double[] dat; // ряд данных
        double[] p;   // доля категории в общей сумме

        // подписи данных
        private string[] title;


        public Form1()
        {
```

```
InitializeComponent();

try
{
    System.IO.StreamReader sr;

    sr = new System.IO.StreamReader(
        Application.StartupPath + "\\date.dat",
        System.Text.Encoding.GetEncoding(1251));

    // считываем заголовок диаграммы
    header = sr.ReadLine();

    // считываем данные о количестве записей
    // и инициализируем массивы
    N = Convert.ToInt16(sr.ReadLine());
    dat = new double[N];
    p = new double[N];
    title = new string[N];

    // читаем данные
    int i = 0;
    string st;
    st = sr.ReadLine();
    while ((st != null) && (i < N))
    {
        title[i] = st;

        st = sr.ReadLine();
        dat[i++] = Convert.ToDouble(st);

        //i++;

        st = sr.ReadLine();
    }
```

```
            // закрываем поток
            sr.Close();

            // задать процедуру обработки события Paint
            this.Paint += new PaintEventHandler(Diagram);

            double sum = 0;
            int j = 0;

            // вычислить сумму
            for (j = 0; j < N; j++)
                sum += dat[j];

            // вычислить долю каждой категории
            for (j = 0; j < N; j++)
                p[j] = (double)(dat[j] / sum);
        }

        // задаем цвет формы
        //this.BackColor = System.Drawing.Color.White;

        catch ( Exception ex)
        {
            MessageBox.Show(ex.Message,
                "Диаграмма",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }

    // процедура рисует круговую диаграмму;
    // имеет такие же параметры, что и
    // процедура обработки события Paint
    private void Diagram(object sender, PaintEventArgs e)
```

```
{
    // графическая поверхность
    Graphics g = e.Graphics;

    // шрифт заголовка
    Font hFont = new Font("Tahoma", 12);

    // выводим заголовок
    int w = (int)g.MeasureString(header,
    hFont).Width;
    int x = (this.ClientSize.Width - w) / 2;

    g.DrawString(header,
        hFont, System.Drawing.Brushes.Black, x, 10);

    // шрифт легенды
    Font lFont = new Font("Tahoma", 9);

    // диаметр диаграммы
    int d = ClientSize.Height - 70;

    int x0 = 30;
    int y0 = (ClientSize.Height - d) / 2 + 10;

    // координаты верхнего левого угла
    // области легенды
    int lx = 60 + d;
    int ly = y0 + (d - N * 20 + 10) / 2;

    // длина дуги сектора
    int swe;

    // кисть для заливки сектора диаграммы
    Brush fBrush = Brushes.White;
```

```
// начальная точка дуги сектора
int sta = -90;

// рисуем диаграмму
for (int i = 0; i < N; i++)
{
    // длинна дуги
    swe = (int)(360 * p[i]);

    // задать цвет сектора
    switch (i)
    {
        case 0:
            fBrush = Brushes.YellowGreen; break;
        case 1:
            fBrush = Brushes.Gold; break;
        case 2:
            fBrush = Brushes.Pink; break;
        case 3:
            fBrush = Brushes.Violet; break;
        case 4:
            fBrush = Brushes.OrangeRed; break;
        case 5:
            fBrush = Brushes.RoyalBlue; break;
        case 6:
            fBrush = Brushes.SteelBlue; break;
        case 7:
            fBrush = Brushes.Chocolate; break;
        case 8:
            fBrush = Brushes.LightGray; break;
    }

    // из-за округления возможна ситуация,
    // при которой будет промежуток между
```

```
                // последним и первым секторами
                if (i == N - 1)
                {
                    // последний сектор
                    swe = 270 - sta;
                }

                // рисуем сектор
                g.FillPie(fBrush, x0, y0, d, d, sta, swe);

                // рисуем границу сектора
                g.DrawPie(System.Drawing.Pens.Black, x0, y0,
                        d, d, sta, swe);

                // прямоугольник легенда
                g.FillRectangle(fBrush,
                          lx, ly + i * 20, 20, 10);
                g.DrawRectangle(System.Drawing.Pens.Black,
                          lx, ly + i * 20, 20, 10);

                // подпись
                g.DrawString( title[i] + " - " +
                          p[i].ToString("P"),
                          lFont,
                          System.Drawing.Brushes.Black,
                          lx + 24, ly + i * 20 - 3);

                // начальная точка дуги для следующего
                // сектора
                sta = sta + swe;
            }
        }
    }
}
```

Кисти

Программа **Кисти** (рис. 1.25, листинг 1.23) демонстрирует работу с кистями различных типов. Следует обратить внимание на то, что для того, чтобы можно было использовать градиентную и текстурную кисти, в программу нужно добавить ссылку на пространство имен `System.Drawing.Drawing2D`.

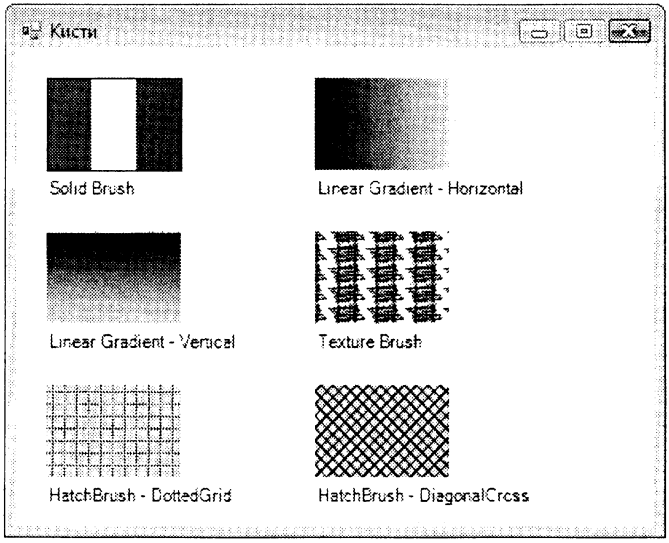


Рис. 1.25. Окно программы **Кисти**

Листинг 1.23. Модуль формы программы *Кисти*

```
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;

using System.Drawing.Drawing2D;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender,
                                 PaintEventArgs e)
        {
            // Solid Brush
            e.Graphics.FillRectangle(Brushes.ForestGreen,
                                     20, 20, 30, 60);
            e.Graphics.FillRectangle(Brushes.White,
                                     50, 20, 30, 60);
            e.Graphics.FillRectangle(Brushes.Red,
                                     80, 20, 30, 60);
            e.Graphics.DrawRectangle(Pens.Black,
                                     20, 20, 90, 60);
            e.Graphics.DrawString("Solid Brush", this.Font,
                                  Brushes.Black, 20, 85);

            // градиентная кисть LinearGradientBrush
            LinearGradientBrush lgBrush_1 =
                new LinearGradientBrush(
                    new RectangleF(200, 20, 90, 10),
                    Color.Blue, Color.LightBlue,
                    LinearGradientMode.Horizontal);
```

```
e.Graphics.FillRectangle(lgBrush_1,
                         200, 20, 90, 60);
e.Graphics.DrawString(
     "Linear Gradient - Horizontal", this.Font,
     Brushes.Black, 200, 85);

// градиентная кисть LinearGradientBrush
LinearGradientBrush lgBrush_2 =
    new LinearGradientBrush(
        new RectangleF(0, 0, 10, 60),
        Color.Blue, Color.LightBlue,
        LinearGradientMode.Vertical);

e.Graphics.FillRectangle(lgBrush_2,
                         20, 120, 90, 60);
e.Graphics.DrawString(
       "Linear Gradient - Vertical", this.Font,
       Brushes.Black, 20, 185);

// текстурная кисть
try
{
   TextureBrush tBrush =
        new TextureBrush
         (Image.FromFile("brush_2.bmp"));

    e.Graphics.FillRectangle(tBrush,
                             200, 120, 90, 60);
}
catch (System.IO.FileNotFoundException)
{
    e.Graphics.DrawRectangle(Pens.Black,
           200, 120, 90, 60);
```

```
            e.Graphics.DrawString("Source image",
                                  this.Font,
                    Brushes.Black, 160, 135);
            e.Graphics.DrawString(" not found",
                this.Font, Brushes.Black, 170, 200);
        }
        e.Graphics.DrawString("Texture Brush", this.Font,
                              Brushes.Black, 200, 185);

        // штриховка (HatchBrush кисть)
        HatchBrush hBrush =
            new HatchBrush(HatchStyle.DottedGrid,
                Color.Black, Color.Gold);

        e.Graphics.FillRectangle(hBrush, 20, 220, 90, 60);
        e.Graphics.DrawString("HatchBrush - DottedGrid",
                this.Font, Brushes.Black, 20, 285);

        HatchBrush hBrush_2 =
            new HatchBrush(HatchStyle.DiagonalCross,
                Color.Black, Color.Gold);

        e.Graphics.FillRectangle(hBrush_2,
                                 200, 220, 90, 60);
        e.Graphics.DrawString(
             "HatchBrush - DiagonalCross", this.Font,
             Brushes.Black, 200, 285);
    }
}
```

Бегущая строка

Программа **Бегущая строка** (рис. 1.26, листинг 1.24) демонстрирует использование битового образа для отображения баннера в

стиле "бегущей" строки. При наведении курсора мыши на баннер процесс прокрутки приостанавливается.

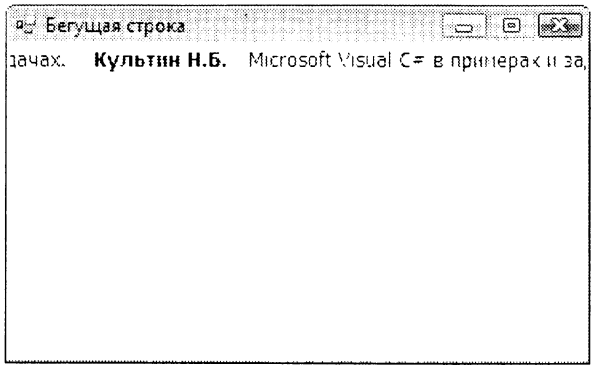


Рис. 1.26. Окно программы **Бегущая строка**

Листинг 1.24. Модуль формы программы *Бегущая строка*

```
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
    {
        // графическая поверхность
        Graphics g;
```

```
// баннер
Bitmap baner;

// область вывода баннера
Rectangle rct;

public Form1()
{
    InitializeComponent();

    try
    {
        // загружаем файл баннера
        baner = new Bitmap("banner.png");
    }
    catch (Exception exception)
    {
        MessageBox.Show(
           "Ошибка загрузки файла баннера\n" +
           exception.ToString(), "Баннер");
        this.Close();// закрываем форму
        return;
    }

    // определяем графическую поверхность
    g = this.CreateGraphics();

    // определяем область отображения баннера
    rct.X = 0;
    rct.Y = 0;
    rct.Width = baner.Width;
    rct.Height = baner.Height;

    // настройка таймера
    timer1.Interval = 50;
```

```
        timer1.Enabled = true;
    }

    // сигнал от таймера
    private void timer1_Tick(object sender, EventArgs e)
    {
        // сместить область отображения
        // баннера влево
        rct.X -= 1;

        // если область отображения целиком
        // "уехала" за левую границу формы,
        // вернем ее обратно
        if (Math.Abs(rct.X) > rct.Width)
            rct.X += rct.Width;

        // т.к. область отображения "едет" влево, то после
        // вывода в "съехавшую" область выведем баннер
        // еще раз (как минимум один, если ширина формы
        // равна ширине баннера)
        for (int i = 0; i <= Convert.ToInt16(
             this.ClientSize.Width / rct.Width) + 1; i++)
            g.DrawImage(baner,
                        rct.X + i * rct.Width, rct.Y);
    }

    // перемещение указателя мыши
    private void Form1_MouseMove(object sender,
                                 MouseEventArgs e)
    {
        // при наведении курсора мыши на баннер
        // движение останавливается
        if ((e.Y < rct.Y + rct.Height) && (e.Y > rct.Y))
```

```
            {
                if (timer1.Enabled != false)
                    timer1.Enabled = false;
            }
            else
            {
                if (timer1.Enabled != true)
                    timer1.Enabled = true;
            }
        }
    }
}
```

Полет

Программа **Полет** (рис. 1.27, листинг 1.25) демонстрирует принцип реализации анимации.

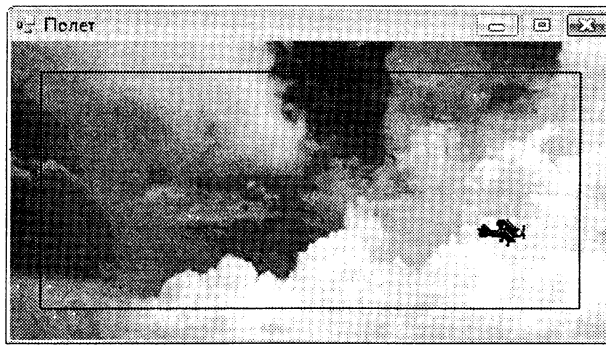


Рис. 1.27. Окно программы **Полет**

Кадры анимации формируются путем вывода в нужную область окна изображения объекта. Фоновый рисунок и изображение объекта (рис. 1.28) загружаются из ресурса.

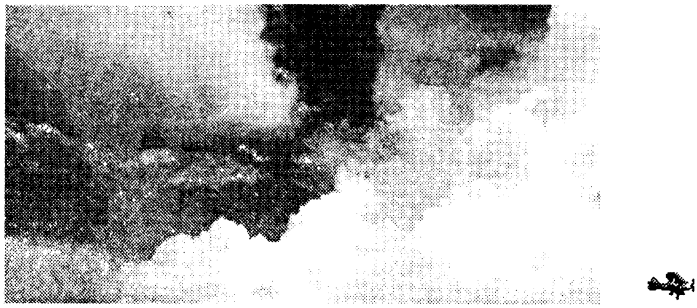


Рис. 1.28. Фоновый рисунок и объект

Листинг 1.25. Модуль формы программы *Полет*

```
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
    {
        // битовые образы: небо и самолет
        System.Drawing.Bitmap sky, plane;

        // рабочая поверхность
        Graphics g;

        // приращение координаты X
        // определяет скорость полета
        int dx;
```

```
// область, в которой находится самолет
Rectangle rct;

// true - самолет скрывается в облаках
Boolean demo = true;

// генератор случайных чисел
System.Random rnd;

public Form1()
{
    InitializeComponent();

    try
    {
     /*
     // Вариант 1: загрузка битовых образов из файлов
     sky = new Bitmap("sky.bmp");      // небо
     plane = new Bitmap("plane.bmp"); // самолет

     // загрузить и задать фоновый рисунок формы
     this.BackgroundImage = new Bitmap("sky.bmp");
    */

     // Вариант 2: загрузка битовых образов из ресурса
     sky = Properties.Resources.sky;      // фон
     plane = Properties.Resources.plane; // самолет

     // загрузить и задать фоновый рисунок формы
     this.BackgroundImage = Properties.Resources.sky;
    }
    catch (Exception exception)
    {
        MessageBox.Show(exception.ToString(),
```

```
            "Полет",  MessageBoxButtons.OK,
            MessageBoxIcon.Error);

        this.Close(); // закрыть приложение
        return;
    }

    // сделать прозрачным фон вокруг объекта
    plane.MakeTransparent();

    // задать размер формы в соответствии
    // с размером фонового рисунка
    this.ClientSize =
        new System.Drawing.Size(
            new Point(BackgroundImage.Width,
            BackgroundImage.Height));

    // g - графическая поверхность, на которой
    // будем формировать рисунок
    g = Graphics.FromImage(BackgroundImage);

    // инициализация генератора случ. чисел
    rnd = new System.Random();

    // исходное положение самолета
    rct.X = -40;
    rct.Y = 20 + rnd.Next(20);

    rct.Width = plane.Width;
    rct.Height = plane.Height;

    /*
    скорость полета определяется периодом следования
    сигналов от таймера и величиной
```

```
        приращения координаты X
      */
      dx = 2; // скорость полета - 2 пиксела/
                  тик_таймера

      timer2.Interval = 20;
      timer2.Enabled = true;
  }

  private void timer2_Tick(object sender, EventArgs e)
  {
      // стираем изображение самолета путем копирования
      // области фона на рабочую поверхность
      g.DrawImage(sky,new Point(0,0));

      // изменяем положение самолета
      if (rct.X < this.ClientRectangle.Width)
          rct.X += dx;
      else {
          // если достигли границы, задаем заново
          // положение самолета
          rct.X = -40;
          rct.Y = 20 + rnd.Next(40);

          // скорость полета от 2 до 5
          // пикселов/тик_таймера
          dx = 2 + rnd.Next(4);
      }

      // рисуем самолет на рабочей поверхности
      // (фактически на поверхности формы),
      // но для того, чтобы изменения появились,
      // надо инициировать обновление формы
```

```
            g.DrawImage(plane, rct.X, rct.Y);
            /*
            Метод Refresh инициирует перерисовку всей формы
            Метод Invalidate позволяет инициировать
            перерисовку только той области
            формы, которая указана в качестве
            параметра метода.
            */

            if ( ! demo )
                // обновить область формы,
                // в которой находится объект
                this.Invalidate(rct);
            else
            {
                // если объект находится вне области,
                // указанной в качестве параметра метода
                // Invalidate, то он не будет виден

                Rectangle reg =
                     new Rectangle(20,20,
                           sky.Width - 40, sky.Height - 40);

                // показать обновляемую область
                g.DrawRectangle(Pens.Black,
                    reg.X ,reg.Y ,reg.Width-1, reg.Height-1);

                this.Invalidate(reg); // обновить область
            }
        }
    }
}
```

Базы данных

В Microsoft Visual C# есть компоненты, позволяющие создавать программы для работы с базами данных Microsoft Access и Mictosoft SQL Server. Также есть возможность создать приложение для работы с базой данных Microsoft SQL Server Compact Edition.

Общие замечания

- Соединение с базой данных обеспечивает компонент `oleDbConnection`.
- Взаимодействие с базой данных, после того, как соединение установлено, осуществляет компонент `oleDbDataAdapter`.
- Хранение информации, полученной из базы данных, обеспечивает компонент `dataSet`.
- Для отображения информации, полученной из базы данных, а также выполнения операций редактирования, добавления и удаления предназначен компонент `dataGridView`.

Контакты

Программа **Контакты** (листинг 1.26) является примером приложения работы с базой данных Microsoft Access. Демонстрирует использование компонентов `dataSet`, `oleDbConnection`, `oleDbDataAdapter` и `dataGridView`. База данных **Контакты** (`contacts.mdb`) состоит из одной-единственной таблицы `contacts` (табл. 1.3). Форма и окно программы приведены на рис. 1.29 и 1.30, значения свойств компонентов — в табл. 1.4–1.7. Выполнять настройку компонентов и устанавливать значения их

свойств следует в той последовательности, в которой приведены таблицы и значения свойств в таблицах.

Рис. 1.29. Форма программы **Контакты**

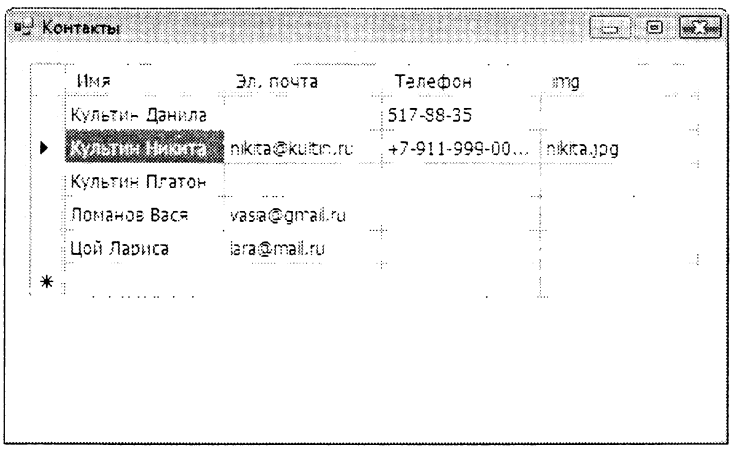


Рис. 1.30. Окно программы **Контакты**

Таблица 1.3. *Поля таблицы* `contacts`

Поле	Тип	Размер	Информация
`cid`	Целое, авто-увеличение	–	Идентификатор контакта
`name`	Текстовый	50	Имя
`phone`	Текстовый	50	Телефон
`email`	Текстовый	50	Эл. почта
`img`	Текстовый	50	Файл иллюстрации

Таблица 1.4. *Значения свойств компонента* `oleDbConnection1`

Свойство	Значение
`Name`	`oleDbConnection1`
`ConnectionString`	`Provider=Microsoft.Jet.OLEDB.4.0;` `Data Source=D:\Database\Contacts.mdb`

Таблица 1.5. *Значения свойств компонента* `oleDbDataAdapter1`

Свойство	Значение
`SelectCommand.Connection`	`oleDbConnection1`
`SelectCommand.CommandText`	`SELECT * FROM contacts`
`InsertCommand.Connection`	`oleDbConnection1`

`InsertCommand.CommandText`	`INSERT INTO contacts`
	`(name, phone, email, img)`
	`VALUES (?, ?, ?, ?)`

Таблица 1.5 *(продолжение)*

Свойство	Значение
InsertCommand.Parameters[0].ParameterName	Name
InsertCommand.Parameters[0].SourceColumn	name
InsertCommand.Parameters[1].ParameterName	phone
InsertCommand.Parameters[1].SourceColumn	phone
InsertCommand Parameters[2].ParameterName	email
InsertCommand.Parameters[2].SourceColumn	email
InsertCommand.Parameters[3].ParameterName	img
InsertCommand.Parameters[3].SourceColumn	img
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE contacts SET name = ?, phone = ?, email = ?, img = ? WHERE (cid = ?)
UpdateCommand.Parameters[0].ParameterName	name
UpdateCommand.Parameters[0].SourceColumn	name
UpdateCommand.Parameters[1].ParameterName	phone
UpdateCommand.Parameters[1].SourceColumn	phone

Таблица 1.5 *(окончание)*

Свойство	Значение
UpdateCommand.Parameters[2].ParameterName	Email
UpdateCommand.Parameters[2].SourceColumn	email
UpdateCommand.Parameters[3].ParameterName	img
UpdateCommand.Parameters[3].SourceColumn	img
UpdateCommand.Parameters[4].ParameterName	Original_cid
UpdateCommand.Parameters[4].SourceColumn	cid
UpdateCommand.Parameters[4].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1
DeleteCommand.CommandText	DELETE FROM contacts WHERE (cid = ?)
DeleteCommand.Parameters[0].ParameterName	cid
DeleteCommand.Parameters[0].SourceColumn	cid
DeleteCommand.Parameters[0].SourceVersion	Original

Таблица 1.6. *Значения свойств компонента* `dataSet1`

Свойство	Значение
Name	dataSet1
Tables[0].TableName	contacts

Таблица 1.6 *(окончание)*

Свойство	Значение
Tables[0].Columns[0].ColumnName	Name
Tables[0].Columns[0].Caption	name
Tables[0].Columns[1].ColumnName	phone
Tables[0].Columns[1].Caption	phone
Tables[0].Columns[2].ColumnName	email
Tables[0].Columns[2].Caption	email
Tables[0].Columns[3].ColumnName	img
Tables[0].Columns[3].Caption	img

Таблица 1.7. *Значения свойств компонента* `dataGridView1`

Свойство	Значение
DataSource	dataSet1
DataMember	contacts
AutoSizeColumnsMode	Fill
BorderStyle	None
Columns[0].DataPropertyName	cid
Columns[0].HeaderText	cid
Columns[0].Visible	False
Columns[1].DataPropertyName	name
Columns[1].HeaderText	Имя
Columns[2].DataPropertyName	email
Columns[2].HeaderText	Эл. почта

Таблица 1.7 *(окончание)*

Свойство	Значение
Columns[3].DataPropertyName	phone
Columns[3].HeaderText	Телефон
Columns[4].DataPropertyName	img
Columns[4].HeaderText	img

Листинг 1.26. Модуль формы программы *Контакты*

```
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;

using WindowsFormsApplication1.Properties;
/*
 Пространство имен WindowsFormsApplication1.Properties
 определено в файле Settings.Designer.cs,
 который создает среда разработки в результате
 формирования списка параметров на вкладке
 Settings (команда Project>Properties)

Параметры программы:
Параметр: ConnectionString
Name: ConnectionString
Type: String
```

```
Scope: Application
Value: Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=D:\Database\Contacts.md
*/

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // загрузка формы - начало работы программы
        private void Form1_Load(object sender, EventArgs e)
        {
            // загрузить строку соединения
            // из файла конфигурации
            oleDbConnection1.ConnectionString =
                    Settings.Default.ConnectionString;

            // прочитать данные из БД
            oleDbDataAdapter1.Fill(dataTable1);
        }

        // пользователь выделил строку и нажал <Delete>
        private void dataGridView1_UserDeletingRow
             (object sender, DataGridViewRowCancelEventArgs e)
        {
           DialogResult dr =
            MessageBox.Show("Удалить запись?",
                            "Удаление записи",
```

```
                                MessageBoxButtons.OKCancel,
                                MessageBoxIcon.Warning,
                                MessageBoxDefaultButton.Button2);
            if (dr == DialogResult.Cancel )
            {
                e.Cancel = true;
            }
        }

        // завершение работы программы
        private void Form1_FormClosing(object sender,
                                       FormClosingEventArgs e)
        {

oleDbDataAdapter1.Update(dataSet1.Tables["contacts"]);
        }
    }
}
```

Контакты-2

Программа **Контакты-2** (рис. 1.31, листинг 1.27) показывает технологию извлечения нужной информации из базы данных. Компоненты `oleDbConnection`, `oleDbDataAdapter`, `dataSet` и `dataGridView` настраиваются точно так же, как и в предыдущем примере (см. табл. 1.3–1.7), за исключением настройки свойства `SelectCommand` компонента `oleDbDataAdapter` (табл. 1.8).

Окно программы (результат выполнения запроса на поиск информации) приведено на рис. 1.32.

Рис. 1.31. Форма программы **Контакты-2**

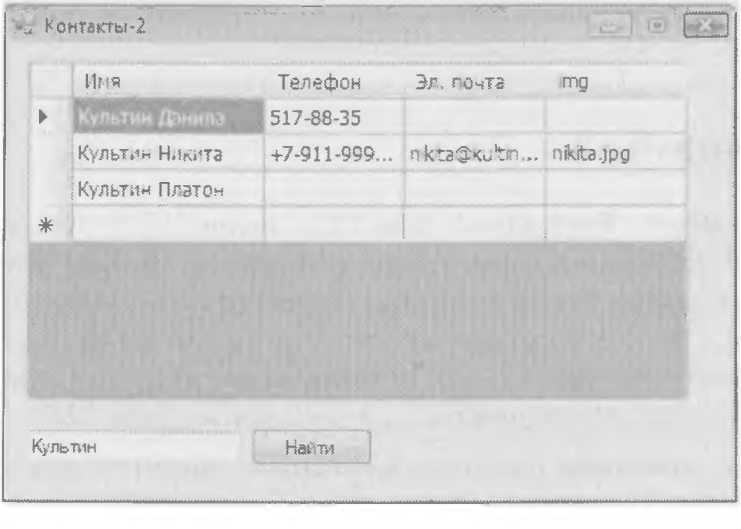


Рис. 1.32. Окно программы **Контакты-2**

Таблица 1.8. *Значения свойств объекта* `SelectCommand` *компонента* `oleDbDataAdapter1`

Свойство	Значение

`CommandText`	`SELECT *`
	`FROM contacts`
	`WHERE (name LIKE ?)`
	`ORDER BY name`

`Parameters[0].ParameterName`	`name`
`Parameters[0].SourceColumn`	`name`

Листинг 1.27. Модуль формы программы *Контакты-2*

```
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;

using WindowsFormsApplication1.Properties;
/*
 Пространство имен WindowsFormsApplication1.Properties
 определено в файле Settings.Designer.cs,
 который создает среда разработки в результате
 формирования списка параметров на вкладке
 Settings (команда Project>Properties)

Параметры программы:
Параметр: ConnectionString
```

```
Name: ConnectionString
Type: String
Scope: Application
Value: Provider=Microsoft.Jet.OLEDB.4.0;
       Data Source=D:\Database\Contacts.md
*/

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // загрузка формы - начало работы программы
        private void Form1_Load(object sender, EventArgs e)
        {
          // загрузить строку соединения
          // из файла конфигурации
          oleDbConnection1.ConnectionString =
                    Settings.Default.ConnectionString;

          // получить информацию из БД
          // т.к. у команды SELECT есть параметр,
          // который задает критерий отбора записей,
          // то надо задать его значение */
          oleDbDataAdapter1.SelectCommand.Parameters[0].Value
                                        = "%%";
          // прочитать данные из БД
          oleDbDataAdapter1.Fill(dataTable1);
        }
```

```
// щелчок на кнопке Найти
private void button1_Click(object sender, EventArgs e)
{

    dataSet1.Clear(); // удалить старые данные

    // для получения информации из базы данных
    // используется команда SELECT с параметром:
    // SELECT *FROM contacts WHERE (name Like ? ),
    // где: ? - параметр
    // В программе доступ к параметру можно получить
    // по номеру или по имени.

    oleDbDataAdapter1.
        SelectCommand.Parameters["name"].Value = "%" +
                                    textBox1.Text + "%";

    // выполнить команду
    oleDbDataAdapter1.Fill(dataTable1);
}

// пользователь выделил строку и нажал <Delete>
private void dataGridView1_UserDeletingRow
     (object sender, DataGridViewRowCancelEventArgs e)
{
   DialogResult dr =
    MessageBox.Show("Удалить запись?",
                    "Удаление записи",
                    MessageBoxButtons.OKCancel,
                    MessageBoxIcon.Warning,
                    MessageBoxDefaultButton.Button2);
    if (dr == DialogResult.Cancel )
    {
        e.Cancel = true;
```

```
            }
        }

        // завершение работы программы
        private void Form1_FormClosing(object sender,
                                       FormClosingEventArgs e)
        {

oleDbDataAdapter1.Update(dataSet1.Tables["contacts"]);
        }
    }
}
```

Контакты-3

Программа **Контакты-3** (рис. 1.33) демонстрирует отображение данных в режиме формы (данные записи, выбранной в таблице, отображаются в полях редактирования). Помимо текстовой информации в окне программы отображается иллюстрация, связанная с текущей записью (имя файла иллюстрации хранится в поле `img`). Необходимо обратить внимание на то, как формируется значение поля `img`: имя файла иллюстрации, выбранной пользователем во время работы программы в окне просмотра каталогов (окно появляется в результате щелчка кнопкой мыши в области отображения иллюстрации), записывается в поле `img`.

Форма программы **Контакты-3** приведена на рис. 1.34, текст — в листинге 1.28. Поля `textBox1`, `textBox2` и `textBox3` предназначены для отображения, соответственно, содержимого полей `name`, `phone` и `email` текущей записи. Следует обратить внимание на то, что за компонентом `pictureBox` находится компонент `textBox4`, предназначенный для хранения значения поля `img` текущей записи.

Значения свойств компонентов приведены в табл. 1.9–1.14.

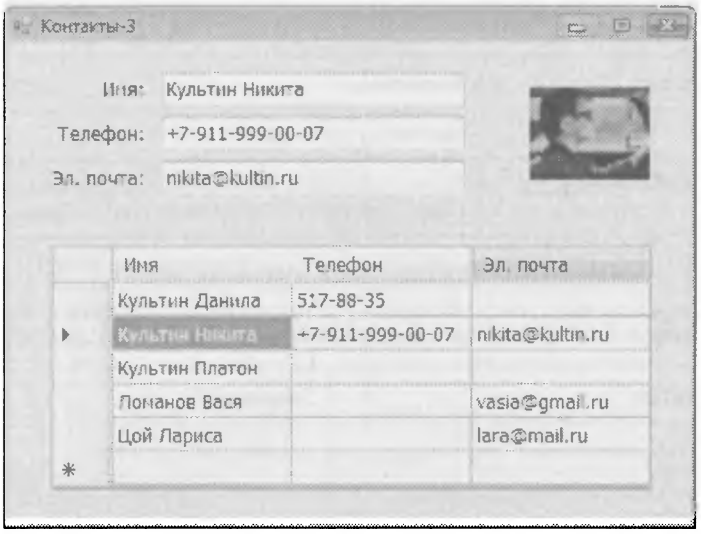


Рис. 1.33. Окно программы **Контакты-3**

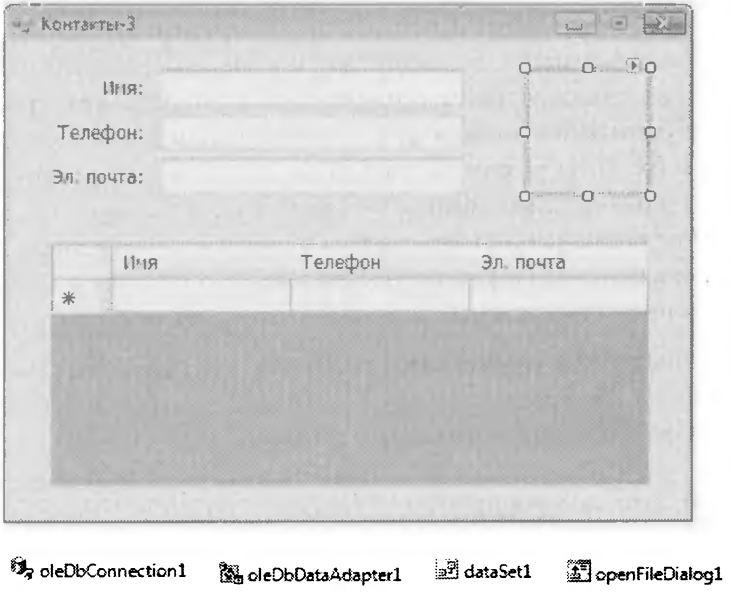


Рис. 1.34. Форма программы **Контакты-3**

Таблица 1.9. *Значения свойств компонента* `oleDbConnection1`

Свойство	Значение
Name	oleDbConnection1
ConnectionString	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=D:\Database\Contacts.mdb

Таблица 1.10. *Значения свойств компонента* `oleDbDataAdapter1`

Свойство	Значение
SelectCommand.Connection	oleDbConnection1
SelectCommand.CommandText	SELECT * FROM contacts
InsertCommand.Connection	oleDbConnection1
InsertCommand.CommandText	INSERT INTO contacts (name, phone, email, img) VALUES (?, ?, ?, ?)
InsertCommand.Parameters[0]. ParameterName	name
InsertCommand.Parameters[0]. SourceColumn	name
InsertCommand.Parameters[1]. ParameterName	phone
InsertCommand.Parameters[1]. SourceColumn	phone
InsertCommand Parameters[2]. ParameterName	email
InsertCommand.Parameters[2]. SourceColumn	email
InsertCommand.Parameters[3]. ParameterName	img

Таблица 1.10 *(продолжение)*

Свойство	Значение
InsertCommand.Parameters[3].SourceColumn	Img
UpdateCommand.Connection	oleDbConnection1
UpdateCommand.CommandText	UPDATE contacts SET name = ?, phone = ?, email = ?, img = ? WHERE (cid = ?)
UpdateCommand.Parameters[0].ParameterName	name
UpdateCommand.Parameters[0].SourceColumn	name
UpdateCommand.Parameters[1].ParameterName	phone
UpdateCommand.Parameters[1].SourceColumn	phone
UpdateCommand.Parameters[2].ParameterName	email
UpdateCommand.Parameters[2].SourceColumn	email
UpdateCommand.Parameters[3].ParameterName	img
UpdateCommand.Parameters[3].SourceColumn	img
UpdateCommand.Parameters[4].ParameterName	Original_cid
UpdateCommand.Parameters[4].SourceColumn	cid

Таблица 1.10 *(окончание)*

Свойство	Значение
UpdateCommand.Parameters[4].SourceVersion	Original
DeleteCommand.Connection	oleDbConnection1
DeleteCommand.CommandText	DELETE FROM contacts WHERE (cid = ?)
DeleteCommand.Parameters[0].ParameterName	cid
DeleteCommand.Parameters[0].SourceColumn	cid
DeleteCommand.Parameters[0].SourceVersion	Original

Таблица 1.11. *Значения свойств компонента* `dataSet1`

Свойство	Значение
Name	dataSet1
Tables[0].TableName	contacts
Tables[0].Columns[0].ColumnName	name
Tables[0].Columns[0].Caption	name
Tables[0].Columns[1].ColumnName	phpone
Tables[0].Columns[1].Caption	phone
Tables[0].Columns[2].ColumnName	email
Tables[0].Columns[2].Caption	email
Tables[0].Columns[3].ColumnName	img
Tables[0].Columns[3].Caption	img

Таблица 1.12. *Значения свойств компонента* `dataGridView1`

Свойство	Значение
DataSource	dataSet1
DataMember	contacts
AutoSizeColumnsMode	Fill
BorderStyle	None
Columns[0].DataPropertyName	cid
Columns[0].HeaderText	cid
Columns[0].Visible	False
Columns[1].DataPropertyName	name
Columns[1].HeaderText	Имя
Columns[2].DataPropertyName	email
Columns[2].HeaderText	Эл. почта
Columns[3].DataPropertyName	phone
Columns[3].HeaderText	Телефон

Таблица 1.13. *Значения свойств компонентов* `textBox`

Компонент	Свойство	Значение
textBox1	dataBindings.Text	dataSet1 - contacts.name
textBox2	dataBindings.Text	dataSet1 - contacts.phone
textBox3	dataBindings.Text	dataSet1 - contacts.email
textBox4	dataBindings.Text	dataSet1 - contacts.img
textBox4	Text	nodody

Таблица 1.14. *Значения свойств компонента* `pictureBox`

Свойство	Значение
`SizeMode`	`Zoom`
`Tool Tip on toolTip1`	Щелкните, чтобы задать или изменить иллюстрацию

Листинг 1.28. Модуль формы программы *Контакты-3*

```
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;

using System.IO; // для доступа к DirectoryInfo

using WindowsFormsApplication1.Properties;

namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }

    string dbPath;    // папка приложение
    string imFolder; // папка иллюстраций
```

```
// начало работы программы
private void Form1_Load(object sender, EventArgs e)
{
    // загрузить путь к файлу БД  из файла конфигурации
    dbPath = Settings.Default.DbPath;

    // открыть соединение с БД
    oleDbConnection1.ConnectionString =
       "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
       dbPath + "\\Contacts.mdb";

    // папка иллюстраций
    imFolder =  dbPath + "\\Images\\";

    // получить информацию из БД
    oleDbDataAdapter1.Fill(dataSet1.Tables[0]);
}

// пользователь нажал клавишу <Delete>
private void dataGridView1_UserDeletingRow
       (object sender, DataGridViewRowCancelEventArgs e)
{
    DialogResult r;

    r= MessageBox.Show
           ("Вы действительно хотите удалить запись?",
            "Удаление записи",
            MessageBoxButtons.OKCancel,
            MessageBoxIcon.Exclamation,
            MessageBoxDefaultButton.Button2);

    if (r == DialogResult.Cancel)
    {
        // отменить операцию удаления записи
        e.Cancel = true;
```

```
    }
}

// изменилось содержимое поля textBox4 -
// отобразить иллюстрацию, имя файла которой
// находится в этом поле
private void textBox4_TextChanged
                    (object sender, EventArgs e)
{
  string imageFile;
  string msg; // сообщение об ошибке

  if (textBox4.Text.Length == 0)
  {
      imageFile = imFolder + "nobody.jpg";
  }
  else
      imageFile = imFolder + textBox4.Text;

  try
  {
      msg = "";
      pictureBox1.Image =
           System.Drawing.Bitmap.FromFile(imageFile);
  }
  catch (System.IO.FileNotFoundException)
  {
      // вывести сообщение об ошибке в поле
      // компонента pictureBox1
      msg = "File nof found: " + imageFile;
      pictureBox1.Image = null;
      pictureBox1.Refresh();
  }
}
```

```
// щелчок в поле отображения иллюстрации
private void pictureBox1_Click
                    (object sender, EventArgs e)
{
  openFileDialog1.Title = "Выберите иллюстрацию";
  openFileDialog1.InitialDirectory = imFolder;
  openFileDialog1.Filter = "фото|*.jpg|все файлы|*.*";
  openFileDialog1.FileName = "";

  if (openFileDialog1.ShowDialog() == DialogResult.OK)
  {
      // пользователь указал файл иллюстрации

      // проверим, находится ли выбранный файл
      // в каталоге imFolder
      bool r =
        openFileDialog1.FileName.ToLower().Contains(
          openFileDialog1.InitialDirectory.ToLower());
      if (r == true)
      {
          // копировать не надо, т.к. пользователь
          // указал иллюстрацию, которая
          // находится в imFolder
          textBox4.Text =
                   openFileDialog1.SafeFileName;
      }
      else
      {
        // Скопировать файл иллюстрации в папку Images
        // Если в каталоге-приемнике есть файл
        // с таким же именем, что и копируемый,
        // возникает исключение
        try
        {
```

```
                // копировать файл
                System.IO.File.Copy(openFileDialog1.FileName,
                      imFolder +
penFileDialog1.SafeFileName);

                textBox4.Text = openFileDialog1.SafeFileName;
            }
            catch (Exception ex)
            {
                DialogResult dr;
                dr = MessageBox.Show(ex.Message +
                        " Заменить его?", "",
                        MessageBoxButtons.OKCancel,
                        MessageBoxIcon.Warning,
                        MessageBoxDefaultButton.Button2);
                 if (dr == DialogResult.OK)
                 {
                   // перезаписать файл
                  System.IO.File.Copy(
                     openFileDialog1.FileName,
                     imFolder +
                     openFileDialog1.SafeFileName,
                     true);
                   textBox4.Text =
                             openFileDialog1.SafeFileName;
                   }
                }
            }
        }
    }

    // завершение работы программы
    private void Form1_FormClosing
           (object sender, FormClosingEventArgs e)
    {
```

```
        try
        {
            oleDbDataAdapter1.Update(dataTable1);
        }
        catch (Exception e1)
        {
            MessageBox.Show(e1.ToString());
        }
    }
  }
}
```

SQL Server Compact Edition

Следующая программа демонстрирует работу с базой данных Microsoft SQL Server Compact Edition.

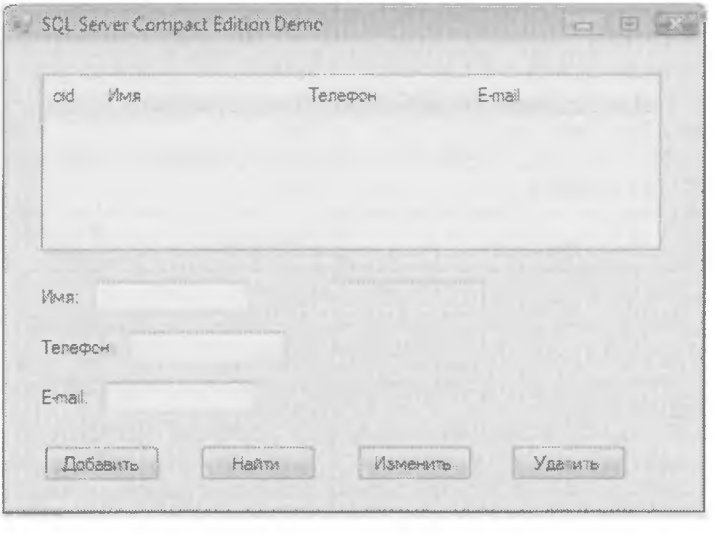


Рис. 1.35. Форма программы работы с базой данных Microsoft SQL Server Compact Edition

Программа позволяет просматривать базу данных **Контакты** (contacts.sdf), а также вносить в нее изменения (добавлять, редактировать и удалять записи). Форма программы приведена на рис. 1.35, текст — в листинге 1.29. Для отображения записей в табличной форме используется компонент `listView` (значения его свойств приведены в табл. 1.15). Компоненты `textBox1`, `textBox2` и `textBox3` предназначены для ввода и редактирования полей `name`, `phone` и `emal`, а компонент `textBox4` (значение свойства `ReadOnly` которого равно `True`) — для хранения значения поля `cid`. Нетрудно заметить, что на форме нет компонентов для доступа к базе данных. Все объекты, обеспечивающие работу с базой данных, создаются во время работы программы.

Таблица 1.15. *Значения свойств компонента* `listView`

Свойство	Значение
`Columns[0].Text`	`cid`
`Columns[0].Width`	`35`
`Columns[1].Text`	`Имя`
`Columns[1].Width`	`130`
`Columns[2].Text`	`Телефон`
`Columns[2].Width`	`110`
`Columns[3].Text`	`E-mail`
`Columns[3].Width`	`110`
`HideSelection`	`False`
`MultiSelect`	`False`
`View`	`Detals`

Листинг 1.29. Модуль формы программы для работы с SQL Server Compact Edition

```
/*
   Программа демонстрирует выполнение основных действий
   с базой данных Microsoft SQL Server Compact Edition.
   Показывает как:
   - создать базу данных;
   - создать таблицу;
   - добавить, получить, изменить и удалить информацию.

   (c) Культин Н.Б., 2009
   http://kultin.ru
*/
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;

using System.IO;

/*
    Чтобы пространство имен стало доступно,
    надо добавить ссылку на файл сборки, в котором
    оно определено. Для этого надо в меню Project
    выбрать команду Add Reference и указать файл сборки.
*/

using System.Data.SqlServerCe;
```

```
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();

        // настройка компонента ListView:
        // увеличим ширину компонента на 17 - ширину
        // полосы прокрутки
        int w = 0;
        for (int i = 0; i < listView1.Columns.Count; i++)
        {
            w += listView1.Columns[i].Width;
        }

        if (listView1.BorderStyle == BorderStyle.Fixed3D)
            w += 4;

        listView1.Width =
                    w+17; // 17 - ширина полосы прокрутки

        // выделять всю строку (элемент и подэлементы)
        listView1.FullRowSelect = true;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        // Создать БД SQL Server Compact Edition.
        // Если путь к файлу не указан, БД будет создана
        // в каталоге приложения
        SqlCeEngine engine;
```

```
    engine =
       new SqlCeEngine("Data Source='contacts.sdf';");
    if (!(File.Exists("contacts.sdf")))
    {
        engine.CreateDatabase();
        SqlCeConnection connection =
        new
        SqlCeConnection(engine.LocalConnectionString);
        connection.Open();
        SqlCeCommand command =
        connection.CreateCommand();
        command.CommandText =
            "CREATE TABLE contacts (cid int
             IDENTITY(1,1),
             name nvarchar(50) NOT NULL,
             phone nvarchar(50), email nvarchar(50))";
        command.ExecuteScalar();
        connection.Close();
    }
    else
    {
        ShowDB();
    }
}

private void ShowDB()
{
    SqlCeEngine engine =
       new SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection connection =
       new SqlCeConnection(engine.LocalConnectionString);
    connection.Open();
    SqlCeCommand command = connection.CreateCommand();
    command.CommandText =
         "SELECT * FROM contacts ORDER BY name";
    SqlCeDataReader dataReader = command.ExecuteReader();
```

```
        string st;  // значение поля БД
        int itemIndex = 0;

        listView1.Items.Clear();

        while (dataReader.Read())
        {
          for (int i = 0; i < dataReader.FieldCount; i++)
          {
              st = dataReader.GetValue(i).ToString();
              switch (i)
              {
                case 0:  // поле cid
                    listView1.Items.Add(st);
                    break;
                case 1:  // поле name
listView1.Items[itemIndex].SubItems.Add(st);
                  break;
                case 2:  // поле phone
listView1.Items[itemIndex].SubItems.Add(st);
                  break;
                case 3:  // поле email
listView1.Items[itemIndex].SubItems.Add(st);
                  break;
              };
            }
            itemIndex++;
        }
        connection.Close();
    }
```

```
// пользователь выбрал строку в поле компонента listView
private void listView1_ItemSelectionChanged(object
sender, ListViewItemSelectionChangedEventArgs e)
{
  /* При выборе строки событие ItemSelectionChanged
     возникает два раза: первый раз, когда выделенная в
     данный момент строка теряет фокус, второй - когда
     строка, в которой сделан щелчок, получает фокус.
     Нас интересует строка, которая получает фокус.
  */
    if (e.IsSelected)
    {
      // строка выбрана, т.е. она получила фокус
      textBox4.Text = listView1.Items[e.ItemIndex].Text;

      for (int i = 1;
         i < listView1.Items[e.ItemIndex].SubItems.Count;
                  i++)
      {
          switch (i)
          {
            case 1:
                textBox1.Text =
           listView1.Items[e.ItemIndex].SubItems[i].Text;
                break;
            case 2:
                textBox2.Text =
                listView1.Items[e.ItemIndex].
                SubItems[i].Text;
                break;
            case 3:
                textBox3.Text =
                listView1.Items[e.ItemIndex].
                SubItems[i].Text;
                break;
```

```
            }
        }
    }
}

// щелчок на кнопке Добавить
private void button1_Click(object sender, EventArgs e)
{
    SqlCeConnection conn =
       new SqlCeConnection("Data Source
       ='contacts.sdf'");
    conn.Open();
    SqlCeCommand command = conn.CreateCommand();
    command.CommandText =
  "INSERT INTO contacts(name, phone,email)
   VALUES(?,?,?)";
    command.Parameters.Add("name", textBox1.Text);
    command.Parameters.Add("phone", textBox2.Text);
    command.Parameters.Add("email", textBox3.Text);
    command.ExecuteScalar();
    conn.Close();

    // очистить поля ввода
    textBox1.Clear();
    textBox2.Clear();
    textBox3.Clear();

    ShowDB();

    // установить курсор в поле textBox1
    textBox1.Focus();
}

// щелчок на кнопке Найти
private void button2_Click(object sender, EventArgs e)
```

```
{
    SqlCeEngine engine =
       new SqlCeEngine("Data Source='contacts.sdf';");
    SqlCeConnection connection =
       new SqlCeConnection(engine.LocalConnectionString);
    connection.Open();

    SqlCeCommand command = connection.CreateCommand();

    command.CommandText =
         "SELECT * FROM contacts WHERE (name LIKE ?)";
    command.Parameters.Add("name",
                           "%" + textBox1.Text + "%");

    SqlCeDataReader dataReader = command.ExecuteReader();

    string st;  // значение поля БД
    int itemIndex = 0;

    listView1.Items.Clear();

    while (dataReader.Read())
    {
        for (int i = 0; i < dataReader.FieldCount; i++)
        {
          st = dataReader.GetValue(i).ToString();
          switch (i)
          {
            case 0:  // поле cid
                listView1.Items.Add(st);
                break;
            case 1:  // поле name
```

```
listView1.Items[itemIndex].SubItems.Add(st);
                    break;
                case 2:  // поле phone
listView1.Items[itemIndex].SubItems.Add(st);
                    break;
                case 3:  // поле email
listView1.Items[itemIndex].SubItems.Add(st);
                    break;
              };
            }
            itemIndex++;
        }
        connection.Close();

    }

    // щелчок на кнопке Удалить
    private void button3_Click(object sender, EventArgs e)
    {
        if (listView1.SelectedItems.Count != 0)
        {
          SqlCeEngine engine =
               new SqlCeEngine("Data
               Source='contacts.sdf';");
          SqlCeConnection connection =
            new
            SqlCeConnection(engine.LocalConnectionString);
          connection.Open();

          SqlCeCommand command = connection.CreateCommand();

          command.CommandText =
                  "DELETE FROM contacts WHERE (cid = ?)";
          command.Parameters.Add("cid", textBox4.Text);
```

```
        command.ExecuteScalar(); // выполнить команду

        ShowDB();

        textBox1.Clear();
        textBox2.Clear();
        textBox3.Clear();
        textBox4.Clear();
    }
}

// щелчок на кнопке Изменить
private void button4_Click(object sender, EventArgs e)
{
    if (listView1.SelectedItems.Count != 0)
    {
        SqlCeEngine engine =
          new SqlCeEngine("Data Source='contacts.sdf';");
        SqlCeConnection connection = new
          SqlCeConnection(engine.LocalConnectionString);
        connection.Open();

        SqlCeCommand command =
        connection.CreateCommand();

        command.CommandText =
           "UPDATE contacts" +
           "SET name = ?, phone =?, email=? " + "
           "WHERE cid = ?";
        command.Parameters.Add("name", textBox1.Text);
        command.Parameters.Add("phone", textBox2.Text);
        command.Parameters.Add("email", textBox3.Text);
        command.Parameters.Add("cid", textBox4.Text);

        command.ExecuteScalar(); // выполнить команду
```

```
            ShowDB();

            textBox1.Clear();
            textBox2.Clear();
            textBox3.Clear();
            textBox4.Clear();
        }
    }

    // изменился текст в поле редактирования
    private void textBox1_TextChanged
                (object sender, EventArgs e)
    {
        // кнопка Добавить становится доступной,
        //  если информация введена в поле Имя и в
        //  какое-либо из полей Телефон или E-mail
        if ((textBox1.TextLength > 0) &&
            ((textBox2.TextLength > 0) ||
                  (textBox3.TextLength > 0)))
            button1.Enabled = true;
        else
        {
            button1.Enabled = false;
        }
    }
  }
}
```

Игры и другие полезные программы

Парные картинки

Игра **Парные картинки** (рис. 1.36) развивает память. Правила игры следующие: игровое поле разделено на клетки, за каждой из

которых скрыта картинка. Картинки парные, т. е. на поле есть две клетки с одинаковыми картинками. Задача игрока — найти все пары картинок. В начале игры все клетки закрыты. Щелчок мыши в клетке открывает первую картинку, щелчок в другой клетке — вторую. Если картинки в открытых клетках одинаковые, считается, что пара найдена, и клетки исчезают. Если картинки разные, то они остаются открытыми.

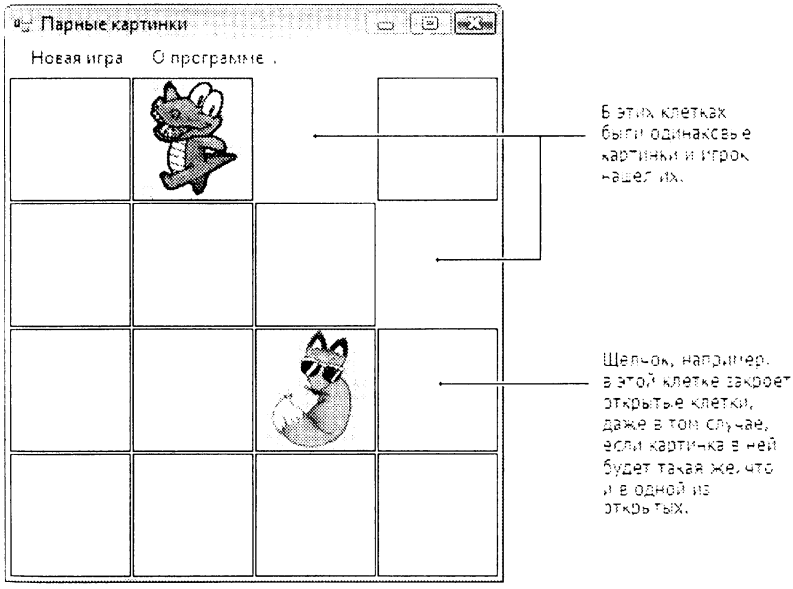


Рис. 1.36. Окно игры **Парные картинки**

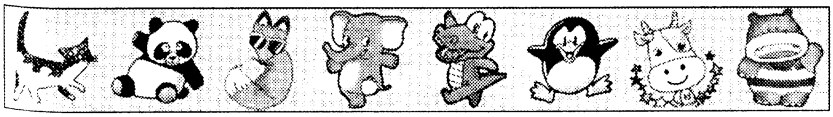


Рис. 1.37. Содержимое файла картинок

Следующий щелчок открывает клетку, в которой он сделан, и закрывает открытые, причем даже в том случае, если картинка в

ней такая же, что и одна из открытых. Игра заканчивается, когда будут открыты все пары картинок. Текст программы приведен в листинге 1.30. Картинки загружаются из файла (рис. 1.37).

Листинг 1.30. Модуль главной формы программы *Парные картинки*

```
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
  {
    const int nw = 4; // кол-во клеток по горизонтали
    const int nh = 4; // кол-во клеток по вертикали

    const int np = (nw * nh) / 2;  // кол-во пар картинок

    // рабочая графическая поверхность
    System.Drawing.Graphics g;

    // картинки (загружаются из файла)
    Bitmap pics;

    // размер (ширина и высота) клетки (картинки)
    int cw, ch;
```

```
// игровое поле:
int[,] field = new int[nw, nh];

// field[i,j] == 1 ... k - клетка закрыта
//                         (k-номер картинки);
// field[i,j] == 101 ... (100+k) - клетка открыта
//                       (игрок видит картинку);
// field[i,j] == 201 ... (200+k) - в клетке картинка,
//                        для которой найдена пара

// кол-во открытых (найденных) пар картинок
int nOpened = 0;

// количество открытых в данный момент клеток
int cOpened = 0;

// координаты 1-й открытой клетки
int[] open1 = new int[2];

// координаты 2-й открытой клетки
int[] open2 = new int[2];

// таймер
System.Windows.Forms.Timer timer1;

// нарисовать клетку:
// - картинку, если клетка открыта;
// - границу, если клетка закрыта;
// - фон, если в клетке картинка, для которой найдена пара
private void cell(int i, int j)
{
    int x,y;  // координаты левого верхнего угла клетки
```

```
        // между картинками
        // оставляем промежутки в 1 пиксел
        x = i * (cw + 2);
        y = j * (ch + 2) + menuStrip1.Height;

        if (field[i, j] > 200)
            // для этой клетки найдена пара,
            // картинку отображать не надо
            g.FillRectangle(SystemBrushes.Control,
                x, y, cw + 2, ch + 2);

        if ((field[i, j] > 100) && (field[i, j] < 200))
        {
            // клетка открыта - отобразить картинку
            g.DrawImage(pics,
              new Rectangle(x + 1, y + 1, cw, ch),
              new Rectangle((field[i, j] - 101) * cw, 0,
              cw, ch),
              GraphicsUnit.Pixel);
            g.DrawRectangle(Pens.Black,
                x + 1, y + 1, cw, ch);
        }

        if ((field[i, j] > 0) && (field[i, j] < 100))
        {
            // клетка закрыта, рисуем контур
            g.FillRectangle(SystemBrushes.Control,
                x + 1, y + 1, cw, ch);
            g.DrawRectangle(Pens.Black,
                x + 1, y + 1, cw, ch);
        }
    }
```

```
// нарисовать поле
private void drawField()
{
    for (int i = 0; i < nw; i++)
        for (int j = 0; j < nh; j++)
            this.cell(i, j);
}

// новая игра
private void newGame()
{
    // Раскидаем пары картинок по игровому полю:
    // запишем в массив field случайные числа
    // от 1 до k, где k - количество картинок.
    // Каждое число должно быть записано
    // в массив два раза.

    Random rnd; // генератор случайных чисел
    int rndN;   // случайное число

    rnd = new Random();

    int[] buf = new int[np];
    // np - кол-во картинок;
    // в buf[i] записываем, сколько i чисел
    // (идентификаторов картинок) записали в массив field

    for (int i = 0; i < nw; i++)
        for (int j = 0; j < nh; j++)
        {
            do
            {
                rndN = rnd.Next(np) + 1;
            } while (buf[rndN - 1] == 2);
```

```
                field[i, j] = rndN;
                buf[rndN - 1]++;
            }

        nOpened = 0;
        cOpened = 0;

        this.drawField();
    }

    public Form1()
    {
        InitializeComponent();

        try
        {
            // загружаем файл с картинками
            pics = new Bitmap("pictures.bmp");
        }
        catch (Exception exc)
        {
          MessageBox.Show("Файл 'pictures.bmp' не найден.\n"

              exc.ToString(), "Парные картинки",
              MessageBoxButtons.OK,
              MessageBoxIcon.Error);
          this.Close();
          return;
        }

        // определяем размер картинки и устанавливаем
        // размер клеток игрового поля
        cw = (int)(pics.Width / np);
        ch = pics.Height;
```

```
        // установить размер клиентской области формы
        // в соответствии с размером картинок и их количеством
        // (см. определения констант cw и ch)
        this.ClientSize =
            new System.Drawing.Size(nw * (cw + 2) + 1,
                    nh * (ch + 2) + 1 + menuStrip1.Height);

        // рабочая графическая поверхность
        g = this.CreateGraphics();

        // создать объект timer1
        timer1 = new Timer();
        timer1.Tick +=
            new System.EventHandler(this.timer1_Tick);
        timer1.Interval = 200;

        newGame();
    }

    // щелчок на игровом поле
    private void Form1_MouseClick(object sender,
                                  MouseEventArgs e)
    {
        {
            int i,j; // индексы элемента массива fild,
                     // соответствующего клетке, в которой
                     // сделан щелчок

            i = e.X / (cw +3);
            j = (e.Y - menuStrip1.Height) / (ch+3);

            // если таймер работает, это значит, что в данный
            // момент открыты две клетки, в которых находятся
            // одинаковые картинки, но они еще не "стерты".
```

```
// Если щелчок сделан в одной из этих картинок,
// то ничего делать не надо.
if ((timer1.Enabled) && (field[i,j] > 200))
{
    return;
}

// щелчок на месте одной из двух уже найденных
// парных картинок
if (field[i, j] > 200) return;

// открытых клеток нет
if (cOpened == 0)
{
    cOpened++;

    // записываем координаты 1-й открытой клетки
    open1[0] = i; open1[1] = j;

    // клетка помечается как открытая
    field[i, j] += 100;

    // отрисовать клетку
    this.cell(i, j);

    return;
}

// открыта одна клетка, надо открыть вторую
if (cOpened == 1)
{
    // записываем координаты 2-й открытой клетки
    open2[0] = i; open2[1] = j;
```

```
// если открыта одна клетка, и щелчок сделан
// в той же клетке, ничего не происходит
if ((open1[0] == open2[0]) &&
                        (open1[1] == open2[1]))
    return;
else
{
    // теперь открыты две клетки
    cOpened++;

    // клетка помечается как открытая
    field[i, j] += 100;

    // отрисовать клетку
    this.cell(i, j);

    // открыты 2 одинаковые картинки
    if (field[open1[0], open1[1]] ==
        field[open2[0], open2[1]])
    {
      nOpened++;

      // пометим клетки как найденные
      field[open1[0], open1[1]] += 100;
      field[open2[0], open2[1]] += 100;

      cOpened = 0;

      // Запускаем таймер. Процедура обработки
      // сигнала от таймера "сотрет" клетки,
      // клетки с одинаковыми картинками
      timer1.Enabled = true;
    }
}
```

```
            return;
        }

        // открыты 2 клетки но в них разные картинки,
        // закроем их и откроем ту клетку, в которой
        // сделан щелчок
        if (cOpened == 2)
        {
            // закрываем открытые клетки
            field[open1[0], open1[1]] -= 100;
            field[open2[0], open2[1]] -= 100;

            this.cell(open1[0], open1[1]);
            this.cell(open2[0], open2[1]);

            // записываем в open1 номер текущей клетки
            open1[0] = i; open1[1] = j;

            cOpened = 1;        // счетчик открытых клеток

            // открыть клетку, в которой сделан щелчок
            field[i, j] += 100;
            this.cell(i, j);
        }
    }
}

// команда Новая игра
private void новаяИграToolStripMenuItem_Click
                        (object sender, EventArgs e)
{
    newGame();
}
```

```
    // обработка события таймера
    private void timer1_Tick(object sender,
                             System.EventArgs e)
    {
        // отрисовать клетки
        this.cell(open1[0], open1[1]);
        this.cell(open2[0], open2[1]);

        // остановить таймер
        timer1.Enabled = false;

        if (nOpened == np)
        {
            MessageBox.Show
                    ("Вы справились с поставленной задачей!");
        }
    }

    private void Form1_Paint(object sender, PaintEventArgs e)
    {
        // отрисовать игровое поле
        drawField();
    }

    // команда О программе
    private void оПрограммеToolStripMenuItem_Click
                                (object sender, EventArgs e)
    {
        AboutBox1 aboutBox; // диалог О программе

        aboutBox = new AboutBox1();
        aboutBox.Show();
    }
}
```

Собери картинку

Программа **Собери картинку** (рис. 1.38) — графический вариант хорошо известной игры "15". Ее цель — расположить фишки (фрагменты картинки) в правильном порядке. Текст программы приведен в листинге 1.31.

Рис. 1.38. Игра **Собери картинку**

Листинг 1.31. Модуль главной формы программы *Собери картинку*

```
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
  {
    // 4x4 - размер игрового поля
    const int nw = 4, nh = 4;

    // графическая поверхность формы
    System.Drawing.Graphics g;

    // картинка
    Bitmap pics;

    // размер (ширина и высота) клетки
    int cw, ch;

    // игровое поле: хранит номера фрагментов
    // картинки
    int[,] field = new int[nw, nh];

    // координаты пустой клетки
    int ex, ey;

    // признак отображения номера фишки
    Boolean showNumbers = false;

    public Form1()
    {
        InitializeComponent();

        try
        {
```

```
            // загружаем файл картинки
            pics = new Bitmap("puzzle.bmp");
        }
        catch (Exception exc)
        {
            MessageBox.Show("Файл 'puzzle.bmp' не найден.\n",
                "Собери картинку",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            this.Close();
            return;
        }

        // определяем высоту и ширину клетки (фишки)
        cw = (int)(pics.Width / nw);
        ch = (int)(pics.Height / nh);

        // установить размер клиентской области формы
        this.ClientSize =
            new System.Drawing.Size(cw * nw + 1,
                                    ch * nh + 1 + menuStrip1.Height);

        // рабочая графическая поверхность
        g = this.CreateGraphics();

        this.newGame();
    }

    // новая игра
    private void newGame()
    {
        // располагаем фишки в правильном порядке
        for (int j = 0; j < nh; j++)
```

```
            for (int i = 0; i < nw; i++)
                field[i, j] = j * nw + i + 1;

        // последняя фишка - пустая
        field[nw - 1, nh - 1] = 0;
        ex = nw - 1; ey = nh - 1;

        this.mixer();          // перемешиваем фишки
        this.drawField();      // выводим игровое поле
    }

    // перемешивает фишки
    private void mixer()
    {
        int d; // положение (относительно пустой) перемещаемой
               // клетки: 0 - слева; 1 - справа;
               //         2 - сверху; 3 - снизу.

        int x, y; // перемещаемая клетка

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

        for (int i = 0; i < nw * nh * 10; i++)
        // nw * nh * 10 - кол-во перестановок
        {
            x = ex;
            y = ey;

            d = rnd.Next(4);
            switch (d)
            {
                case 0: if (x > 0) x--; break;
```

```
                case 1: if (x < nw - 1) x++; break;
                case 2: if (y > 0) y--; break;
                case 3: if (y < nh - 1) y++; break;
            }

            // здесь определили фишку, которую
            // нужно переместить в пустую клетку
            field[ex, ey] = field[x, y];
            field[x, y] = 0;

            // запоминаем координаты пустой фишки
            ex = x; ey = y;
        }
    }

    // отрисовывает поле
    private void drawField()
    {
        // содержимое клеток
        for (int i = 0; i < nw; i++)
            for (int j = 0; j < nh; j++)
            {
                if (field[i, j] != 0)
                    // выводим фишку с картинкой:
                    // ( ((field[i,j] - 1) % nw) * cw,
                    //   (int)((field[i,j] - 1) / nw) * ch ) -
                    // координаты левого верхнего угла
                    // области файла-источника картинки
                    g.DrawImage(pics,
                        new Rectangle(i * cw,
                                  j * ch + menuStrip1.Height,
                                   cw, ch),
                        new Rectangle(
                            ((field[i, j] - 1) % nw) * cw,
                            ((field[i, j] - 1) / nw) * ch,
```

```
                    cw, ch),
                GraphicsUnit.Pixel);
        else
            // выводим пустую фишку
            g.FillRectangle(SystemBrushes.Control,
                i * cw, j * ch + menuStrip1.Height,
                 cw, ch);

        // рисуем границу
        g.DrawRectangle(Pens.Black,
            i * cw, j * ch + menuStrip1.Height,
            cw, ch);

        // номер фишки
        if ((showNumbers) && field[i, j] != 0)
            g.DrawString(
                Convert.ToString(field[i, j]),
                new Font("Tahoma", 10,
                FontStyle.Bold),
                Brushes.Black, i * cw + 5,
                j * ch + 5 + menuStrip1.Height);
    }
}

// проверяет, расположены ли фишки в правильном порядке
private Boolean finish()
{
    // координаты клетки
    int i = 0;
    int j = 0;

    int c;        // число в клетке

    // фишки расположены правильно, если
    // числа в них образуют матрицу:
    //   1  2  3  4
    //   5  6  7  8
```

```
        //   9 10 11 12
        //  13 14 15

        for (c = 1; c < nw * nh; c++)
        {
            if (field[i, j] != c) return false;

            // к следующей клетке
            if (i < nw - 1) i++;
            else { i = 0; j++; }
        }
        return true;
    }

    // перемещает фишку, на которой сделан щелчок,
    // в соседнюю пустую клетку:
    // (cx, cy) - клетка, в которой сделан щелчок,
    // (ex, ey) - пустая клетка
    private void move(int cx, int cy)
    {
        // проверим, возможен ли обмен
        if (!(((Math.Abs(cx - ex) == 1) && (cy - ey == 0)) ||
            ((Math.Abs(cy - ey) == 1) && (cx - ex == 0))))
            return;

        // обмен. переместим фишку из (x, y) в (ex, ey)
        field[ex, ey] = field[cx, cy];
        field[cx, cy] = 0;

        ex = cx; ey = cy;

        // отрисовать поле
        this.drawField();
        if (this.finish())
        {
            field[nw - 1, nh - 1] = nh * nw;
            this.drawField();
```

```
            // игра закончена. сыграть еще раз?
            // No  - завершить работу программы,
            // Yes - новая игра
            if (MessageBox.Show(
                    "Вы справились с поставленной задачей!\n" +
                    "Еще раз?", "Собери картинку",
                    MessageBoxButtons.YesNo,
                    MessageBoxIcon.Question)
                       ==
                    System.Windows.Forms.DialogResult.No)
                this.Close();
            else this.newGame();
        }
    }

    // обработка события Paint
    private void Form1_Paint(object sender, PaintEventArgs e)
    {
           drawField();
    }

    // щелчок кнопкой мыши на игровом поле
    private void Form1_MouseClick(object sender,
                                  MouseEventArgs e)
    {
        // преобразуем координаты мыши в координаты клетки
        move( e.X/cw, (e.Y-menuStrip1.Height)/ch);
    }
    // команда Новая игра
    private void новаяИграToolStripMenuItem_Click
                    (object sender, EventArgs e)
    {
        newGame();
    }

    private void оПрограммеToolStripMenuItem_Click
                    (object sender, EventArgs e)
```

```
    {
        Form2 f = new Form2();
        f.ShowDialog();
    }
  }
}
```

Сапер

Программа **Сапер** (рис. 1.39) — аналог игры, хорошо знакомой всем пользователям Windows, — демонстрирует работу с графикой, массивами, вывод справочной информации, отображение окна **О программе**.

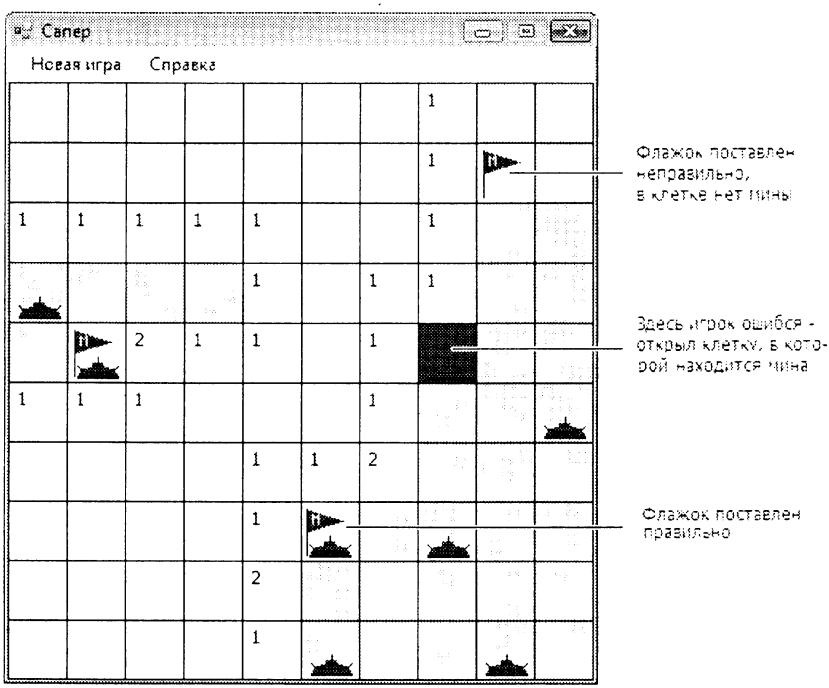


Рис. 1.39. Игра **Сапер**

Главная форма программы и форма **О программе** приведены на рис. 1.40 и 1.41, а их текст — в листингах 1.32 и 1.33 соответственно.

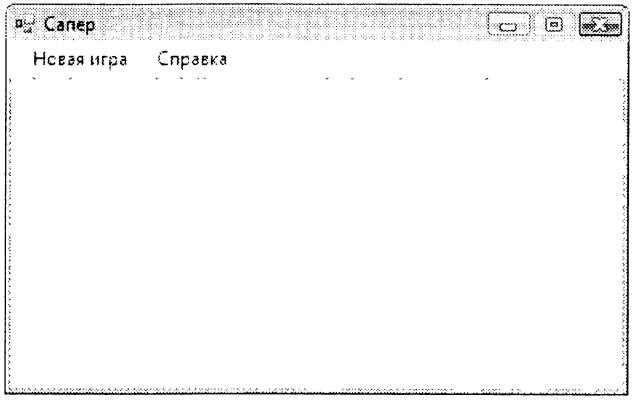


Рис. 1.40. Главная форма

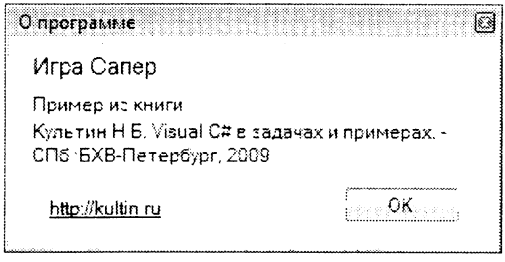


Рис. 1.41. Форма **О программе**

Листинг 1.32. Модуль главной формы программы *Сапер*

```
// модуль главной 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 Form1 : Form
  {

  private const int
    MR = 10, // кол-во клеток по вертикали
    MC = 10, // кол-во клеток по горизонтали
    NM = 10, // кол-во мин
    W = 40,  // ширина клетки
    H = 40;  // высота клетки

  // игровое (минное) поле
  private int[,] Pole = new int[MR + 2, MC + 2];
  // значение элемента массива:
  // 0..8 - количество мин в соседних клетках
  // 9 - в клетке мина
  // 100..109 - клетка открыта
  // 200..209 - в клетку поставлен флаг

  private int nMin;  // кол-во найденных мин
  private int nFlag; // кол-во поставленных флагов

  // статус игры
  private int status;
  // 0 - начало игры,
  // 1 - игра,
  // 2 - результат
```

```
// графическая поверхность формы
private System.Drawing.Graphics g;

public Form1()
{
    InitializeComponent();

    // В неотображаемые эл-ты массива, соответствующие
    // клеткам границы игрового поля, запишем число -3.
    // Это значение используется процедурой open()
    // для завершения рекурсивного процесса открытия
    // соседних пустых клеток
    for(int row = 0; row <= MR+1; row++)
    {
        Pole[row,0] = -3;
        Pole[row,MC+1] = -3;
    }

    for(int col = 0; col <= MC+1; col++)
    {
        Pole[0,col] = -3;
        Pole[MR+1,col] = -3;
    }

    // устанавливаем размер формы в соответствии
    // с размером игрового поля
    this.ClientSize = new Size(W*MC + 1,
                    H*MR +  menuStrip1.Height +  1);

    newGame(); // новая игра

    // графическая поверхность
    g = panel1.CreateGraphics();
}
```

```
// новая игра
private void newGame()
{
  int row, col;     // индексы клетки
  int n = 0;        // количество поставленных мин
  int k;            // кол-во мин в соседних клетках

  // очистить поле
  for(row = 1; row <= MR; row++)
      for(col = 1; col <= MC; col++)
          Pole[row,col] = 0;

  // инициализация генератора случайных чисел
  Random rnd = new Random();

  // расставим мины
  do
  {
      row = rnd.Next(MR) + 1;
      col = rnd.Next(MC) + 1;

      if (Pole[row,col] != 9)
      {
          Pole[row,col] = 9;
          n++;
      }
  }
  while (n != NM);

  // для каждой клетки вычислим кол-во
  // мин в соседних клетках
  for(row = 1; row <= MR; row++)
      for(col = 1; col <= MC; col++)
          if (Pole[row,col] != 9)
```

```
            {
                k = 0;

                if (Pole[row-1,col-1] == 9) k++;
                if (Pole[row-1,col]   == 9) k++;
                if (Pole[row-1,col+1] == 9) k++;
                if (Pole[row,col-1]   == 9) k++;
                if (Pole[row,col+1]   == 9) k++;
                if (Pole[row+1,col-1] == 9) k++;
                if (Pole[row+1,col]   == 9) k++;
                if (Pole[row+1,col+1] == 9) k++;

                Pole[row,col] = k;
            }

    status = 0;       // начало игры
    nMin   = 0;       // нет обнаруженных мин
    nFlag  = 0;       // нет поставленных флагов
}

// рисует поле
private void showPole(Graphics g, int status)
{
    for(int row = 1; row <= MR; row++)
        for(int col = 1; col <= MC; col++)
            this.kletka(g, row, col, status);
}

// рисует клетку
private void kletka(Graphics g,
    int row, int col, int status)
{
    int x,y; // координаты левого верхнего угла клетки
```

```
x = (col - 1) * W + 1;
y = (row-1)* H + 1;

// не открытые клетки - серые
if (Pole[row,col] < 100)
    g.FillRectangle(SystemBrushes.ControlLight,
        x-1, y-1, W, H);

// открытые или помеченные клетки
if (Pole[row,col] >= 100) {

    // открываем клетку, открытые - белые
    if (Pole[row,col] != 109)
        g.FillRectangle(Brushes.White,
            x-1, y-1, W, H);
    else
        // на этой мине подорвались!
        g.FillRectangle(Brushes.Red,
            x-1, y-1, W, H);

    // если в соседних клетках есть мины,
    // указываем их количество
    if ((Pole[row,col] >= 101) && (Pole[row,col] <= 108))
        g.DrawString((Pole[row,col]-100).ToString(),
            new Font("Tahoma", 10,
                System.Drawing.FontStyle.Regular),
            Brushes.Blue, x+3, y+2);
}

// в клетке поставлен флаг
if (Pole[row,col] >= 200)
    this.flag(g, x, y);
```

```
    // рисуем границу клетки
    g.DrawRectangle(Pens.Black,
        x-1, y-1, W, H);

    // если игра завершена (status = 2),
    // показываем мины
    if ((status == 2) && ((Pole[row,col] % 10) == 9))
        this.mina(g, x, y);
}

// открывает текущую и все соседние с ней клетки,
// в которых нет мин
private void open(int row, int col)
{
    // координаты области вывода
    int x = (col-1)* W + 1,
        y = (row-1)* H + 1;

    if (Pole[row,col] == 0)
    {
        Pole[row,col] = 100;

        // отобразить содержимое клетки
        this.kletka(g, row, col, status);

        // открыть примыкающие клетки
        // слева, справа, сверху, снизу
        this.open(row, col-1);
        this.open(row-1, col);
        this.open(row, col+1);
        this.open(row+1, col);

        //примыкающие диагонально
        this.open(row-1,col-1);
```

```
        this.open(row-1,col+1);
        this.open(row+1,col-1);
        this.open(row+1,col+1);
    }
    else
        if ((Pole[row,col] < 100) &&
            (Pole[row,col] != -3))
        {
            Pole[row,col] += 100;
            // отобразить содержимое клетки
            this.kletka(g, row, col, status);
        }
}

// рисует мину
private void mina(Graphics g, int x, int y)
{
    // корпус
    g.FillRectangle(Brushes.Green,
        x+16, y+26, 8, 4);
    g.FillRectangle(Brushes.Green,
        x+8, y+30, 24, 4);
    g.DrawPie(Pens.Black,
        x+6, y+28, 28, 16, 0, -180);
    g.FillPie(Brushes.Green,
        x+6, y+28, 28, 16, 0, -180);

    // полоса на корпусе
    g.DrawLine(Pens.Black,
        x+12, y+32, x+28, y+32);

    // вертикальный "ус"
    g.DrawLine(Pens.Black,
        x+20, y+22, x+20, y+26);
```

```
    // боковые "усы"
    g.DrawLine(Pens.Black,
        x+8, y+30, x+6, y+28);
    g.DrawLine(Pens.Black,
        x+32, y+30, x+34, y+28);
  }

  // рисует флаг
  private void flag(Graphics g, int x, int y)
  {
    Point[] p = new Point[3];
    Point[] m = new Point[5];

    // флажок
    p[0].X = x+4;   p[0].Y = y+4;
    p[1].X = x+30;  p[1].Y = y+12;
    p[2].X = x+4;   p[2].Y = y+20;
    g.FillPolygon(Brushes.Red, p);

    // древко
    g.DrawLine(Pens.Black,
        x+4, y+4, x+4, y+35);

    // буква M на флажке
    m[0].X = x+8;   m[0].Y = y+14;
    m[1].X = x+8;   m[1].Y = y+8;
    m[2].X = x+10;  m[2].Y = y+10;
    m[3].X = x+12;  m[3].Y = y+8;
    m[4].X = x+12;  m[4].Y = y+14;
    g.DrawLines(Pens.White, m);
  }

  // щелчок кнопкой в клетке игрового поля
  private void panel1_MouseClick(object sender,
                                 MouseEventArgs e)
```

```
{
  // игра завершена
  if (status == 2) return;

  // первый щелчок
  if (status == 0) status = 1;

  // преобразуем координаты мыши в индексы
  // клетки поля, в которой был сделан щелчок;
  // (e.X, e.Y) - координаты точки формы,
  // в которой была нажата кнопка мыши
  int row = (int)(e.Y/H) + 1,
      col = (int)(e.X/W) + 1;

  // координаты области вывода
  int x = (col-1)* W + 1,
      y = (row-1)* H + 1;

  // щелчок левой кнопки мыши
  if (e.Button == MouseButtons.Left)
  {
      // открыта клетка, в которой есть мина
      if (Pole[row,col] == 9)
      {
          Pole[row,col] += 100;

          // игра закончена
          status = 2;

          // перерисовать форму
          this.panel1.   Invalidate();
      }
```

```
    else
        if (Pole[row,col] < 9)
            this.open(row,col);
}

// щелчок правой кнопки мыши
if (e.Button == MouseButtons.Right) {

    // в клетке не было флага, ставим его
    if (Pole[row,col] <= 9) {
        nFlag += 1;

        if (Pole[row,col] == 9)
            nMin += 1;

        Pole[row,col] += 200;

        if ((nMin == NM) && (nFlag == NM)) {
            // игра закончена
            status = 2;

            // перерисовываем все игровое поле
            this.Invalidate();
        }
        else
            // перерисовываем только клетку
            this.kletka(g, row, col, status);
    }
    else
        // в клетке был поставлен флаг,
        // повторный щелчок правой кнопки мыши
        // убирает его и закрывает клетку
        if (Pole[row,col] >= 200)
```

```
            {
                nFlag -= 1;
                Pole[row,col] -= 200;

                // перерисовываем клетку
                this.kletka(g, row, col, status);
            }
    }
}

// команда Новая игра
private void новаяИграToolStripMenuItem_Click
                        (object sender, EventArgs e)
{
  newGame();
  showPole(g, status);
}

// обработка события Paint панели
private void panel1_Paint(object sender, PaintEventArgs e)
{
  showPole(g, status);
}

// выбор в меню Справка команды О программе
private void оПрограммеToolStripMenuItem_Click
                        (object sender, EventArgs e)
{
  Form2 aboutBox = new Form2();
  aboutBox.ShowDialog();
}

// выбор в меню Справка команды Правила игры
private void правилаToolStripMenuItem_Click
                        (object sender, EventArgs e)
```

```
  {
    Help.ShowHelp(this,
              helpProvider1.HelpNamespace,"saper_02.htm");
  }
 }
}
```

Листинг 1.33. Модуль формы *О программе*

```
// модуль формы "О программе"
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 Form2()
        {
            InitializeComponent();
        }

        // щелчок на WEB-ссылке
        private void linkLabel1_LinkClicked
              (object sender, LinkLabelLinkClickedEventArgs e)
        {
```

```
            string webRef = linkLabel1.Text;
            System.Diagnostics.Process.Start(webRef);
        }
    }
}
```

Будильник

Программа **Будильник** выводит в установленный пользователем момент времени окно с напоминанием о необходимости что-либо сделать. Особенность программы в том, что ее значок отображается в системной области панели задач. При наведении указателя мыши на значок появляется окно, в котором отображается время, на которое установлен будильник (рис. 1.42).

Рис. 1.42. Значок программы **Будильник** отображается в системной области панели задач

Рис. 1.43. В результате щелчка правой кнопкой мыши на значке программы появляется меню команд

Кроме того, в результате щелчка на значке правой кнопкой мыши появляется контекстное меню (рис. 1.43), в котором можно вы-

брать команду управления программой. Форма программы **Будильник** приведена на рис. 1.44, текст — в листинге 1.34. Форма окна сообщения — на рис. 1.45.

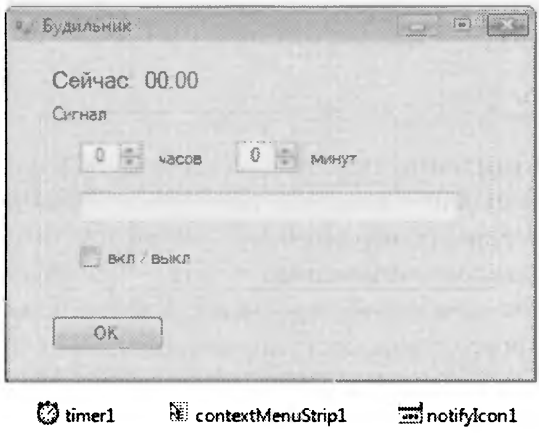


Рис. 1.44. Главная форма программы **Будильник**

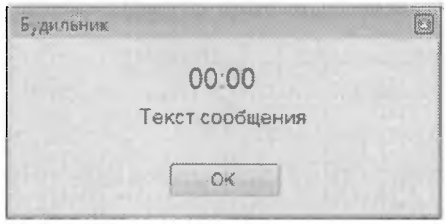


Рис. 1.45. Форма сообщения

Листинг 1.34. Модуль главной формы программы *Будильник*

```
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
  {
    // функция PlaySound, обеспечивающая воспроизведение
    // wav-файлов, находится в библиотеке winmm.dll
    // подключим эту библиотеку
    [System.Runtime.InteropServices.DllImport("winmm.dll")]
    private static extern
      Boolean PlaySound(string lpszName, int hModule, int
dwFlags);

    // время сигнала (отображения сообщения)
    private DateTime alarm;

    public Form1()
    {
        InitializeComponent();
        // параметры компонентов numericUpDown
        numericUpDown1.Maximum = 23;
        numericUpDown1.Minimum = 0;

        numericUpDown2.Maximum = 59;
        numericUpDown2.Minimum = 0;

        numericUpDown1.Value = DateTime.Now.Hour;
        numericUpDown2.Value = DateTime.Now.Minute;

        notifyIcon1.Visible = false;
```

```
    // период обработки сигнала от таймера
    timer1.Interval = 1000;
    timer1.Enabled = true;

    label2.Text = DateTime.Now.ToLongTimeString();
}

private void timer1_Tick(object sender, EventArgs e)
{
    label2.Text = DateTime.Now.ToLongTimeString();

    // будильник установлен
    if (checkBox1.Checked)
    {
        // время срабатывания будильника
        if (DateTime.Compare(DateTime.Now, alarm) > 0)
        {
            checkBox1.Checked = false;

           PlaySound(Application.StartupPath +
                "\\ring.wav", 0, 1);

            Form2 frm = new Form2();
            // чтобы получить доступ к компонентам формы
            // frm (Form2), их надо объявить как public
            // (см. Form2.Designer.cs)
            frm.label1.Text =
                   DateTime.Now.ToShortTimeString();
            frm.label2.Text = this.textBox1.Text;
            frm.ShowDialog();

            this.Show();
        }
    }
}
```

```
private void checkBox1_CheckedChanged(object sender,
                                      EventArgs e)
{
    if (checkBox1.Checked)
    {
        numericUpDown1.Enabled = false;
        numericUpDown2.Enabled = false;

        // установить время сигнала
        alarm = new DateTime(
            DateTime.Now.Year,
            DateTime.Now.Month,
            DateTime.Now.Day,
            Convert.ToInt16(numericUpDown1.Value),
            Convert.ToInt16(numericUpDown2.Value),
            0, 0);

        // если установленное время будильника меньше
        // текущего, нужно увеличить дату срабатывания
        // на единицу (+1 день)
        if (DateTime.Compare(DateTime.Now, alarm) > 0)
            alarm = alarm.AddDays(1);

        // текст подсказки
        notifyIcon1.Text =
                    alarm.ToShortTimeString() + "\n" +
                    textBox1.Text;

        button1.Enabled = true;
    }
    else
    {
        numericUpDown1.Enabled = true;
        numericUpDown2.Enabled = true;
```

```
            notifyIcon1.Text =
                "Будильник не установлен";

            button1.Enabled = false;
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        this.Hide();
        notifyIcon1.Visible = true;
    }

    // выбор в контекстном меню команды Показать/Свернуть
    private void toolStripMenuItem1_Click(object sender,
                                          EventArgs e)
    {
        if (this.Visible)
        {
            this.Hide();
        }

        else
        {
            this.Show();
            notifyIcon1.Visible = false;
        }
    }

    // выбор в контекстном меню команды О программе
    private void toolStripMenuItem2_Click(object sender,
                                          EventArgs e)
    {

    }
```

```
    // выбор в контекстном меню команды Завершить
    private void toolStripMenuItem3_Click(object sender,
                                          EventArgs e)
    {
        this.Close();
    }
  }
}
```

Экзаменатор

Программа **Экзаменатор** (листинг 1.35) позволяет автоматизировать процесс тестирования.

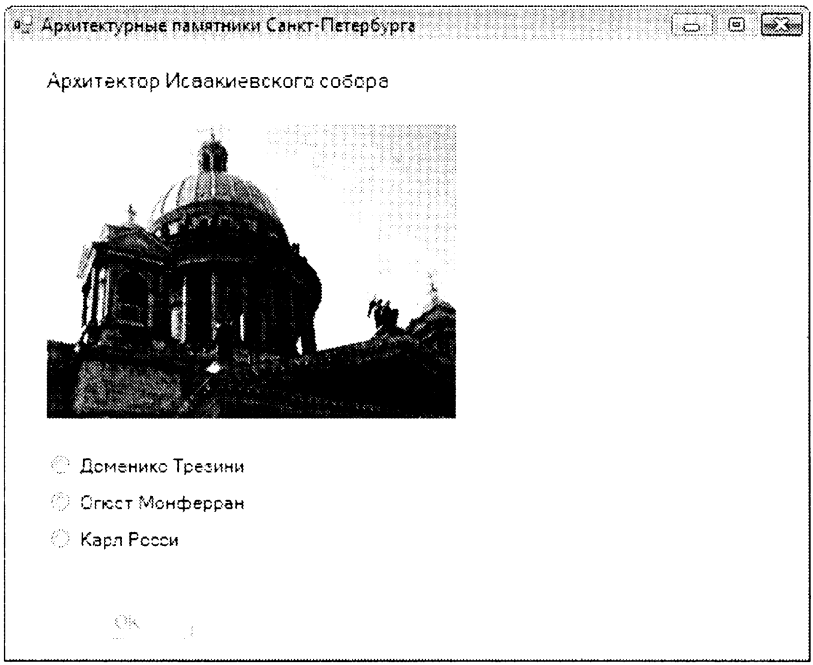


Рис. 1.46. Окно программы **Экзаменатор**

В окне программы (рис. 1.46) отображается тест — последовательность вопросов, на которые испытуемый должен ответить путем выбора правильного ответа из нескольких предложенных вариантов. Форма программы **Экзаменатор** приведена на рис. 1.47. В рассматриваемой программе вопросы загружаются из XML-файла (пример файла теста приведен в листинге 1.36). Имя файла теста передается программе при запуске — указывается в командной строке.

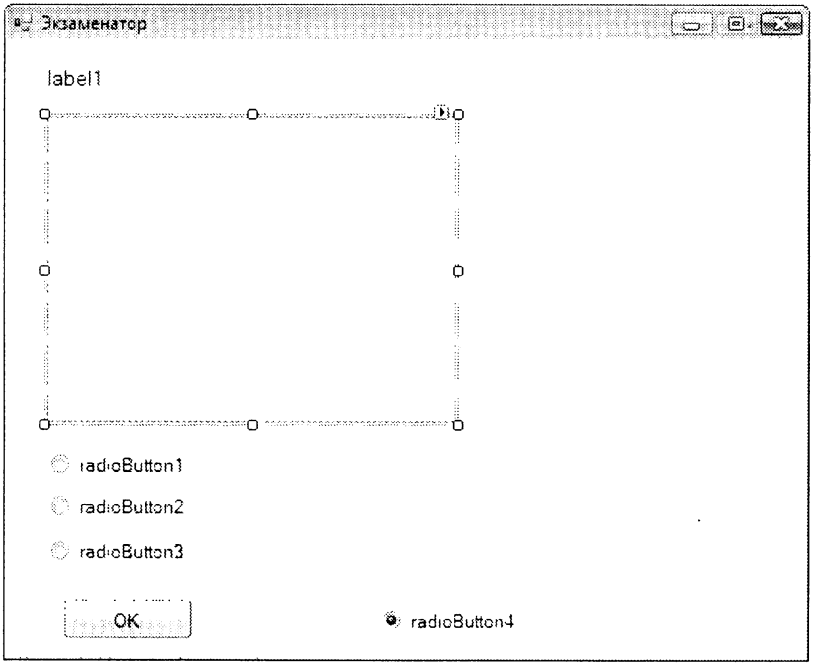


Рис. 1.47. Форма программы **Экзаменатор**

Листинг 1.35. Модуль формы программы *Экзаменатор*

```
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
  {
      string fpath; // путь к файлу теста
      string fname; // файл теста

      // XmlReader обеспечивает чтение данных xml-файла
      System.Xml.XmlReader xmlReader;

      string qw;      // вопрос

      // варианты ответа
      string[] answ = new string[3];

      string pic;     // путь к файлу иллюстрации

      int right; // правильный ответ (номер)
      int otv;   // выбранный ответ (номер)
      int n;     // количество правильных ответов
      int nv;    // общее количество вопросов
      int mode;  // состояние программы:
                        // 0 - начало работы;
                        // 1 - тестирование;
                        // 2 - завершение работы
```

```
// конструктор формы
// (см. также Program.cs )
public Form1(string[] args)
{
    InitializeComponent();

    radioButton1.Visible = false;
    radioButton2.Visible = false;
    radioButton3.Visible = false;

    // имя файла теста должно быть указано
    // в качестве параметра команды запуска программы
    if (args.Length > 0)
    {
       // указано только имя файла теста
       if (args[0].IndexOf(":") == -1) {
       fpath = Application.StartupPath + "\\";
       fname = args[0];
    }

    else
    {
      // указан путь к файлу теста
      fpath =
      args[0].Substring(0,args[0].LastIndexOf("\\")+1);
      fname =
      args[0].Substring(args[0].LastIndexOf("\\")+1);
    }

try
{
    // прочитать xml-файл
```

```
            xmlReader =
                new System.Xml.XmlTextReader(fpath + fname);
            xmlReader.Read();

            mode = 0;
            n    = 0;

            // загрузить и показать заголовок теста
            this.showHead();

            // загрузить и показать описание теста
            this.showDescription();
        }

        catch(Exception exc)
        {
            label1.Text = "Ошибка доступа к файлу  " +
                fpath + fname;

            MessageBox.Show("Ошибка доступа к файлу.\n" +
                fpath + fname + "\n",
                "Экзаменатор",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);

            mode = 2;
        }
    }
    else
    {
        label1.Text =
            "Файл теста необходимо указать " +
            "в команде запуска программы.\n" +
            "Например: 'exam economics.xml' " +
```

```
            "или 'exam c:\\spb.xml'.";
        mode = 2;
    }
}

// выводит название (заголовок) теста
private void showHead()
{
    // ищем узел <head>
    do xmlReader.Read();
    while(xmlReader.Name != "head");

    // считываем заголовок
    xmlReader.Read();

    // вывести название теста в заголовок окна
    this.Text = xmlReader.Value;

    // выходим из узла <head>
    xmlReader.Read();
}

    // выводит описание теста
    private void showDescription()
    {
        // ищем узел <description>
        do
            xmlReader.Read();
        while(xmlReader.Name != "description");

        // считываем описание теста
        xmlReader.Read();
```

```
    // выводим описание теста
    label1.Text = xmlReader.Value;

    // выходим из узла <description>
    xmlReader.Read();

    // ищем узел вопросов <qw>
    do
        xmlReader.Read();
    while(xmlReader.Name != "qw");

    // входим внутрь узла
    xmlReader.Read();
}

// читает вопрос из файла теста
private Boolean getQw() {
    // считываем тег <q>
    xmlReader.Read();

    if (xmlReader.Name == "q")
    {
        // здесь прочитан тег <q>,
        // атрибут text которого содержит вопрос,
        // а атрибут src - имя файла иллюстрации.

        // извлекаем значение атрибутов:
        qw  = xmlReader.GetAttribute("text");
        pic = xmlReader.GetAttribute("src");
        if (!pic.Equals(string.Empty)) pic = fpath + pic;

        // входим внутрь узла
        xmlReader.Read();
        int i = 0;
```

```
        // считываем данные узла вопроса <q>
        while (xmlReader.Name != "q")
        {
            xmlReader.Read();

            // варианты ответа
            if (xmlReader.Name == "a")
            {
                // если есть атрибут right, то это
                // правильный ответ
                if (xmlReader.GetAttribute("right") ==
                      "yes")
                    right = i;

                // считываем вариант ответа
                xmlReader.Read();
                if (i < 3) answ[i] = xmlReader.Value;

                // выходим из узла <a>
                xmlReader.Read();

                i++;
            }
        }

        // выходим из узла вопроса <q>
        xmlReader.Read();

        return true;
    }
    // если считанный тег не <q>
    else
        return false;
}
```

```
// выводит вопрос и варианты ответа
private void showQw() {
    // выводим вопрос
    label1.Text = qw;

    // иллюстрация
    if (pic.Length != 0)
    {
        try
        {
            pictureBox1.Image =
            new Bitmap(pic);

            pictureBox1.Visible = true;

            radioButton1.Top = pictureBox1.Bottom + 16;
        }
        catch
        {
            if (pictureBox1.Visible)
                pictureBox1.Visible = false;

            label1.Text +=
                "\n\n\nОшибка доступа к файлу " + pic + ".";

            radioButton1.Top = label1.Bottom + 8;
        }
    }
    else
    {
        if (pictureBox1.Visible)
            pictureBox1.Visible = false;
```

```
            radioButton1.Top = label1.Bottom;
        }

        // показать варианты ответа
        radioButton1.Text = answ[0];
        radioButton2.Top  = radioButton1.Top + 24;;
        radioButton2.Text = answ[1];
        radioButton3.Top  = radioButton2.Top + 24;;
        radioButton3.Text = answ[2];

        radioButton4.Checked = true;
        button1.Enabled = false;
    }

    // щелчок на кнопке выбора ответа
    // функция обрабатывает событие Click
    // компонентов radioButton1 - radioButton3
    private void radioButton1_Click(object sender, EventArgs e)
    {
        if ((RadioButton)sender == radioButton1) otv = 0;
        if ((RadioButton)sender == radioButton2) otv = 1;
        if ((RadioButton)sender == radioButton3) otv = 2;

        button1.Enabled = true;
    }

    // щелчок на кнопке Ok
    private void button1_Click_1(object sender, EventArgs e)
    {
        switch (mode)
        {
        case 0:         // начало работы программы
            radioButton1.Visible = true;
```

```
        radioButton2.Visible = true;
        radioButton3.Visible = true;

        this.getQw();
        this.showQw();

        mode = 1;

        button1.Enabled = false;
        radioButton4.Checked = true;
        break;

    case 1:
        nv++;

        // правильный ли ответ выбран
        if (otv == right) n++;

        if (this.getQw()) this.showQw();
        else {
            // больше вопросов нет
            radioButton1.Visible = false;
            radioButton2.Visible = false;
            radioButton3.Visible = false;

            pictureBox1.Visible  = false;

            // обработка и вывод результата
            this.showLevel();

            // следующий щелчок на кнопке Ok
            // закроет окно программы
            mode = 2;
        }
```

```
            break;
        case 2:   // завершение работы программы
            this.Close(); // закрыть окно
            break;
        }
    }

    // выводит оценку
    private void showLevel()
    {
        // ищем узел <levels>
        do
            xmlReader.Read();
        while (xmlReader.Name != "levels");

        // входим внутрь узла
        xmlReader.Read();

        // читаем данные узла
        while (xmlReader.Name != "levels")
        {
            xmlReader.Read();

            if (xmlReader.Name == "level")
                // n - кол-во правильных ответов,
                // проверяем, попадаем ли в категорию
                if (n >= System.Convert.ToInt32(
                    xmlReader.GetAttribute("score")))
                break;
        }

        // выводим оценку
        label1.Text =
            "Тестирование завершено.\n" +
```

```
            "Всего вопросов: " + nv.ToString() + ". " +
            "Правильных ответов: " + n.ToString() + ".\n" +
            xmlReader.GetAttribute("text");
    }
    }
}
```

Листинг 1.35. Пример файла теста

```
<?xml version="1.0" encoding="Windows-1251"?>
<test>
  <head>Архитектурные памятники Санкт-Петербурга</head>
  <description>Сейчас Вам будут предложены вопросы по архи-
тектуре Петербурга. Вы должны из предложенных нескольких ва-
риантов ответа выбрать правильный.</description>
  <qw>
    <q text="Архитектор Исаакиевского собора" src ="is.jpg" >
      <a right="no">Доменико Трезини</a>
      <a right="yes">Огюст Монферран</a>
      <a right="no">Карл Росси</a>
    </q>
    <q text="На фотографии" src ="marks.jpg">
      <a right="yes">Зимний дворец(Эрмитаж)</a>
      <a right="no">Мариинский дворец</a>
      <a right="no">Строгановский дворец</a>
      </q>
  </qw>
  <levels>
    <level score="2" text = "На все вопросы вы ответили
     правильно. Оценка - ОТЛИЧНО."/>
    <level score="1" text = "На некоторые вопросы вы ответили
     неправильно. Оценка - ХОРОШО."/>
    <level score="0" text = "Вы плохо подготовились
     к испытанию. Оценка - ПЛОХО!"/>
  </levels>
</test>
```

LINQ

В этом разделе собраны примеры, которые позволяют получить представление о LINQ (Language Integrated Query) — интегрированном языке запросов.

Общие замечания

- LINQ позволяет оперировать с данными (массивами, списками, XML-документами и другими источниками данных).
- Выполнение операций над данными обеспечивают Query-операторы (q-операторы).
- Значением q-оператора может быть последовательность или элемент данных.
- Значением операторов `Where`, `Order` и `Select` является последовательность.
- Значением операторов `First()`, `Last()`, `ElementAt()`, `Count()`, `Min()`, `Max()`, `Contains()` и `Any()` является элемент.
- Чтобы использовать LINQ, в программу нужно добавить ссылку на пространства имен `System.Linq` и `System.Linq.XML`.

Поиск в массиве

Программа **LINQ – Поиск в массиве** (листинг 1.37), ее форма и окно (результат поиска элементов, удовлетворяющих критерию запроса) приведены на рис. 1.48 и 1.49, демонстрирует использование LINQ для поиска информации в массиве. Выбор информации обеспечивает оператор `Where`.

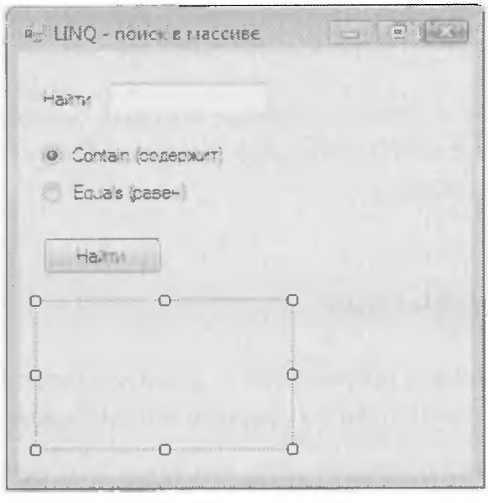


Рис. 1.48. Форма программы
LINQ – Поиск в массиве

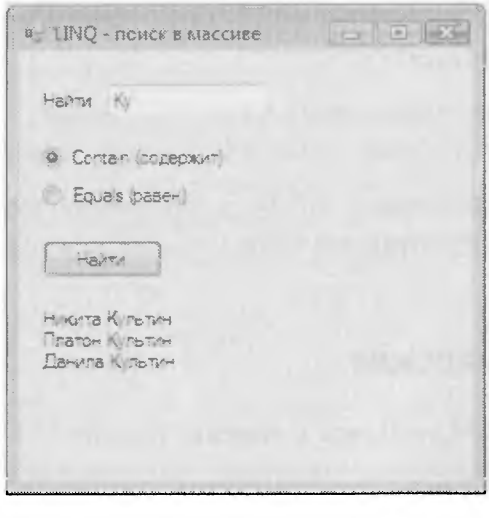


Рис. 1.49. Окно программы
LINQ – Поиск в массиве

Листинг 1.37. Модуль формы программы *LINQ – Поиск в массиве*

```
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
  {
    // массив
    string[] names = { "Никита Культин", "Платон Культин",
                       "Данила Культин", "Лариса Цой",};

    public Form1()
    {
        InitializeComponent();
    }

    // щелчок на кнопке найти
    private void button1_Click(object sender, EventArgs e)
    {
        // образец для поиска
        string aName;

        // результат поиска
        IEnumerable<string> filteredNames;

        aName = textBox1.Text;
```

```
        if (radioButton1.Checked)
        {
            // поиск подстроки
            filteredNames =
                System.Linq.Enumerable.Where
                        (names, n => n.Contains(aName));
        }
        else
        {
            // равенство
            filteredNames =
                System.Linq.Enumerable.Where
                        (names, n => n.Equals(aName));
        }

        label1.Text = "";

        foreach (string st in filteredNames)
            label1.Text = label1.Text + st + '\n';
    }
  }
}
```

Поиск в массиве-2

Программа **LINQ – Поиск в массиве-2** (ее форма и окно приведены на рис. 1.50 и 1.51 соответственно, а текст — в листинге 1.38) демонстрирует использование LINQ для поиска информации в массиве. В отличие от предыдущей программы, поиск дает положительный результат независимо от того, в каком регистре (верхнем или нижнем) введен критерий отбора записей.

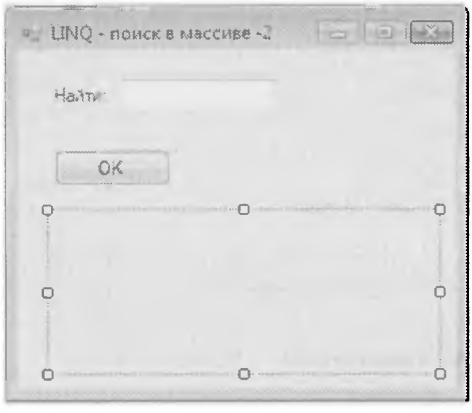


Рис. 1.50. Форма программы
LINQ – Поиск в массиве-2

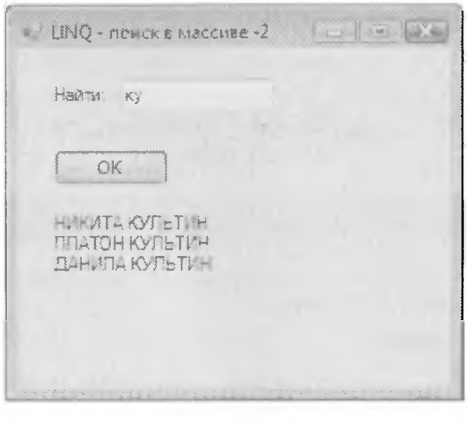


Рис. 1.51. Окно программы
LINQ – Поиск в массиве-2

Листинг 1.38. Модуль формы программы *LINQ – Поиск в массиве-2*

```
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 WindowsFormsApplication2
{
  public partial class Form1 : Form
  {
    public Form1()
    {
        InitializeComponent();
    }

    // массив
    string[] names = { "Никита Культин", "Платон Культин",
                       "Данила Культин", "Лариса Цой" };

    private void button1_Click(object sender, EventArgs e)
    {
        // образец для поиска
        string aName;

        // результат поиска
        IEnumerable<string> filteredNames;

        aName = textBox1.Text;

        // преобразовать элементы массива к верхнему регистру
        filteredNames =
            System.Linq.Enumerable.Select
                        ( names, n => n.ToUpper());
```

```
        filteredNames =
                System.Linq.Enumerable.Where
    (filteredNames, n =>
n.Contains(textBox1.Text.ToUpper()));

        label1.Text = "";

        foreach (string st in filteredNames)
            label1.Text = label1.Text + st + '\n';
    }
  }
}
```

Обработка массива

Программа **LINQ – обработка массива** (рис. 1.52, листинг 1.39) демонстрирует использование q-операторов для поиска минимального и максимального элементов, вычисления суммы элементов, а также для выполнения сортировки.

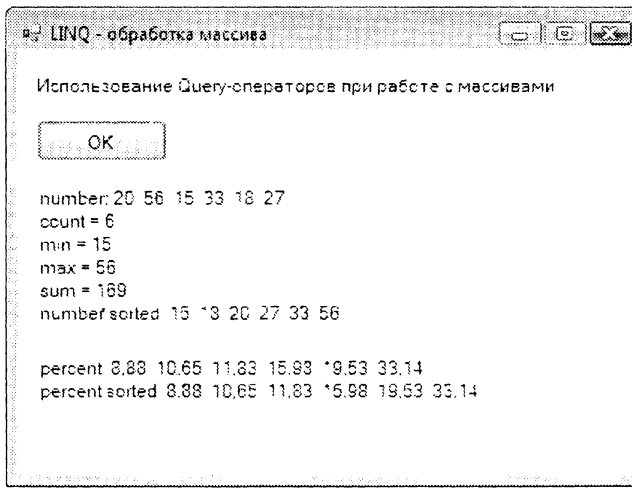


Рис. 1.52. Окно программы **LINQ – обработка массива**

Листинг 1.39. Модуль формы программы *LINQ – обработка массива*

```
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 Form1()
    {
        InitializeComponent();
    }

    // массив
    double[] numbers = { 20, 56, 15, 33, 18, 27 };

    private void button1_Click(object sender, EventArgs e)
    {
        double count, min, max, sum;

        count = numbers.Count();  // количество элементов
        max = numbers.Max();      // максимальный элемент
        min = numbers.Min();      // минимальный элемент
        sum = numbers.Sum();      // сумма элементов массива

        label2.Text = "number: ";
```

```
for (int i = 0; i < count; i++)
{
    label2.Text =
       label2.Text + numbers[i].ToString() + "  ";
}

label2.Text = label2.Text +
              "\ncount = " + count.ToString() +
              "\nmin = " + min.ToString() +
              "\nmax = " + max.ToString() +
              "\nsum = " + sum.ToString();

// список sorted содержит упорядоченные
// по возрастанию элементы массива numbers
IEnumerable<double> sorted = numbers.OrderBy(n => n);

label2.Text = label2.Text + "\nnumber sorted: ";
foreach (double a in sorted)
    label2.Text = label2.Text + a.ToString() + "  ";


// элемент percent(i) - доля i-го элемента sorted[i]
// в общей сумме элементов массива sorted
IEnumerable<double> percent =
         numbers.Select(n => n/sum*100);

percent = percent.OrderBy(n => n);

label3.Text = "percent: ";
foreach (double a in percent)
    label3.Text =label3.Text + a.ToString("f") + "  ";

label3.Text = label3.Text + "\npercent sorted: ";
foreach (double a in percent)
```

```
            label3.Text = label3.Text + a.ToString("f")+ "  ";
    }
  }
}
```

Обработка массива записей

Программа **LINQ – обработка массива записей** (рис. 1.53, листинг 1.40) показывает использование q-операторов для обработки массива записей.

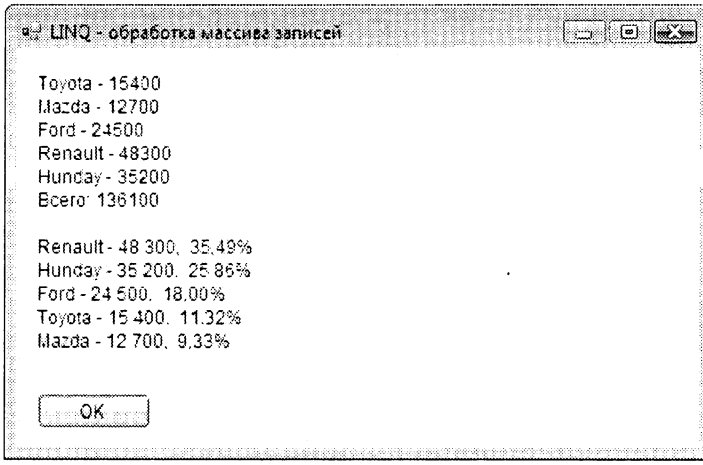


Рис. 1.53. Окно программы
LINQ – обработка массива записей

Листинг 1.40. Модуль формы программы *LINQ – обработка массива записей*

```
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
  {
    class sale
    {
        public string Title;
        public int n;
        public double p;
    }

    sale[] sales = {
              new sale { Title = "Toyota", n = 15400 },
              new sale { Title = "Mazda", n = 12700 },
              new sale { Title = "Ford", n = 24500 },
              new sale { Title = "Renault", n = 48300 },
              new sale { Title = "Hunday", n = 35200 }
    };

    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        // отобразить исходные данные
        string st = "";
```

```
        foreach (sale aSale in sales)
            st = st + aSale.Title + " - " +
                      aSale.n.ToString() + "\n";

        // обработка данных
        long sum = 0;
        // IEnumerable<int> count = sales.Select(a => a.n);
        // sum = count.Sum();

        foreach (sale aSale in sales)
            sum = sum + aSale.n;

        st = st + "Всего: " + sum.ToString();

        // вычислить долю каждой категории в общей сумме
        foreach (sale aSale in sales)
            aSale.p = (double)aSale.n / sum;

        // сортировка:
        // OrderBy - по возрастанию
        // OrderByDescending - по убыванию
        st = st + "\n\n";
        IEnumerable<sale> sorted =
                  sales.OrderByDescending(a => a.n);

        // вывод результата
        foreach (sale aSale in sorted)
            st = st + aSale.Title + " - " +
                     aSale.n.ToString("### 000") + ",  " +
                    (aSale.p * 100).ToString("f") + "%\n";

        label1.Text = st;
    }
}

}
```

Работа с XML-документами

Программа **LINQ to XML** (рис. 1.54, листинг 1.41) показывает, как, используя LINQ, создать XML-документ, загрузить существующий документ и добавить в него информацию. Кроме этого, программа показывает, как выбрать из XML-документа нужную информацию и отобразить ее в поле компонента `ListView`.

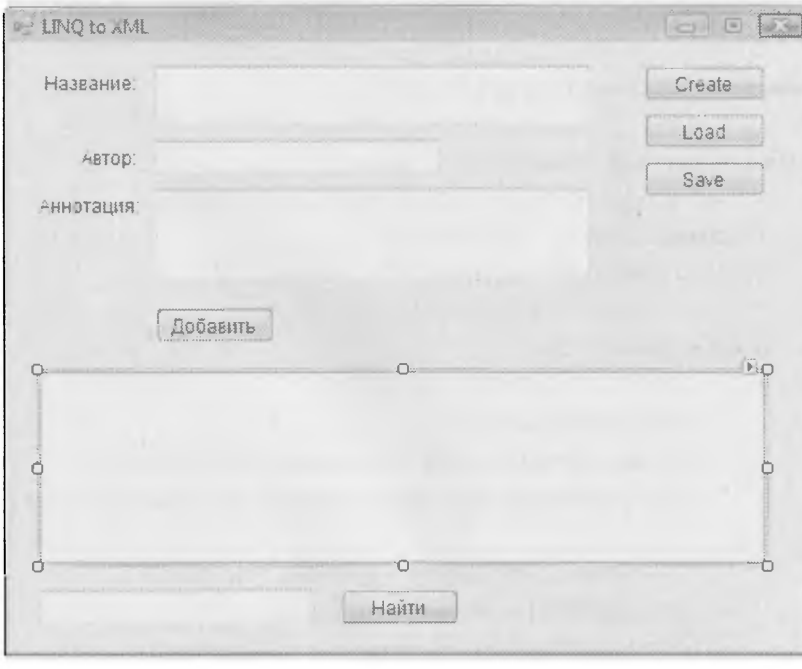


Рис. 1.54. Форма программы **LINQ to XML**

Листинг 1.41. Модуль формы программы *LINQ to XML*

```
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;

// эти ссылки вставлены вручную
using System.Xml.Linq;
using System.IO;

namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    XDocument doc;
    DirectoryInfo di;

    public Form1()
    {
        InitializeComponent();
        di = new DirectoryInfo(Environment.GetFolderPath
                (Environment.SpecialFolder.ApplicationData));

        // настройка компонента listView
        listView1.View = View.Details;
        listView1.GridLines = true;
        //listView1.Sorting = SortOrder.Ascending;

        listView1.Columns.Add("Автор");
        listView1.Columns[0].Width = 90;
        listView1.Columns.Add("Название");
        listView1.Columns[1].Width =
             listView1.Width - listView1.Columns[0].Width - 4;
    }
```

```
// загрузить XML-документ из файла
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        doc = XDocument.Load(di.FullName +
        "\\books.xml");
        button1.Enabled = true;
        button4.Enabled = true;
        button5.Enabled = true;
    }
    catch (Exception aException)
    {
        MessageBox.Show(aException.Message,
                        "Load XML",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }

    IEnumerable<XElement> books = doc.Elements();
    foreach (XElement aBooks in books.Elements())
    {
      listView1.Items.Add
              (aBooks.Element("Author").Value);
      listView1.Items[listView1.Items.Count -
      1].SubItems.
                    Add(aBooks.Element("Title").Value);
    }
    button2.Enabled = false;
}

// добавить элемент в XML-документ
private void button1_Click(object sender, EventArgs e)
```

```
{
    doc.Element("books").Add(
           new XElement("book",
              new XElement("Author", textBox2.Text),
              new XElement("Title", textBox1.Text),
              new XElement("Description", textBox4.Text)
              )
           );
}

// создать XML-файл
private void button2_Click(object sender, EventArgs e)
{
    FileInfo fi = new
    FileInfo(di.FullName+"\\books.xml");
    if (fi.Exists)
    {
        DialogResult dr;
        dr = MessageBox.Show("Файл " + fi.FullName +
               " существует.\nЗаменить его новым?",
               "Create XML",
               MessageBoxButtons.YesNo,
               MessageBoxIcon.Warning,
               MessageBoxDefaultButton.Button2);
        if (dr == DialogResult.No)
        {
            return;
        }
    }

    doc =
      new XDocument(
        new XElement("books",
          new XElement("book",
```

```
                new XElement("Author","Культин Н.Б."),
                new XElement("Title","Visual C# в примерах"),
                 new XElement("Description", "")
            ),
            new XElement("book",
                new XElement("Author", "Н. Культин"),
                new XElement("Title",
                             "Самоучитель Visual C#"),
                 new XElement("Description", "")
                ),
            new XElement("book",
                 new XElement("Author", "Культин"),
                 new XElement("Title",
                              "Turbo Pascal 7.0 и Delphi"),
                 new XElement("Description", "")
            )
        )
      );
    doc.Save(di.FullName+"\\books.xml");
}

// сохранить изменения
private void button4_Click(object sender, EventArgs e)
{
    doc.Save(di.FullName + "\\books.xml");
}

// щелчок на кнопке Найти
private void button5_Click(object sender, EventArgs e)
{
    // выбрать элементы, у которых поле Author
    // содержит текст, введенный пользователем
    IEnumerable<XElement> query =
        from b in doc.Elements().Elements()
```

```
            where b.Elements("Author").Any
               (n => n.Value.Contains(textBox3.Text))
            select b;

        listView1.Items.Clear();

        // вывести список элементов
        foreach (XElement el in query)
        {
            //st = st + "\n" + xe.ToString();
            bool firstElement = true;
            foreach (XElement e2 in el.Elements())
            {
                if ( firstElement == true)
                {
                    listView1.Items.Add(e2.Value);
                    firstElement = false;
                }
                else
                {
                  listView1.Items[listView1.Items.Count-1]
                                    .SubItems.Add(e2.Value);
                }
             }
        }
    }
  }
}
```

Отображение XML-документа

Программа **LINQ – Show XML** (рис. 1.55, листинг 1.42) показывает, как, используя LINQ, можно прочитать XML-файл и ото-

бразить информацию, находящуюся в нем, в поле компонента `ListView`.

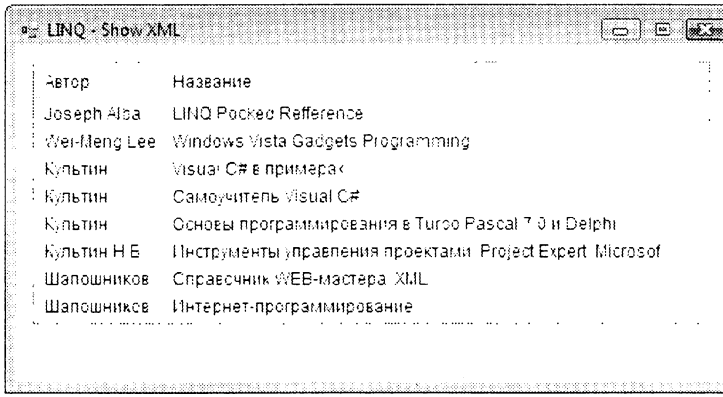


Рис. 1.55. Программа **LINQ – Show XML**

Листинг 1.42. Модуль формы программы *LINQ – Show XML*

```
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;

// эти ссылки вставлены вручную
using System.Xml.Linq;
using System.IO;

namespace WindowsFormsApplication1
{
```

```
public partial class Form1 : Form
{
  XDocument doc;
  DirectoryInfo di;

  public Form1()
  {
      InitializeComponent();

      // настройка компонента listView
      listView1.View = View.Details;
      listView1.GridLines = true;
      //listView1.Sorting = SortOrder.Ascending;

      listView1.Columns.Add("Автор");
      listView1.Columns[0].Width = 90;
      listView1.Columns.Add("Название");
      listView1.Columns[1].Width =
               listView1.Width -
               listView1.Columns[0].Width - 4 -17;
      // 4 - ширина границы; 17 - ширина полосы прокрутки


      // предполагается, что XML-файл находится
      // в каталоге ApplicationData
      di = new DirectoryInfo(Environment.GetFolderPath
          (Environment.SpecialFolder.ApplicationData));

      // загрузить данные из XML-файла
      try
      {
          doc = XDocument.Load(di.FullName +
          "\\books.xml");
                  }
```

```
catch (Exception aException)
{
    MessageBox.Show(aException.Message,
                    "Load XML",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
    return;
}

// отобразить данные
IEnumerable<XElement> query =
    from b in doc.Elements().Elements()
    select b;

listView1.Items.Clear();

// * вывести список элементов (узлов) XML-документа *
// Чтобы добавить в компонент ListView строку,
// точнее ячейку в первый столбец,
// надо добавить элемент в коллекцию Items.
// Чтобы данные появились во второй строке,
// надо добавить подэлемент (SubItem) в коллекцию
// SubItems, соответствующую той строке,
// в которую надо добавить ячейку.
foreach (XElement e1 in query)
{
    bool newElement = true;
    foreach (XElement e2 in e1.Elements())
    {
      if (newElement == true)
      {
          // первый узел - элемент
          listView1.Items.Add(e2.Value);
```

```
                    newElement = false;
                }
                else
                {
                    // остальные узлы - подэлементы
                    listView1
                        .Items[listView1.Items.Count - 1]
                        .SubItems.Add(e2.Value);
                }
            }
        }
    }

}
```

Экзаменатор-2

Программа **Экзаменатор-2** является усовершенствованным вариантом приведенной ранее программы тестирования. Ее отличительная особенность в том, что при каждом запуске программы из вопросов, находящихся в файле, путем перемешивания формируется новый тест. Это позволяет использовать программу для группового тестирования в компьютерном классе, т. к. вероятность того, что на мониторах соседних компьютеров в один момент времени будут отображаться одинаковые вопросы, невелика и, следовательно, у испытуемых практически отсутствует возможность "списать" у соседа. Форма программы приведена на рис. 1.56. Нетрудно заметить, что на ней нет компонентов, обеспечивающих отображение вариантов ответа и выбор правильного ответа. Эти компоненты создаются во время работы программы, "в коде". Делает это конструктор формы. Следует обратить внимание на то, что свойству `AutoSize` компонентов `label` при-

сваивается значение `True`, и задается значение свойства `MaximumSize.Width`. Такая настройка обеспечивает возможность отображения в поле компонента нескольких строк текста. Модуль формы приведен в листинге 1.43, а в листинге 1.44 — пример файла теста, который позволяет понять требуемую структуру XML-документа.

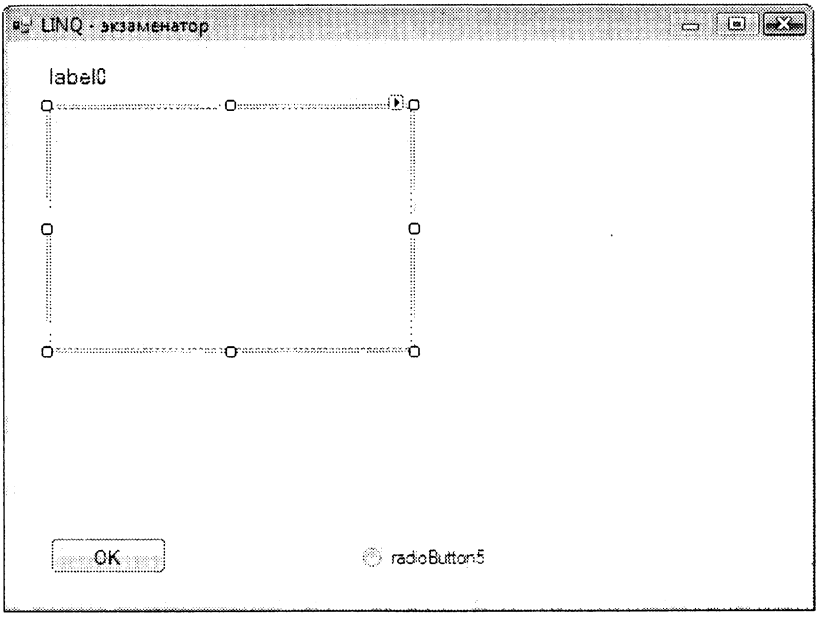


Рис. 1.56. Форма программы **Экзаменатор-2**

Листинг 1.43. Модуль формы программы *Экзаменатор-2*

```
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;

// эти ссылки вставлены вручную
using System.Xml.Linq;
using System.IO;

namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {

    // массив радиокнопок и меток
    RadioButton[] radioButton;
    Label[] label;

    XDocument xdoc;
    DirectoryInfo di;

    // тест - XML-элементы (узлы)
    IEnumerable<XElement> xel;


    string fpath; // путь к файлу теста
    string fname; // файл теста

    // тест - последовательность номеров вопросов
    int[] test;
    int cv = 0; // текущий вопрос

    int mode = 0;  // состояние программы:
                   // 0 - начало работы;
```

```
            // 1 - тестирование;
            // 2 - завершение работы

int otv;    // номер ответа, выбранного испытуемым
int right; // номер правильного ответа

int nr;      // количество правильных ответов
int n;       // общее количество вопросов

// имя файла теста надо указать в поле Command Line
// Arguments на вкладке Debug, для доступа к которой надо
// в меню Project выбрать команду Properties
public Form1(string[] args) //(см. также Program.cs )
{
    InitializeComponent();

    radioButton = new RadioButton[4];
    label = new Label[4];

    for (int i = 0; i < 4; i++)
    {
        // создать компонент RadioButton
        radioButton[i] = new
               System.Windows.Forms.RadioButton();

        radioButton[i].Location = new
              System.Drawing.Point(25, 20+i*16);
        radioButton[i].Name = "radioButton"+i.ToString();
        radioButton[i].Size = new
              System.Drawing.Size(14, 13);
        radioButton[i].Visible = false;

        // процедура обработки события Click
        radioButton[i].Click += new
```

```
            System.EventHandler(this.radioButton1_Click);
        radioButton[i].Parent = this;

        // создать компонент Label
        label[i] = new System.Windows.Forms.Label();

        label[i].AutoSize = true;
        label[i].Font = new System.Drawing.Font(
                    "Microsoft Sans Serif", 9.75F,
                     System.Drawing.FontStyle.Regular,
                    System.Drawing.GraphicsUnit.Point,
                    ((byte)(204)));
        label[i].Location = new
                 System.Drawing.Point(45, 20 + i*16);
        label[i].MaximumSize = new
                 System.Drawing.Size(400, 0);
        label[i].Name = "label" + i.ToString();
        label[i].Size = new System.Drawing.Size(45, 16);
        radioButton[i].Visible = false;
        label[i].Parent = this;
    }

    radioButton5.Checked = true;

    // имя файла теста должно быть указано
    // в качестве парамета команды запуска программы
    if (args.Length > 0)
    {
        // если указано только имя файла теста,
        // то предполагается, что файл находится
        // в папке C:\Users\<User>\AppData\Roaming
        if (args[0].IndexOf(":") == -1)
        {
```

```
            di = new DirectoryInfo(
              Environment.GetFolderPath(

            Environment.SpecialFolder.ApplicationData));
            fpath = di.FullName + "\\";
            fname = args[0];
        }

        else
        {
            // указано полное имя файла теста
            fpath = args[0].Substring(0,
                        args[0].LastIndexOf("\\") + 1);
            fname = args[0].Substring(
                       args[0].LastIndexOf("\\") + 1);
        }
    }
    else
    {
        // не указано имя файла теста
        label0.ForeColor = Color.DarkRed;
        label0.Text = "Не указано имя файла теста!";
        mode = 2;
        return;
    }

    try
    {
        xdoc = XDocument.Load(fpath + fname);

        xel = xdoc.Elements(); // прочитать XML-файл

        // информация о тесте
        label0.Text =
             xel.Elements("info").ElementAt(0).Value;
```

```
n = xel.Elements("queries").Elements().Count();

// генерируем тест
test  = new int[n]; // создать массив

// запишем в массив test числа от 0 до n-1
// так, чтобы каждое число было в массиве
// только один раз. Это и есть тест -
// перемешанные вопросы

Boolean[] b; // вспомогательный массив
             // b[i] == true, если число i
             // записано в массив test

b = new Boolean[n];
for (int i = 0; i < n; i++)
{
    b[i] = false;
}

Random rnd = new Random();
int r; // случайное число
for (int i = 0; i < n; i++)
{
    do
        r = rnd.Next(n);
    while ( b[r] == true );

    test[i] = r;
    b[r] = true;
}

mode = 0;
cv = 0;
```

```
        }
        catch (Exception aException)
        {
            // ошибка доступа к файлу теста
            label0.ForeColor = Color.DarkRed;
            label0.Text = aException.Message;
            mode = 2;
            return;
        }
    }

    // щелчок на кнопке OK
    private void button1_Click_1(object sender, EventArgs e)
    {
        switch (mode)
        {
            case 0:
                // отобразить первый вопрос
                qw(test[cv]);
                cv++;
                mode = 1;
                break;

            case 1:
                // переход к следующему вопросу
                if (otv == right)
                    nr++; // выбран правильный вариант ответа

                if (cv < n)
                {
                    // отобразить следующий вопрос
                    qw(test[cv]);
                    cv++;
                }
```

```
            else
            {
                // больше вопросов нет
                for (int j = 0; j < 4; j++)
                {
                    label[j].Visible = false;
                    radioButton[j].Visible = false;
                }

                pictureBox1.Visible = false;

                // обработка и вывод результата
                ShowResult();

                // следующий щелчок на кнопке Ok
                // закроет окно программы
                mode = 2;
            }
            break;

        case 2:   // завершение работы программы
            this.Close(); // закрыть окно
            break;
    }
}

// отображает вопрос
private void qw(int i)
{
    int j;
    for ( j= 0; j < 4; j++)
    {
        label[j].Visible = false;
```

```
            radioButton[j].Visible = false;
        }

        radioButton5.Checked = true;

        // вопрос
        label0.Text = xel.Elements("queries").Elements()
                    .ElementAt(i).Element("q").Value;

        // правильный ответ
        right = System.Convert.ToInt32(
              xel.Elements("queries").Elements()
              .ElementAt(i).Element("q").Attribute("right")
              .Value);

        // файл иллюстрации
        string src = xel.Elements("queries").Elements()
.ElementAt(i).Element("q").Attribute("src")
                    .Value;

        if (src.Length != 0)
        {
            // отобразить иллюстрацию
            try
            {
                pictureBox1.Image = new Bitmap(fpath + src);
                pictureBox1.Visible = true;

                radioButton[0].Top = pictureBox1.Bottom + 16;
                label[0].Top = radioButton[0].Top - 3;
            }
            catch
            {
```

```
            if (pictureBox1.Visible)
                pictureBox1.Visible = false;

            radioButton[0].Top = label0.Bottom + 10;
            label[0].Top = radioButton[0].Top - 3;
        }
    }
    else
    {
        pictureBox1.Visible = false;
        radioButton[0].Top = label0.Bottom + 10;
        label[0].Top = radioButton[0].Top - 3;
    }

    j = 0;
    foreach (XElement a in xel.Elements("queries")
                .Elements().ElementAt(i)
                .Element("as").Elements())
   {
       label[j].Text = a.Value;
       label[j].Visible = true;
       radioButton[j].Visible = true;
       if (j != 0)
       {
           radioButton[j].Top = label[j - 1].Bottom + 10;
           label[j].Top = radioButton[j].Top - 3;
       }
        j++;
   }
    button1.Enabled = false;
}

// щелчок на кнопке выбора ответа
// функция обрабатывает событие Click
```

```
// компонентов radioButton
private void radioButton1_Click(object sender,
                                EventArgs e)
{
    if ((RadioButton)sender == radioButton[0]) otv = 1;
    if ((RadioButton)sender == radioButton[1]) otv = 2;
    if ((RadioButton)sender == radioButton[2]) otv = 3;
    if ((RadioButton)sender == radioButton[3]) otv = 4;

    button1.Enabled = true;
}

// отобразить результат
private void ShowResult()
{
    int k; // количество уровней оценки

    int i;      // номер уровня
    int p = 0; // кол-во правильных ответов, необходимых
               // для достижения i-го уровня

    k = xel.Elements("levels").Elements().Count();

    for (i = 0; i < k-1; i++)
    {
        p = System.Convert.ToInt32(
             xel.Elements("levels").Elements()
             .ElementAt(i).Attribute("p").Value);
        if (nr >= p) // кол-во правильных ответов
                     // достаточно для достижения уровня
            break;
    }
```

```
        // отобразить оценку
        label0.Text =
            "Всего вопросов: " + n.ToString() + "\n" +
            "Правильных ответов: " + nr.ToString() + "\n" +
            "Оценка: " +
            xel.Elements("levels").Elements()
            .ElementAt(i).Value;
    }
  }
}
```

Листинг 1.44. Файл теста для программы *Экзаменатор-2*

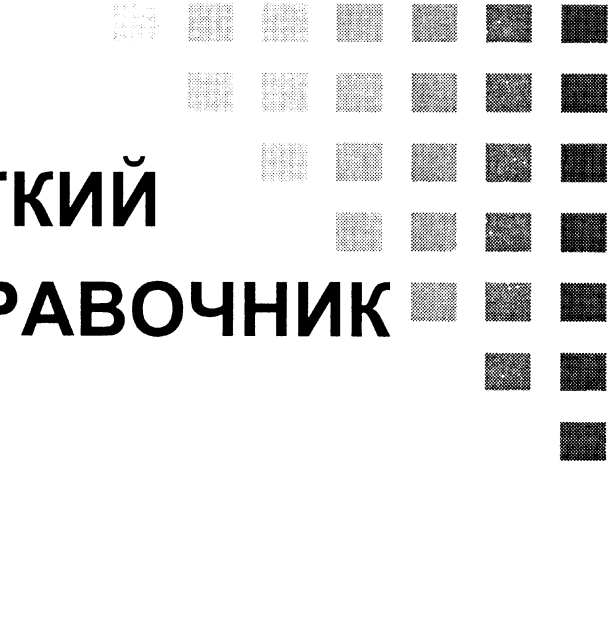
```
<?xml version="1.0" encoding="utf-8"?>
<test>
  <info>
     <descr>Основы Turbo Pascal</descr>
  </info>
  <levels>
      <level p="10">Отлично!</level>
      <level p="9">Хорошо</level>
      <level p="7">Удовлетворительно</level>
      <level p="6">Плохо</level>
  </levels>
  <queries>
    <qw>
     <q right="2" src="">Переменные A и B целого типа.
      Значение переменной A равно 10, переменной B - 2.
      Укажите тип выражения A/B</q>
    <as>
       <a>integer</a>
       <a>real</a>
```

```
    </as>
   </qw>
   <qw>
    <q right="2" src="">Переменные A и B целого типа.
     Значение переменной A равно 10, переменной B - 2.
     Какого типа должна быть переменная C, чтобы выражение
     C := A/B было верным?</q>
    <as>
       <a>integer</a>
       <a>real</a>
    </as>
   </qw>
<qw>
    <q right="1" src="">Какое из приведенных ниже объявлений
     массива правильное?</q>
    <as>
       <a>a:array[1..10] of integer</a>
       <a>array a[1..10] of integer</a>
       <a>a:array[1..10] integer</a>
    </as>
   </qw>
   <qw>
    <q right="2" src="">Переменные A и B целого типа.
     Укажите тип выражения A = B</q>
    <as*>
       <a>integer</a>
       <a>boolean</a>
    </as>
   </qw>
   <qw>
    <q right="3" src="">Инструкция a:array[25] of real </q>
    <as>
       <a>это объявление массива целого типа</a>
       <a>это объявление массива дробного типа</a>
       <a>Содержит ошибку, т.к. неправильно записан диапазон
        изменения индекса</a>
```

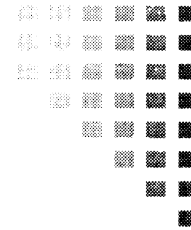
```
    </as>
   </qw>
  <qw>
      <q right="2" src="">Компилятор Turbo Pascal выводит
сообщение Type missmath в случае если</q>
    <as>
      <a>в объявлении переменной не указан ее тип</a>
      <a>тип выражения не соответствует типу переменной,
       которой присваивается значение</a>
    </as>
   </qw>
<qw>
      <q right="1" src="">Компилятор Turbo Pascal выводит
       сообщение Unknown identifier в случае если</q>
    <as>
      <a>переменная, указанная в каком-либо выражении, не
       объявлена</a>
      <a>неправильно записано имя переменной, например,
       объявлена переменная Sum, а в программе записано
      sum</a>
      <a>неправильно указан тип переменной</a>
    </as>
   </qw>
<qw>
      <q right="2" src="">В скобках после слова while
       указывается</q>
    <as>
      <a>условие завершения цикла</a>
      <a>условие выполнения инструкций цикла</a>
      <a>количество повторений инструкций цикла</a>
    </as>
   </qw>
<qw>
      <q right="1" src="">В скобках после слова until
       указывается</q>
    <as>
      <a>условие завершения цикла</a>
```

```
        <a>условие выполнения инструкций цикла</a>
        <a>количество повторений инструкций цикла</a>
      </as>
    </qw>
<qw>
      <q right="2" src="">Сколько раз будут выполнены
       инструкции тела цикла for i:=1 to n do,
       если значение переменной n равно нулю</q>
      <as>
        <a>1 раз</a>
        <a>ни разу</a>
      </as>
    </qw>
  </queries>
</test>
```

Часть 2



Краткий справочник



Часть 2

Краткий справочник

Форма

Свойства формы (объекта `WinForm`) приведены в табл. 2.1.

Таблица 2.1. *Свойства формы*

Свойство	Описание
`Name`	Имя формы
`Text`	Текст в заголовке
`Size`	Размер формы. Уточняющее свойство `Width` определяет ширину, свойство `Heigh` — высоту
`StartPosition`	Положение формы в момент первого появления на экране. Форма может находиться в центре экрана (`CenterScreen`), в центре родительского окна (`CenterParent`). Если значение свойства равно `Manual`, то положение формы определяется значением свойства `Location`.
`Location`	Положение формы на экране. Расстояние от верхней границы формы до верхней границы экрана задает уточняющее свойство `Y`, расстояние от левой границы формы до левой границы экрана — уточняющее свойство `X`

Таблица 2.1 *(продолжение)*

Свойство	Описание
FormBorder-Style	Тип формы (границы). Форма может представлять собой обычное окно (Sizable), окно фиксированного размера (FixedSingle, Fixed3D), диалог (FixedDialog) или окно без кнопок **Свернуть** и **Развернуть** (SizeableToolWindow, FixedToolWindow). Если свойству присвоить значение None , у окна не будет заголовка и границы
ControlBox	Управляет отображением системного меню и кнопок управления окном. Если значение свойства равно False, то в заголовке окна кнопка системного меню, а также кнопки **Свернуть**, **Развернуть**, **Закрыть** не отображаются
MaximazeBox	Кнопка **Развернуть**. Если значение свойства равно False, то находящаяся в заголовке окна кнопка **Развернуть** недоступна
MinimazeBox	Кнопка **Свернуть**. Если значение свойства равно False, то находящаяся в заголовке окна кнопка **Свернуть** недоступна
Icon	Значок в заголовке окна
Font	Шрифт, используемый по умолчанию компонентами, находящимися на поверхности формы. Изменение значения свойства приводит к автоматическому изменению значения свойства Font всех компонентов формы (при условии, что значение свойства компонента не было задано явно)
ForeColor	Цвет, наследуемый компонентами формы и используемый ими для отображения текста. Изменение значения свойства приводит к автоматическому изменению соответствующего свойства всех компонентов формы (при условии, что значение свойства Font компонента не было задано явно)

Таблица 2.1 (окончание)

Свойство	Описание
`BackColor`	Цвет фона. Можно задать явно (выбрать на вкладке **Custom** или **Web**) или указать элемент цветовой схемы (выбрать на вкладке **System**)
`Opacity`	Степень прозрачности формы. Форма может быть непрозрачной (100%) или прозрачной. Если значение свойства меньше 100%, то сквозь форму видна поверхность, на которой она отображается

Компоненты

В этом разделе кратко описаны основные (базовые) компоненты. Подробное описание этих и других компонентов можно найти в справочной системе.

Button

Компонент `Button` представляет собой командную кнопку. Свойства компонента приведены в табл. 2.2.

***Таблица 2.2.** Свойства компонента `Button`*

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст на кнопке
`TextAlign`	Положение текста на кнопке. Текст может располагаться в центре кнопки (`MiddleCenter`), быть прижат к левой (`MiddleLeft`) или правой (`MiddleRight`) границе.

Таблица 2.2 *(продолжение)*

Свойство	Описание
	Можно задать и другие способы размещения надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`FlatStyle`	Стиль. Кнопка может быть стандартной (`Standard`), плоской (`Flat`) или "всплывающей" (`Popup`)
`Location`	Положение кнопки на поверхности формы. Уточняющее свойство `X` определяет расстояние от левой границы кнопки до левой границы формы, уточняющее свойство `Y` — от верхней границы кнопки до верхней границы клиентской области формы (нижней границы заголовка)
`Size`	Размер кнопки
`Font`	Шрифт, используемый для отображения текста на кнопке
`ForeColor`	Цвет текста, отображаемого на кнопке
`Enabled`	Признак доступности кнопки. Кнопка доступна, если значение свойства равно `True`, и недоступна, если значение свойства равно `False` (в этом случае нажать кнопку нельзя, событие `Click` в результате щелчка на ней не возникает)
`Visible`	Позволяет скрыть кнопку (`False`) или сделать ее видимой (`True`)
`Cursor`	Вид указателя мыши при позиционировании указателя на кнопке
`Image`	Картинка на поверхности кнопки. Рекомендуется использовать gif-файл, в котором определен прозрачный цвет

Таблица 2.2 (окончание)

Свойство	Описание
ImageAlign	Положение картинки на кнопке. Картинка может располагаться в центе (MiddleCenter), быть прижата к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения картинки на кнопке (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
ImageList	Набор изображений, из которых может быть выбрано то, которое будет отображаться на поверхности кнопки. Представляет собой объект типа ImageList. Чтобы задать значение свойства, в форму приложения нужно добавить компонент ImageList
ImageIndex	Номер (индекс) изображения из набора ImageList, которое отображается на кнопке
ToolTip	Подсказка, появляющаяся рядом с указателем мыши при его позиционировании на кнопке. Чтобы свойство стало доступно, в форму приложения нужно добавить компонент ToolTip

ComboBox

Компонент ComboBox представляет собой комбинацию поля редактирования и списка, что дает возможность ввести данные путем набора на клавиатуре или выбором из списка. Свойства компонента приведены в табл. 2.3.

Таблица 2.3. *Свойства компонента* `ComboBox`

Свойство	Описание
`DropDownStyle`	Вид компонента: `DropDown` — поле ввода и раскрывающийся список; `Simple` — поле ввода со списком; `DropDownList` — раскрывающийся список
`Text`	Текст, находящийся в поле редактирования (для компонентов типа `DropDown` и `Simple`)
`Items`	Элементы списка — коллекция строк
`Items.Count`	Количество элементов списка
`SelectedIndex`	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
`Sorted`	Признак автоматической сортировки (`True`) списка после добавления очередного элемента
`MaxDropDownItems`	Количество отображаемых элементов в раскрытом списке. Если количество элементов списка больше, чем `MaxDropDownItems`, то появляется вертикальная полоса прокрутки
`Location`	Положение компонента на поверхности формы
`Size`	Размер компонента без (для компонентов типа `DropDown` и `DropDownList`) или с учетом (для компонента типа `Simple`) размера области списка или области ввода
`DropDownWidth`	Ширина области списка
`Font`	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

ContextMenuStrip

Компонент ContextMenuStrip представляет собой контекстное меню — список команд, который отображается в результате щелчка правой кнопкой мыши на форме или в поле компонента. Команды меню определяет значение свойства Items, представляющего собой коллекцию объектов MenuItem. Свойства объекта MenuItem приведены в табл. 2.4. Чтобы задать контекстное меню компонента или формы, нужно указать имя контекстного меню (компонента ContextMenuStrip) в качестве значения свойства ContextMenuStrip.

Таблица 2.4. *Свойства объекта* `MenuItem`

Свойство	Описание
`Text`	Команда
`Enabled`	Признак доступности команды. Если значение свойства равно `False`, то команда недоступна (название команды отображается серым цветом)
`Image`	Картинка, которая отображается в строке команды
`Checked`	Признак того, что элемент меню выбран. Если значение свойства равно `True`, то элемент считается выбранным и помечается галочкой или (если значение свойства `RadioCheck` равно `True`) точкой. Свойство `Checked` обычно используется для тех элементов меню, которые предназначены для отображения параметров
`RadioCheck`	Признак того, что для индикации состояния свойства `Checked` используется точка (`True`), а не галочка (`False`)

CheckBox

Компонент CheckBox представляет собой переключатель, который может находиться в одном из двух состояний: выбранном или невыбранном (часто вместо "выбранный" говорят "установленный", а вместо "невыбранный" — "сброшенный"). Свойства компонента CheckBox приведены в табл. 2.5.

Таблица 2.5. *Свойства компонента* CheckBox

Свойство	Описание
Text	Текст, отображаемый справа от кнопки
Checked	Состояние переключателя. Если переключатель выбран (в поле компонента отображается галочка), то значение свойства равно True. Если переключатель сброшен (галочка не отображается), то значение свойства равно False
TextAllign	Положение текста в поле отображения текста. Текст может располагаться в центре поля (MiddleCenter), быть прижат к левой (MiddleLeft) или правой (MiddleRight) границе. Можно задать и другие способы размещения текста надписи (TopLeft, TopCenter, TopRight, BottomLeft, BottomCenter, BottomRight)
CheckAllign	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (TopLeft), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (MiddleLeft). Есть и другие варианты размещения кнопки в поле компонента
Enabled	Управляет доступностью компонента. Позволяет сделать переключатель недоступным (False)

Таблица 2.5 *(окончание)*

Свойство	Описание
`Visible`	Управляет видимостью компонента. Позволяет скрыть, сделать невидимым (`False`) переключатель
`AutoCheck`	Определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно `True`
`FlatStyle`	Стиль (вид) переключателя. Переключатель может быть обычным (`Standard`), плоским (`Flat`) или "всплывающим" (`Popup`). Стиль определяет поведение переключателя при позиционировании указателя мыши на его изображении
`Appearance`	Определяет вид переключателя. Переключатель может выглядеть обычным образом (`Normal`) или как кнопка (`Button`)
`Image`	Картинка, которая отображается в поле компонента
`ImageAlign`	Положение картинки в поле компонента. Картинка может располагаться в центре (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`ImageList`	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения следует добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) картинки из набора `ImageList`, которая отображается в поле компонента

CheckedListBox

Компонент CheckedListBox представляет собой список, перед каждым элементом которого находится переключатель CheckBox. Свойства компонента CheckedListBox приведены в табл. 2.6.

Таблица 2.6. *Свойства компонента* CheckedListBox

Свойство	Описание
Items	Элементы списка — коллекция строк
Items.Count	Количество элементов списка
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
CheckOnClick	Способ пометки элемента списка. Если значение свойства равно False, то первый щелчок выделяет элемент списка (строку), а второй устанавливает в выбранное состояние переключатель. Если значение свойства равно True, то щелчок на элементе списка выделяет элемент и устанавливает во включенное состояние переключатель
CheckedItems	Элементы, выбранные в списке
CheckedItems.Count	Количество выбранных элементов
CheckedIndices	Коллекция, элементы которой содержат номера выбранных элементов списка
MultiColumn	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка

Таблица 2.6 (окончание)

Свойство	Описание
Location	Положение компонента на поверхности формы
Size	Размер компонента без (для компонентов типа DropDown и DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

GroupBox

Компонент GroupBox представляет собой контейнер для других компонентов. Обычно он используется для объединения в группы компонентов RadioButton по функциональному признаку. Свойства компонента GroupBox приведены в табл. 2.7.

Таблица 2.7. Свойства компонента GroupBox

Свойство	Описание
Text	Заголовок — текст, поясняющий назначение компонентов, которые находятся в поле компонента GroupBox
Enabled	Позволяет управлять доступом к компонентам, находящимся в поле (на поверхности) компонента GroupBox. Если значение свойства равно False, то все находящиеся в поле GroupBox компоненты недоступны

Таблица 2.7 (окончание)

Свойство	Описание
Visible	Позволяет скрыть (сделать невидимым) компонент GroupBox и все компоненты, которые находятся на его поверхности

ImageList

Компонент ImageList представляет собой коллекцию изображений и может использоваться другими компонентами (например, Button или ToolBar) как источник иллюстраций. Компонент является невизуальным, т. е. он не отображается в окне программы во время ее работы. Во время создания формы компонент отображается в нижней части окна редактора формы. Свойства компонента ImageList приведены в табл. 2.8.

Таблица 2.8. Свойства компонента ImageList

Свойство	Описание
Images	Коллекция изображений (объектов Bitmap)
ImageSize	Размер изображений коллекции. Уточняющее свойство Width определяет ширину изображений, Height — высоту
Transparent-Color	Прозрачный цвет. Участки изображения, окрашенные этим цветом, не отображаются
ColorDepht	Глубина цвета — число байтов, используемых для кодирования цвета точки (пиксела)

Label

Компонент Label предназначен для отображения текстовой информации. Задать текст, отображаемый в поле компонента, мож-

но как во время разработки формы, так и во время работы программы, присвоив нужное значение свойству `Text`. Свойства компонента приведены в табл. 2.9.

Таблица 2.9. *Свойства компонента* `Label`

Свойство	Описание
`Name`	Имя компонента. Используется в программе для доступа к свойствам компонента
`Text`	Отображаемый текст
`Location`	Положение компонента на поверхности формы
`AutoSize`	Признак автоматического изменения размера компонента. Если значение свойства равно `True`, то при изменении значения свойства `Text` (или `Font`) автоматически изменяется размер компонента
`Size`	Размер компонента (области отображения текста). Определяет (если значение свойства `AutoSize` равно `False`) размер компонента (области отображения текста)
`MaximumSize`	Если значение свойства `AutoSize` равно `True`, то задает максимально допустимый (макимально возможный) размер компонента (области отображения текста). Свойство `MaximumSize.Width` задает максимально возможную ширину области, свойство `MaximumSize.Height` — высоту
`Font`	Шрифт, используемый для отображения текста
`ForeColor`	Цвет текста, отображаемого в поле компонента
`BackColor`	Цвет закраски области вывода текста
`TextAlign`	Способ выравнивания (расположения) текста в поле компонента. Всего существует девять способов расположения текста. На практике наиболее часто используют выравнивание по левой верхней границе (`TopLeft`), посередине (`TopCentre`) и по центру (`MiddleCenter`)

Таблица 2.9 (окончание)

Свойство	Описание
BorderStyle	Вид рамки (границы) компонента. По умолчанию граница вокруг поля Label отсутствует (значение свойства равно None). Граница компонента может быть обычной (Fixed3D) или тонкой (FixedSingle)

ListBox

Компонент ListBox представляет собой список, в котором можно выбрать нужный элемент. Свойства компонента приведены в табл. 2.10.

Таблица 2.10. Свойства компонента ListBox

Свойство	Описание
Items	Элементы списка — коллекция строк
Items.Count	Количество элементов списка
SelectedIndex	Номер элемента, выбранного в списке. Если ни один из элементов списка не выбран, то значение свойства равно –1
Sorted	Признак необходимости автоматической сортировки (True) списка после добавления очередного элемента
SelectionMode	Определяет режим выбора элементов списка: One — только один элемент; MultiSimple — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка; MultiExtended — можно выбрать несколько элементов, сделав щелчок на нужных элементах списка при нажатой клавише <Ctrl>, или выделить диапазон, щелкнув при нажатой клавише <Shift> на первом и последнем элементе диапазона

Таблица 2.10 (окончание)

Свойство	Описание
MultiColumn	Признак необходимости отображать список в несколько колонок. Число отображаемых колонок зависит от количества элементов и размера области отображения списка
Location	Положение компонента на поверхности формы
Size	Размер компонента без учета (для компонентов типа DropDown и DropDownList) или с учетом (для компонента типа Simple) размера области списка или области ввода
Font	Шрифт, используемый для отображения содержимого поля редактирования и элементов списка

MenuStrip

Компонент MenuStrip представляет собой главное меню программы. И сами меню, и команды, образующие меню, — это объекты ToolStripMenuItem. Свойства объекта ToolStripMenuItem приведены в табл. 2.11.

Таблица 2.11. Свойства объекта ToolStripMenuItem

Свойство	Описание
Text	Название меню (команды)
Enabled	Признак доступности меню (команды). Если значение свойства равно False, то меню (или команда) недоступно
Image	Картинка, отображаемая рядом с названием меню или команды. В качестве картинки следует использовать png-иллюстрацию с прозрачным фоном

Таблица 2.11 (окончание)

Свойство	Описание
`Checked`	Признак того, что элемент меню выбран. Если значение свойства равно `True`, то элемент помечается галочкой (если для него не задана картинка). Свойство `Checked` обычно используется для тех элементов меню, которые применяются для отображения параметров
`Shortcut`	Свойство определяет функциональную клавишу (или комбинацию клавиш), нажатие которой активизирует выполнение команды
`ShowShortcut`	Если значение свойства равно `True` и задано значение свойства `Shortcut`, то после названия команды отображается название функциональной клавиши, нажатие которой активизирует команду

NotifyIcon

Компонент `NotifyIcon` представляет собой значок, который отображается в системной области панели задач. Обычно он используется для изображения программ, работающих в фоновом режиме. При позиционировании указателя мыши на значке, как правило, появляется подсказка, а в результате щелчка правой кнопки — контекстное меню, команды которого позволяют управлять работой программы. Свойства компонента приведены в табл. 2.12.

Таблица 2.12. Свойства компонента `NotifyIcon`

Свойство	Описание
`Icon`	Значок, который отображается на панели задач

Таблица 2.12 *(окончание)*

Свойство	Описание
`Text`	Подсказка (обычно название программы), которая отображается рядом с указателем мыши при позиционировании указателя на находящемся на панели задач значке
`ContextMenuStrip`	Ссылка на компонент `ContextMenuStrip`, обеспечивающий отображение контекстного меню
`Visible`	Свойство позволяет скрыть (`False`) значок (убрать с панели задач) или сделать его видимым

NumericUpDown

Компонент `NumericUpDown` предназначен для ввода числовых данных. Данные в поле редактирования можно ввести путем набора на клавиатуре или при помощи командных кнопок **Увеличить** и **Уменьшить**, которые находятся справа от поля редактирования. Свойства компонента `NumericUpDown` приведены в табл. 2.13.

Таблица 2.13. *Свойства компонента* `NumericUpDown`

Свойство	Описание
`Value`	Значение, соответствующее содержимому поля редактирования
`Maximum`	Максимально возможное значение, которое можно ввести в поле компонента
`Minimum`	Минимально возможное значение, которое можно ввести в поле компонента

Таблица 2.13 (окончание)

Свойство	Описание
`Increment`	Величина, на которую увеличивается или уменьшается значение свойства `Value` при каждом щелчке мышью на кнопках **Увеличить** или **Уменьшить**
`TextAlign`	Расположение текста в поле редактирования (`Left` — прижат к левому краю; `Center` — в центре; `Right` — прижат к правому краю)

OpenFileDialog

Компонент `OpenFileDialog` представляет собой стандартное диалоговое окно **Открыть**. Свойства компонента приведены в табл. 2.14.

Таблица 2.14. Свойства компонента `OpenFileDialog`

Свойство	Описание
`Title`	Текст в заголовке окна. Если значение свойства не задано, то в заголовке отображается текст **Открыть**
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов нужно отобразить только файлы с расширением txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

Таблица 2.14 (окончание)

Свойство	Описание
`FileNane`	Имя файла, введенное пользователем или выбранное в списке файлов
`InitialDirectory`	Каталог, содержимое которого отображается при появлении диалога на экране
`RestoreDirectory`	Признак необходимости отображать содержимое каталога, указанного в свойстве `InitialDirectory`, при каждом появлении окна. Если значение свойства равно `False`, то при следующем появлении окна на экране отображается содержимое каталога, выбранного пользователем в предыдущий раз

Panel

Компонент `Panel` представляет собой контейнер для других компонентов и позволяет легко управлять компонентами, которые находятся на панели. Например, для того чтобы сделать недоступными компоненты, находящиеся на панели, достаточно присвоить значение `False` свойству `Enabled` панели. Свойства компонента `Panel` приведены в табл. 2.15.

***Таблица 2.15.** Свойства компонента `Panel`*

Свойство	Описание
`Dock`	Определяет границу формы, к которой привязана (прикреплена) панель. Панель может быть прикреплена к левой (`Left`), правой (`Right`), верхней (`Top`) или нижней (`Bottom`) границе формы
`BorderStyle`	Вид границы панели: `FixedSingle` — рамка; `Fixed3D` — объемная граница; `None` — граница не отображается

Таблица 2.15 (окончание)

Свойство	Описание
Enabled	Свойство позволяет сделать недоступными (False) все компоненты, которые находятся на панели
Visible	Позволяет скрыть (False) панель

PictureBox

Компонент PictureBox обеспечивает отображение иллюстрации (рисунка, фотографии и т. п.). Свойства компонента приведены в табл. 2.16.

Таблица 2.16. Свойства компонента PictureBox

Свойство	Описание
Image	Иллюстрация, которая отображается в поле компонента
Size	Размер компонента. Уточняющее свойство Width определяет ширину компонента, Height — высоту
SizeMode	Метод отображения (масштабирования) иллюстрации, если ее размер не соответствует размеру компонента: Normal — масштабирование не выполняется (если размер компонента меньше размера иллюстрации, то отображается только часть иллюстрации); StretchImage — выполняется масштабирование иллюстрации так, чтобы она занимала всю область компонента (если размер компонента не пропорционален размеру иллюстрации, иллюстрация искажается);

Таблица 2.16 *(окончание)*

Свойство	Описание
	`AutoSize` — размер компонента автоматически изменяется в соответствии с размером иллюстрации; `CenterImage` — центрирование иллюстрации в поле компонента, если размер иллюстрации меньше размера компонента; `Zoom` — изменение размера иллюстрации таким образом, чтобы она занимала максимально возможную область компонента и при этом отображалась без искажения (с соблюдением пропорций)
`Image.PhysicalDimension`	Свойство содержит информацию об истинном размере картинки (иллюстрации), загруженной в поле компонента
`Location`	Положение компонента (области отображения иллюстрации) на поверхности формы. Уточняющее свойство `X` определяет расстояние от левой границы области до левой границы формы, уточняющее свойство `Y` — от верхней границы области до верхней границы клиентской области формы (нижней границы заголовка)
`Visible`	Признак указывает, отображается ли компонент и, соответственно, иллюстрация на поверхности формы
`BorderStyle`	Вид границы компонента: `None` — граница не отображается; `FixedSingle` — тонкая; `Fixed3D` — объемная

RadioButton

Компонент RadioButton представляет собой кнопку (переключатель), состояние которой зависит от состояния других кнопок (компонентов RadioButton). Обычно компоненты RadioButton объединяют в группу (достигается это путем размещения нескольких компонентов в поле компонента GroupBox). В каждый момент времени только одна из кнопок группы может находиться в выбранном состоянии (возможна ситуация, когда ни одна из кнопок не выбрана). Состояния компонентов, принадлежащих разным группам, независимы. Свойства компонента приведены в табл. 2.17.

Таблица 2.17. *Свойства компонента* `RadioButton`

Свойство	Описание
`Text`	Текст, который находится справа от кнопки
`Checked`	Состояние, внешний вид кнопки. Если кнопка выбрана, то значение свойства `Checked` равно `True`; если кнопка не выбрана, то значение свойства `Checked` равно `False`
`TextAllign`	Положение текста в поле отображения. Текст может располагаться в центре поля (`MiddleCenter`), прижат к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения текста надписи (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`CheckAllign`	Положение кнопки в поле компонента. Кнопка может быть прижата к левой верхней границе (`TopLeft`), прижата к левой границе и находиться на равном расстоянии от верхней и нижней границ поля компонента (`MiddleLeft`). Есть и другие варианты размещения кнопки в поле компонента

Таблица 2.17 *(окончание)*

Свойство	Описание
`Enabled`	Свойство позволяет сделать кнопку недоступной (`False`)
`Visible`	Свойство позволяет скрыть (`False`) кнопку
`AutoCheck`	Свойство определяет, должно ли автоматически изменяться состояние переключателя в результате щелчка на его изображении. По умолчанию значение равно `True`
`FlatStyle`	Стиль кнопки. Кнопка может быть обычной (`Standard`), плоской (`Flat`) или "всплывающей" (`Popup`). Стиль кнопки определяет ее поведение при позиционировании указателя мыши на изображении кнопки
`Appearance`	Определяет вид переключателя. Переключатель может выглядеть обычным образом (`Normal`) или как кнопка (`Button`)
`Image`	Картинка, которая отображается в поле компонента
`ImageAlign`	Положение картинки в поле компонента. Картинка может располагаться в центре (`MiddleCenter`), быть прижата к левой (`MiddleLeft`) или правой (`MiddleRight`) границе. Можно задать и другие способы размещения картинки на кнопке (`TopLeft`, `TopCenter`, `TopRight`, `BottomLeft`, `BottomCenter`, `BottomRight`)
`ImageList`	Набор картинок, используемых для обозначения различных состояний кнопки. Представляет собой объект типа `ImageList`. Чтобы задать значение свойства, в форму приложения нужно добавить компонент `ImageList`
`ImageIndex`	Номер (индекс) картинки из набора `ImageList`, которая отображается в поле компонента

ProgressBar

Компонент ProgressBar — это индикатор, который обычно используется для наглядного представления протекания процесса (например, обработки или копирования файлов, загрузки информации из сети и т. п.). Свойства компонента ProgressBar приведены в табл. 2.18. Следует обратить внимание, что выход значения свойства Value за границы диапазона, заданного свойствами Minimum и Maximum, вызывает исключение.

Таблица 2.18. *Свойства компонента* ProgressBar

Свойство	Описание
Value	Значение, которое отображается в поле компонента в виде полосы, длина которой пропорциональна значению свойства Value
Minimum	Минимально допустимое значение свойства Value
Maximum	Максимально допустимое значение свойства Value
Step	Приращение (шаг) изменения значения свойства Value при использовании метода PerformStep

SaveFileDialog

Компонент SaveFileDialog представляет собой стандартное диалоговое окно **Сохранить**. Свойства компонента приведены в табл. 2.19.

Таблица 2.19. *Свойства компонента* SaveFileDialog

Свойство	Описание
Title	Текст в заголовке окна. Если значение свойства не указано, то в заголовке отображается текст **Сохранить как**

Таблица 2.19 *(продолжение)*

Свойство	Описание
`FileName`	Полное имя файла, которое задал пользователь. В общем случае оно образуется из имени каталога, содержимое которого отображается в диалоговом окне, имени файла, которое пользователь ввел в поле **Имя файла**, и расширения, заданного значением свойства `DefaultExt`
`DefaultExt`	Расширение файла по умолчанию. Если пользователь в поле **Имя файла** не укажет расширение, то к имени файла будет добавлено расширение (при условии, что значение свойства `AddExtension` равно `True`), заданное значением этого свойства
`InitialDirectory`	Каталог, содержимое которого отображается при появлении диалога на экране
`RestoreDirectory`	Признак необходимости отображать содержимое каталога, указанного в свойстве `InitialDirectory`, при каждом появлении окна. Если значение свойства равно `False`, то при следующем появлении окна отображается содержимое каталога, выбранного пользователем в предыдущий раз
`CheckPathExists`	Признак необходимости проверки существования каталога, в котором следует сохранить файл. Если указанного каталога нет, то выводится информационное сообщение
`CheckFileExists`	Признак необходимости проверки существования файла с заданным именем. Если значение свойства равно `True` и файл с указанным именем уже существует, то появляется окно запроса, в котором пользователь может подтвердить необходимость замены (перезаписи) существующего файла

Таблица 2.19 *(окончание)*

Свойство	Описание
`Filter`	Свойство задает описание и фильтр (маску) имени файла. В списке файлов отображаются только те файлы, имена которых соответствуют указанной маске. Описание отображается в поле **Тип файла**. Например, значение **Текст|*.txt** указывает, что в списке файлов нужно отобразить только файлы с расширением txt
`FilterIndex`	Если фильтр состоит из нескольких элементов (например, **Текст|*.txt|Все файлы|*.***), то значение свойства задает фильтр, который используется в момент появления диалога на экране

TextBox

Компонент `TextBox` предназначен для ввода данных (строки символов) с клавиатуры. Свойства компонента приведены в табл. 2.20.

Таблица 2.20. *Свойства компонента* `TextBox`

Свойство	Описание
`Name`	Имя компонента. Используется для доступа к компоненту и его свойствам
`Text`	Текст, который находится в поле редактирования
`Location`	Положение компонента на поверхности формы
`Size`	Размер компонента
`Font`	Шрифт, используемый для отображения текста в поле компонента
`ForeColor`	Цвет текста, находящегося в поле компонента
`BackColor`	Цвет фона поля компонента

Таблица 2.20 *(продолжение)*

Свойство	Описание
BorderStyle	Вид рамки (границы) компонента. Граница компонента может быть обычной (Fixed3D), тонкой (FixedSingle) или отсутствовать (None)
TextAlign	Способ выравнивания текста в поле компонента. Текст в поле компонента может быть прижат к левой границе компонента (Left), правой (Right) или находиться по центру (Center)
MaxLength	Максимальное количество символов, которое можно ввести в поле компонента
PasswordChar	Символ, который используется для отображения вводимых пользователем символов (введенная пользователем строка находится в свойстве Text)
Multiline	Разрешает (True) или запрещает (False) ввод нескольких строк текста
ReadOnly	Разрешает (True) или запрещает (False) редактирование отображаемого текста
Dock	Способ привязки положения и размера компонента к размеру формы. По умолчанию привязка отсутствует (None). Если значение свойства равно Top или Bottom. Ширина компонента устанавливается равной ширине формы и компонент прижимается соответственно к верхней или нижней границе формы. Если значение свойства Dock равно Fill, а свойства Multiline — True, то размер компонента устанавливается максимально возможным
Lines	Массив строк, элементы которого содержат текст, находящийся в поле редактирования, если компонент находится в режиме MultiLine. Доступ к строке осуществляется по номеру. Строки нумеруются с нуля

Таблица 2.20 (окончание)

Свойство	Описание
`ScrollBars`	Задает отображаемые полосы прокрутки: `Horizontal` — горизонтальная; `Vertical` — вертикальная; `Both` — горизонтальная и вертикальная; `None` — не отображать

ToolTip

Компонент `ToolTip` — вспомогательный, он используется другими компонентами формы для вывода подсказок при позиционировании указателя мыши на компоненте. Свойства компонента приведены в табл. 2.21.

Таблица 2.21. Свойства компонента `ToolTip`

Свойство	Описание
`Active`	Разрешает (`True`) или запрещает (`False`) отображение подсказок
`AutoPopDelay`	Время отображения подсказки
`InitialDelay`	Время, в течение которого указатель мыши должен быть неподвижным, чтобы появилась подсказка
`ReshowDelay`	Время задержки отображения подсказки после перемещения указателя мыши с одного компонента на другой

Timer

Компонент `Timer` генерирует последовательность событий `Tick`. Свойства компонента приведены в табл. 2.22.

Таблица 2.22. *Свойства компонента* `Timer`

Свойство	Описание
`Interval`	Период генерации события `Tick`. Задается в миллисекундах
`Enabled`	Разрешение работы. Разрешает (значение `True`) или запрещает (значение `False`) генерацию события `Tick`

Графика

Графические примитивы

Вычерчивание графических примитивов (линий, прямоугольников, окружностей, дуг, секторов) на графической поверхности выполняют соответствующие методы объекта `Graphics` (табл. 2.23).

Таблица 2.23. *Некоторые методы вычерчивания графических примитивов*

Метод	Действие
`DrawLine(Pen, x1, y1, x2, y2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `x1`, `y1`, `x2`, `y2` — координаты точек начала и конца линии
`DrawLine(Pen, p1, p2)`	Рисует линию. Параметр `Pen` определяет цвет, толщину и стиль линии; параметры `p1` и `p2` (структуры `Point`) точки начала и конца линии
`DrawRectangle(Pen, x, y, w, h)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника

Таблица 2.23 *(продолжение)*

Метод	Действие
`DrawRectangle(Pen, rect)`	Рисует контур прямоугольника. Параметр `Pen` определяет цвет, толщину и стиль границы прямоугольника: параметр `rect` (структура `Rectangle`) область, граница которой определяет контур прямоугольника
`FillRectangle(Brush, x, y, w, h)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметры `x`, `y` — координаты левого верхнего угла; параметры `w` и `h` задают размер прямоугольника
`FillRectangle(Brush, rect)`	Рисует закрашенный прямоугольник. Параметр `Brush` определяет цвет и стиль закраски прямоугольника; параметр `rect` (структура `Rectangle`) — закрашиваемую область
`DrawEllipse(Pen, x, y, w, h)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметры `x`, `y`, `w`, `h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс
`DrawEllipse(Pen, rect)`	Рисует эллипс (контур). Параметр `Pen` определяет цвет, толщину и стиль линии эллипса; параметр `rect` — область, внутри которой рисуется эллипс
`FillEllipse(Brush, x, y, w, h)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметры `x`, `y`, `w`, `h` — координаты левого верхнего угла и размер прямоугольника, внутри которого вычерчивается эллипс

Таблица 2.23 (окончание)

Метод	Действие
`FillEllipse(Brush, x, y, rect)`	Рисует закрашенный эллипс. Параметр `Brush` определяет цвет и стиль закраски внутренней области эллипса; параметр `rect` — область, внутри которой вычерчивается эллипс
`DrawPolygon(Pen, P)`	Рисует контур многоугольника. Параметр `Pen` определяет цвет, толщину и стиль линии границы многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника
`FillPolygon(Brush, P)`	Рисует закрашенный многоугольник. Параметр `Brush` определяет цвет и стиль закраски внутренней области многоугольника; параметр `P` (массив типа `Point`) — координаты углов многоугольника
`DrawString(str, Font, Brush, x, y)`	Выводит на графическую поверхность строку текста. Параметр `Font` определяет шрифт; `Brush` — цвет символов; `x` и `y` — точку, от которой будет выведен текст
`DrawImage(Image, x, y)`	Выводит на графическую поверхность иллюстрацию. Параметр `Image` определяет иллюстрацию; `x` и `y` — координату левого верхнего угла области вывода иллюстрации

Карандаш

Карандаш определяет вид линии — цвет, толщину и стиль. В распоряжении программиста есть набор цветных карандашей (всего

их 141), при помощи которых можно рисовать сплошные линии толщиной в *один пиксел* (табл. 2.24).

Таблица 2.24. *Некоторые карандаши из стандартного набора*

Карандаш	Цвет
`Pens.Red`	Красный
`Pens.Orange`	Оранжевый
`Pens.Yellow`	Желтый
`Pens.Green`	Зеленый
`Pens.LightBlue`	Голубой
`Pens.Blue`	Синий
`Pens.Purple`	Фиолетовый
`Pens.Black`	Черный
`Pens.LightGray`	Серый
`Pens.White`	Белый
`Pens.Transparent`	Прозрачный

Программист может создать свой собственный карандаш, объект типа `Pen`, и, задав значения свойств (табл. 2.25), определить цвет, толщину и стиль линии, которую он будет рисовать.

Таблица 2.25. *Свойства объекта* `Pen`

Свойство	Описание
`Color`	Цвет линии. В качестве значения свойства следует использовать одну из констант `Color` (например, `Color.Red`), определенных в пространстве имен `System.Drawing`

Таблица 2.25 *(окончание)*

Свойство	Описание
`Width`	Толщина линии (задается в пикселах)
`DashStyle`	Вид линии. В качестве значения свойства следует использовать одну из констант `DashStyle`, определенных в пространстве имен `System.Drawing.Drawing2D`): `DashStyle.Solid` — сплошная; `DashStyle.Dash` — пунктирная, длинные штрихи; `DashStyle.Dot` — пунктирная, короткие штрихи; `DashStyle.DashDot` — пунктирная, чередование длинного и короткого штрихов; `DashStyle.DashDotDot` — пунктирная, чередование одного длинного и двух коротких штрихов; `DashStyle.Custom` — пунктирная линия, вид которой определяет значение свойства `DashPattern`
`DashPattern`	Длина штрихов и промежутков пунктирной линии `DashStyle.Custom`

Кисть

Кисти используются для закраски внутренних областей геометрических фигур. В распоряжении программиста есть четыре типа кистей: стандартные (`Brush`), штриховые (`HatchBrush`), градиентные (`LinearGradientBrush`) и текстурные (`TextureBrush`).

Стандартная кисть закрашивает область одним цветом (сплошная закраска). В стандартном наборе более 100 кистей, некоторые из них приведены в табл. 2.26.

Таблица 2.26. *Некоторые кисти из стандартного набора*

Кисть	Цвет
`Brushes.Red`	Красный
`Brushes.Orange`	Оранжевый
`Brushes.Yellow`	Желтый
`Brushes.Green`	Зеленый
`Brushes.LightBlue`	Голубой
`Brushes.Blue`	Синий
`Brushes.Purple`	Фиолетовый
`Brushes.Black`	Черный
`Brushes.LightGray`	Серый
`Brushes.White`	Белый
`Brushes.Transparent`	Прозрачный

Штриховая кисть (`HatchBrush`) закрашивает область путем штриховки. Область может быть заштрихована горизонтальными, вертикальными или наклонными линиями разного стиля и толщины. В табл. 2.27 перечислены некоторые из возможных стилей штриховки. Полный список стилей штриховки можно найти в справочной системе.

Таблица 2.27. *Некоторые стили штриховки областей*

Стиль	Штриховка
`HatchStyle.LightHorizontal`	Редкая горизонтальная
`HatchStyle.Horizontal`	Средняя горизонтальная
`HatchStyle.NarrowHorizontal`	Частая горизонтальная
`HatchStyle.LightVertical`	Редкая вертикальная

Таблица 2.27 *(окончание)*

Стиль	Штриховка
`HatchStyle.Vertical`	Средняя вертикальная
`HatchStyle.NarrowVertical`	Частая вертикальная
`HatchStyle.LageGrid`	Крупная сетка из горизонтальных и вертикальных линий
`HatchStyle.SmallGrid`	Мелкая сетка из горизонтальных и вертикальных линий
`HatchStyle.DottedGrid`	Сетка из горизонтальных и вертикальных линий, составленных из точек
`HatchStyle.ForwardDiagonal`	Диагональная штриховка "вперед"
`HatchStyle.BackwardDiagonal`	Диагональная штриховка "назад"
`HatchStyle.Percent05` – `HatchStyle.Percent90`	Точки (степень заполнения 5%, 10%, ..., 90%)
`HatchStyle.HorizontalBrick`	"Кирпичная стена"
`HatchStyle.LargeCheckerBoard`	"Шахматная доска"
`HatchStyle.SolidDiamond`	"Бриллиант" ("Шахматная доска", повернутая на 45°)
`HatchStyle.Sphere`	"Пузырьки"
`HatchStyle.ZigZag`	"Зигзаг"

Градиентная кисть (`LinearGradientBrush`) представляет собой прямоугольную область, цвет точек которой зависит от расстояния до границы. Обычно градиентные кисти двухцветные, т. е.

цвет точек по мере удаления от границы постепенно меняется с одного на другой. Цвет может меняться вдоль горизонтальной или вертикальной границы области. Возможно также изменение цвета вдоль линии, образующей угол с горизонтальной границей.

Текстурная кисть (`TextureBrush`) представляет собой рисунок, который обычно загружается во время работы программы из файла (bmp, jpg или gif) или из ресурса. Закраска текстурной кистью выполняется путем дублирования рисунка внутри области.

Типы данных

Целый тип

Целые типы приведены в табл. 2.28.

Таблица 2.28. *Основные целые типы*

Тип	Диапазон	Формат
`System.SByte`	–128..127	8 битов, со знаком
`System.Int16`	–32768..32767	16 битов, со знаком
`System.Int32`	–2 147 483 648.. 2 147 483 647	32 бита, со знаком
`System.Int64`	$-2^{63}..2^{63}$	64 бита, со знаком
`System.Byte`	0..255	8 битов, без знака
`System.UInt16`	0..65535	16 битов, без знака
`System.UInt32`	0..4294967295	32 бита, без знака

Вещественный тип

Вещественные типы приведены в табл. 2.29.

Таблица 2.29. *Основные вещественные типы*

Тип	Диапазон	Значащих цифр	Байтов
`System.Single`	$-1.5 \cdot 10^{45} .. 3.4 \cdot 10^{38}$	7—8	4
`System.Double`	$-5.0 \cdot 10^{324} .. 1.7 \cdot 10^{308}$	15—16	8

Символьный и строковый типы

Основной символьный тип — `System.Char`.

Основной строковый тип — `System.String`.

Функции

В этом разделе описаны некоторые наиболее часто используемые функции.

Функции преобразования

В табл. 2.30 приведены функции, обеспечивающие преобразование строки символов в число, изображением которого является строка. Если строка не может быть преобразована в число (например, из-за того что содержит недопустимый символ), то возникает исключение типа `FormatException`. Функции преобразования принадлежат пространству имен `System.Convert`.

Таблица 2.30. *Функции преобразования строки в число*

Функция	Значение
`ToSingle(s)` `ToDouble`	Дробное типа `Single`, `Double`

`ToByte(s)`
`ToInt16(s)`
`ToInt32(s)`
`ToInt64(s)`
	Целое типа `Byte`, `Int16`, `Int32`, `Int64`

`ToUInt16(s)`
`ToUInt32(s)`
`ToUInt64(s)`
	Целое типа `UInt16`, `UInt32`, `UInt64`

Преобразование числа в строку символов обеспечивает метод (функция) `ToString`. В качестве параметра функции `ToString` можно указать символьную константу (табл. 2.31), которая задает формат строки-результата.

Таблица 2.31. *Форматы отображения чисел*

Константа	Формат	Пример
`c`, `C`	`Currency` — финансовый (денежный). Используется для представления денежных величин. Обозначение денежной единицы, разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28 р.
`e`, `E`	`Scientific (exponential)` — научный. Используется для представления очень маленьких или очень больших чисел. Разделитель целой и дробной частей числа задается в настройках операционной системы	5,50528+E004

Таблица 2.31 *(окончание)*

Константа	Формат	Пример
`f, F`	`Fixed` — число с фиксированной точкой. Используется для представления дробных чисел. Количество цифр дробной части, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
`n, N`	`Number` — числовой. Используется для представления дробных чисел. Количество цифр дробной части, символ-разделитель групп разрядов, способ отображения отрицательных чисел определяют соответствующие настройки операционной системы	55 055,28
`g, G`	`General` — универсальный формат. Похож на `Number`, но разряды не разделены на группы	55 055,275
`r, R`	`Roundtrip` — без округления. В отличие от формата `N`, этот формат не выполняет округления (количество цифр дробной части зависит от значения числа)	55 055,2775

Функции манипулирования строками

Некоторые функции (методы) манипулирования строками приведены в табл. 2.32.

Таблица 2.32. *Функции манипулирования строками*

Функция (метод)	Значение
`s.Length`	Длина (количество) символов строки. Строго говоря, `Length` — это не метод, а свойство объекта `String`

Таблица 2.32 *(продолжение)*

Функция (метод)	Значение
`s.IndexOf(c)`	Положение (номер) символа `c` в строке `s` (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
`s.LastIndexOf(c)`	Положение (от конца) символа `c` в строке `s` (символы строки нумеруются с нуля). Если указанного символа в строке нет, то значение функции равно минус один
`s.IndexOf(st)`	Положение подстроки `st` в строке `s` (символы строки `s` нумеруются с нуля). Если указанной подстроки в строке нет, то значение функции равно минус один
`s.Trim`	Строка, полученная из строки `s` путем "отбрасывания" пробелов, находящихся в начале и в конце строки
`s.Substring(n)`	Подстрока, выделенная из строки `s`, начиная с символа с номером `n` (символы строки `s` нумеруются с нуля). Если значение `n` больше, чем количество символов в строке, то возникает исключение `ArgumentOutOfRangeException`
`s.Substring(n,k)`	Подстрока длиной `k` символов, выделенная из строки `s`, начиная с символа с номером `n` (символы строки `s` нумеруются с нуля). Если значение `n` больше, чем количество символов в строке или если `k` больше, чем `len - n` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`s.Insert(pos,st)`	Строка, полученная путем вставки в строку `s` строки `st`. Параметр `pos` задает номер символа строки `s`, после которого вставляется строка `st`

Таблица 2.32 (окончание)

Функция (метод)	Значение
`s.Remove(pos,n)`	Строка, полученная путем удаления из строки `s` цепочки символов (подстроки). Параметр `pos` задает положение подстроки, параметр `n` — количество символов, которое нужно удалить. Если значение `pos` больше, чем количество символов в строке или если `n` больше, чем `len - pos` (где `len` — длина строки `s`), то возникает исключение `ArgumentOutOfRangeException`
`s.Replace(old,new)`	Строка, полученная из строки `s` путем замены всех символов `old` на символ `new`
`s.ToUpper()`	Строка, полученная из строки `s` путем замены строчных символов на прописные
`s.ToLower()`	Строка, полученная из строки `s` путем замены прописных символов на строчные
`s.Split(sep)`	Массив строк, полученный разбиением строки `s` на подстроки. Параметр `sep` (массив типа `Char`) задает символы, которые используются методом `Split` для обнаружения границ подстрок

Функции манипулирования датами и временем

Некоторые функции манипулирования датами приведены в табл. 2.33. (`d` — некоторый объект типа `DataTime`).

Таблица 2.33. Функции манипулирования датами и временем

Функция (метод)	Значение
`DateTime.Now`	Структура типа `DateTime`. Текущие дата и время

Таблица 2.33 (продолжение)

Функция (метод)	Значение
`DateTime.Today`	Структура типа `DateTime`. Текущая дата
`d.ToString()`	Строка вида dd.mm.yyyy mm:hh:ss (например, 05.06.2006 10:00)
`d.ToString(f)`	Строка — дата и/или время. Вид строки определяет параметр `f`. Полному формату даты и времени соответствует строка `dd.MM.yyyy hh:mm:ss`
`d.Day`	День (номер дня месяца)
`d.Month`	Номер месяца (1 — январь, 2 — февраль и т. д.)
`d.Year`	Год
`d.Hour`	Час
`d.Minute`	Минуты
`d.DayOfYear`	Номер дня года (отсчет от 1 января)
`d.DayOfWeek`	День недели
`d.ToLongDateString`	"Длинная" дата. Например: 5 июня 2009
`d.ToShortDateString`	"Короткая" дата. Например: 05.06.2009
`d.ToLongTimeString`	Время в формате hh:mm:ss
`d.ToShotrTimeString`	Время в формате hh:mm
`d.AddDays(n)`	Дата, отстоящая от `d` на `n` дней. Положительное `n` "сдвигает" дату вперед, отрицательное — назад
`d.AddMonths(n)`	Дата, отстоящая от `d` на `n` месяцев. Положительное `n` "сдвигает" дату вперед, отрицательное — назад

Таблица 2.33 (окончание)

Функция (метод)	Значение
d.AddYears(n)	Дата, отстоящая от d на n лет. Положительное n "сдвигает" дату вперед, отрицательное — назад
d.AddHours(n)	Дата (время), отстоящая от d на n часов. Положительное n "сдвигает" дату вперед, отрицательное — назад
d.Minutes(т)	Дата (время), отстоящая от d на n минут. Положительное n "сдвигает" дату вперед, отрицательное — назад

Функции манипулирования каталогами и файлами

Функции (методы) манипулирования каталогами и файлами (табл. 2.34) принадлежат пространству имен `System.IO`. При выполнении операций с каталогами и файлами возможны исключения (`di` — объект типа `DirectoryInfo`, `fi` — объект типа `FileInfo`, `sr` — объект типа `StreamReader`, `sw` — объект типа `StreamWriter`).

Таблица 2.34. Функции (методы) манипулирования каталогами и файлами

Функция (метод)	Результат (значение)
DirectoryInfo(Path)	Создает объект типа DirectoryInfo, соответствующий каталогу, заданному параметром Path
di.GetFiles(fn)	Формирует список файлов каталога — массив объектов типа FileInfo.

Таблица 2.34 *(продолжение)*

Функция (метод)	Результат (значение)
	Каждый элемент массива соответствует файлу каталога, заданного объектом `di` (тип `DirectoryInfo`). Параметр `fn` задает критерий отбора файлов
`di.Exists`	Проверяет, существует ли в системе каталог. Если каталог существует, то значение функции равно `True`, в противном случае — `False`
`di.Create (dn)`	Создает каталог. Если путь к новому каталогу указан неправильно, возникает исключение
`fi.CopyTo(Path)`	Копирует файл, заданный объектом `fi` (тип `FileInfo`), в каталог и под именем, заданным параметром `Path`. Значение метода — объект типа `FileInfo`, соответствующий файлу, созданному в результате копирования
`fi.MoveTo(Path)`	Перемещает файл, заданный объектом `fi` (тип `FileInfo`), в каталог и под именем, заданным параметром `Path`
`fi.Delete()`	Уничтожает файл, соответствующий объекту `fi` (тип `FileInfo`)
`StreamReader (fn)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`. Поток открывается для чтения в формате UTF-8
`StreamReader (fn,encd)`	Создает и открывает для чтения поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremReader`.

Таблица 2.34 *(продолжение)*

Функция (метод)	Результат (значение)
	Поток открывается для чтения в кодировке, заданной параметром `encd` (объект типа `System.Text.Encoding`). Для чтения текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System.Text.Encoding.GetEncoding(1251)`
`sr.Read()`	Читает символ из потока `sr` (объект типа `StremReader`). Значение метода — код символа
`sr.Read(buf, p, n)`	Читает из потока `sr` (объект типа `StremReader`) символы и записывает их в массив символов `buf` (тип `Char`). Параметр `p` задает номер элемента массива, в который будет помещен первый прочитанный символ, параметр `n` — количество символов, которое нужно прочитать
`sr.ReadToEnd()`	Читает весь текст из потока `sr` (объект типа `StremReader`). Значение метода — прочитанный текст
`sr.ReadLine()`	Читает строку из потока `sr` (объект типа `StremReader`). Значение метода — прочитанная строка
`StreamWriter (fn)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`. Поток открывается для записи в формате (кодировке) UTF-8
`StreamWriter (fn,a,encd)`	Создает и открывает для записи поток, соответствующий файлу, заданному параметром `fn`. Значение метода — объект типа `StremWriter`.

Таблица 2.34 *(продолжение)*

Функция (метод)	Результат (значение)
	Поток открывается для записи в кодировке, заданной параметром `encd` (объект типа `System.Text.Encoding`). Для записи текста в кодировке Windows 1251 параметр `encd` необходимо инициализировать значением `System.Text.Encoding.GetEncoding(1251)`. Параметр `a` задает режим записи: `True` — добавление в файл, `False` — перезапись
`sw.Write(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`. В качестве `v` можно использовать выражение строкового или числового типа
`sw.WriteLine(v)`	Записывает в поток `sw` (объект типа `StreamWriter`) строку символов, соответствующую значению `v`, и символ "новая строка". В качестве `v` можно использовать выражение строкового или числового типа
`sw.WriteLine`	Записывает в поток `sw` (объект типа `StreamWriter`) символ "новая строка"
`s.Close`	Закрывает поток `s`

Математические функции

Некоторые математические функции, принадлежащие пространству имен `Math`, приведены в табл. 2.35.

Таблица 2.35. *Математические функции*

Функция (метод)	Значение
`Abs(n)`	Абсолютное значение `n`
`Sign(n)`	1 (если `n` — положительное) –1 (если `n` — отрицательное)
`Round(n,m)`	Дробное, полученное путем округления `n` по известным правилам. Параметр `m` задает количество цифр дробной части
`Ceiling(n)`	Дробное, полученное путем округления `n` "с избытком"
`Floor(n)`	Дробное, полученное путем округления `n` "с недостатком" (отбрасыванием дробной части)
`Log(n)`	Натуральный логарифм `n` (логарифм `n` по основанию `E`, где `E` — постоянная, значение которой равно 2.7182818284590452354)
`Log(n,m)`	Логарифм `n` по основанию `m`
`Log10(n)`	Десятичный логарифм `n`
`Exp(n)`	Экспонента `n` (число "`E`" в степени `n`)
`Cos(α)`	Косинус угла α, заданного в радианах
`Sin(α)`	Синус угла α, заданного в радианах
`Tan(α)`	Тангенс угла α, заданного в радианах
`ASin(n)`	Арксинус `n` — угол (в радианах), синус которого равен `n`
`ACos(n)`	Арккосинус `n` — угол (в радианах), косинус которого равен `n`
`ATan(n)`	Арктангенс `n` — угол (в радианах), тангенс которого равен `n`
`Sqrt(n)`	Квадратный корень из `n`
`r.Next(hb)`	Случайное число из диапазона 0 .. (`hb-1`). `r` — объект типа `System.Random`

Величина угла тригонометрических функций должна быть задана в радианах. Чтобы преобразовать величину угла из градусов в радианы, нужно значение, выраженное в градусах, умножить на $\pi/180$, где π – число "Пи". Числовую константу 3.1415926 (значение числа "Пи") можно заменить именованной константой `PI`.

События

В табл. 2.36 приведено краткое описание некоторых событий.

Таблица 2.36. *События*

Событие	Происходит
`Click`	При щелчке кнопкой мыши
`Closed`	Возникает сразу за событием `Clossing`
`Closing`	Возникает как результат щелчка на системной кнопке **Закрыть окно**
`DblClick`	При двойном щелчке кнопкой мыши
`Enter`	При получении элементом управления фокуса
`KeyDown`	При нажатии клавиши клавиатуры. Сразу за событием `KeyDown` возникает событие `KeyPress`. Если нажатая клавиша удерживается, то событие `KeyDown` возникает еще раз. Таким образом, пара событий `KeyDown – KeyPress` генерируется до тех пор, пока не будет отпущена удерживаемая клавиша (в этот момент происходит событие `KeyUp`)
`KeyPress`	При нажатии клавиши клавиатуры. Событие `KeyPress` возникает сразу после события `KeyDown`
`KeyUp`	При отпускании нажатой клавиши клавиатуры
`Leave`	При потере элементом управления фокуса

Таблица 2.36 (окончание)

Событие	Происходит
`Load`	В момент загрузки формы. Используется для инициализации программы
`MouseDown`	При нажатии кнопки мыши
`MouseMove`	При перемещении мыши
`MouseUp`	При отпускании кнопки мыши
`Paint`	При появлении окна (элемента управления) на экране в начале работы программы, после появления части окна, которая, например, была закрыта другим окном, и в других случаях. Только процедура обработки события `Paint` имеет доступ к свойству `Graphics`, методы которого обеспечивают отображение графики
`Resize`	При изменении размера окна. Если размер формы увеличивается, то сначала происходит событие `Paint`, затем — `Resize`. Если размер формы уменьшается, то происходит только событие `Resize`

Исключения

В табл. 2.37 перечислены наиболее часто возникающие исключения и указаны причины, которые могут привести к их возникновению.

Таблица 2.37. Исключения

Исключение	Возникает
`FormatException` — ошибка преобразования	При выполнении преобразования, если преобразуемая величина не может быть приведена к требуемому типу. Наиболее часто возникает при преобразовании строки символов в число

Таблица 2.37 *(окончание)*

Исключение	Возникает
`IndexOutOfRangeException` — выход значения индекса за допустимые границы	При обращении к несуществующему элементу массива
`ArgumentOutOfRangeException` — выход значения аргумента за допустимые границы	При обращении к несуществующему элементу данных, например, при выполнении операций со строками
`OverflowException` — переполнение	Если результат выполнения операции выходит за границы допустимого диапазона, а также при выполнении операции деления, если делитель равен нулю
`FileNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого файла (ошибка в имени файла или обращение к несуществующему или недоступному устройству)
`DirectoryNotFoundException` — ошибка ввода/вывода	При выполнении файловых операций. Наиболее частая причина — отсутствие требуемого каталога (ошибка в имени каталога или обращение к несуществующему или недоступному устройству)

Задачи для самостоятельного решения

1. Напишите программу, при помощи которой можно посчитать сопротивление электрической цепи, состоящей из двух резисторов, которые могут быть соединены последовательно или параллельно. Если сопротивление цепи меньше 1000 Ом, то результат отображать в омах, иначе — в килоомах.
2. Напишите программу, при помощи которой можно рассчитать доход по вкладу. Исходные данные для расчета — сумма и срок вклада (1, 3, 6 или 12 месяцев). Величина процентной ставки определяется сроком вклада.
3. Напишите программу, при помощи которой можно подсчитать цену бензина на автозаправочной станции. Исходные данные для расчета — число литров, марка бензина (92, 95 или 98) и наличие дисконтной карты.
4. Напишите программу, при помощи которой можно пересчитать цену из долларов в рубли или из рублей в доллары.
5. Напишите программу, при помощи которой можно определить стоимость аренды автомобиля. Исходные данные для расчета — время аренды (целое число часов) и тип автомобиля (такси, микроавтобус или автобус).
6. Напишите программу, при помощи которой можно рассчитать платеж КАСКО. Исходные данные для расчета — цена автомобиля и его марка (Ford, BMW, Toyota, Renault, Volkswagen).
7. Напишите программу, при помощи которой можно подсчитать расходы на доставку мебели. Исходные данные для расчета — номер этажа и информация о наличии и типе лифта (грузовой или обычный).

8. Напишите программу, при помощи которой можно посчитать цену стеклопакета. Исходные данные для расчета — габаритные размеры (ширина и высота в миллиметрах), тип механизма открывания (поворотный или поворотно-откидной) и наличие дополнительных опций (фиксатор, микропроветривание, москитная сетка).

9. Напишите программу, при помощи которой можно посчитать стоимость заказа печати фотографий. Исходные данные для расчета — размер (9×12, 12×15 или 18×24) и количество фотографий. Если количество фотографий больше 20, то предоставляется скидка 10%.

10. Напишите программу, при помощи которой можно пересчитать расстояния из миль в километры. Исходные данные для расчета — расстояние в километрах и тип мили (морская или сухопутная).

11. Напишите программу, при помощи которой можно пересчитать температуру из градусов Цельсия в градусы Фаренгейта или Кельвина.

12. Напишите программу, при помощи которой можно рассчитать цену аренды такси. Исходные данные для расчета — расстояние поездки и регион (в черте города, в радиусе до 70 км, в радиусе более 70 км).

13. Напишите программу, при помощи которой можно определить затраты на грузоперевозки. Исходные данные для расчета — расстояние и информация о предоставлении грузчиков (с грузчиками, без грузчиков).

14. Напишите программу, при помощи которой можно вычислить стоимость тиражирования материалов в типографии. Исходные данные для расчета — количество копий, формат (A5, A4, A2, A1, A0) и условие выполнения заказа (в присутствии заказчика или на следующий день).

15. Напишите программу, в окне которой отображается график изменения температуры воздуха за месяц. Предполагается,

что возможны как положительные, так и отрицательные значения температуры.

16. Напишите программу, в окне которой отображается столбчатая диаграмма изменения температуры воздуха за месяц. Положительные температуры отображать красными столбиками, отрицательные — синими.

17. Напишите программу, в окне которой отображается график изменения цены бензина (92, 95, 98) за последние шесть месяцев.

18. Напишите программу, в окне которой отображается график изменения цены бензина одной марки (например, 95) у разных операторов (Лукойл, Shell, Neste) за последние шесть месяцев.

19. Напишите программу, в окне которой отображается в виде столбчатой диаграммы результат социологического опроса (5 вопросов). Вопрос отобразить в заголовке диаграммы. Под каждым столбиком печатать ответ.

20. Напишите программу, в окне которой отображается в виде "горизонтальной" столбчатой диаграммы результат социологического опроса (5 вопросов).

21. Напишите программу, в окне которой отображается столбчатая диаграмма — результат сдачи экзамена (процент "5", "4", "3" и "2"). Исходные данные — количество "пятерок", "четверок", "троек" и "двоек".

22. Напишите программу, в окне которой отображается в виде графика число посещений сайта. Исходные данные — ежедневные значения счетчика посещений (нарастающим итогом).

23. Напишите программу, в окне которой отображается в виде столбчатой диаграммы информация о продажах, например, книги. Исходные данные — информация об остатках на складе (на конец месяца).

24. Напишите программу, в окне которой в виде круговой диаграммы отображается информация о расходах среднестатистической семьи на питание, транспорт, одежду, образование.

25. Напишите программу, в окне которой в виде полигона отображается информация об изменении стоимости акций какой-либо компании (например, ОАО "Газпром") за последний месяц.

26. Напишите программу, в окне которой отображается информация о средней цене бензина (например, 95) в разных городах страны, например, в Санкт-Петербурге, Москве, Ростове, Новосибирске и Хабаровске.

27. Усовершенствуйте программу "Калькулятор" так, чтобы можно было выполнять операции умножения и деления.

28. Внесите изменения в программу "Сапер", чтобы изображения клеток (пустая клетка; флажок; мина; мина, помеченная флажком) загружались из файла.

29. Усовершенствуйте программу "Экзаменатор" так, чтобы результат тестирования сохранялся в файле. В начале работы программа должна запрашивать имя испытуемого, а в конце — записывать результат в файл.

30. Напишите программу, обеспечивающую работу с базой данных Microsoft Access "Расходы". В базе данных должны фиксироваться сумма, дата и на что потрачены деньги (по категориям, например, еда, транспорт, образование, развлечения, прочее). Программа должна обеспечивать фильтрацию данных по содержимому поля "Категория", а также выполнять статистическую обработку — выводить сумму затрат за период.

Приложение

Описание компакт-диска

Прилагаемый компакт-диск содержит проекты, приведенные в книге. Каждый проект находится в отдельном каталоге. Для того чтобы увидеть, как работает приложение, загрузите проект в Microsoft Visual C#, откомпилируйте его и затем запустите. Следует обратить внимание, что некоторые программы, например, для работы с базами данных, требуют, чтобы на компьютере был установлен соответствующий сервер.

Для активной работы, чтобы иметь возможность вносить изменения в программы, скопируйте каталоги проектов на жесткий диск компьютера, в папку проектов Microsoft Visual Sudio.

Предметный указатель

П

С

Т

Ф

Ц

Ч

Ш

Книги издательства "БХВ-Петербург" в продаже:

www.bhv.ru

Магазин "Новая техническая книга": СПб., Измайловский пр., д. 29,

тел.: (812) 251-41-10

www.techkniga.com

Отдел оптовых поставок: e-mail: opt@bhv.spb.su

Серия "Самоучитель"

Пекарев Л.Д.	3ds max 7	432 с.
Кирьянов Д.В.	Adobe After Effects 6.0	368 с.
Бурлаков М.	Adobe Illustrator CS	736 с.
Федорова А.	Adobe PageMaker 7.0.	736 с.
Тайц А.	Adobe Photoshop 7 (+ CD)	688 с.
Кирьянов Д.В.	Adobe Premiere 6.5	480 с.
Кирьянов Д.В.	Adobe Premiere Pro	448 с.
Ломов А.Ю.	Apache, Perl, MySQL: практика создания динамических сайтов. Самоучитель (+ CD)	368 с.
Полещук Н.Н.	AutoCAD 2004	640 с.
Полещук Н.Н.	AutoCAD 2007	624 с.
Трасковский А.В.	BIOS. Самоучитель	400 с.
Гросс К.	C# 2008	576 с.
Шилдт Г.	C++ (+ CD) — 3 издание	688 с.
Культин Н.	C++ Builder (+ CD)	320 с.
Культин Н.Б.	C++ Builder.-2-е изд., перераб. и доп. (+ CD)	464 с.
Понамарев В.	Delphi 7 Studio	512 с.
Ломов А.Ю.	HTML, CSS, скрипты. Практика создания сайтов (+ CD)	416 с.
Понамарев В.	JBuilder 6/7	304 с.
Костромин В.	Linux для пользователя	672 с.
Поляк-Брагинский А.В.	Linux и Windows в домашней сети	336 с.
Колисниченко Д.Н.	Linux на ноутбуке (+DVD)	704 с.
Дронов В.А.	Macromedia Dreamweaver 8	320 с.
Альберт Д.И.	Macromedia Flash Professional 8	736 с.
Кирьянов Д.В.	Mathcad 13	528 с.
Ануфриев И.	MatLab 5.3/6.x (+ прил. на дискете)	736 с.

Куперштейн В.И.	Microsoft Project 2007 в управлении проектами (+ CD)	560 с.
Жилинский А.	Microsoft SQL Server 2005	224 с.
Омельченко Л.	Microsoft Windows XP	560 с.
Хомоненко А.	Microsoft Word 2003	672 с.
Клименко Б.И.	Microsoft Word. Комфортная работа с помощью макросов	496 с.
Меркулов Ю.А.	Mozilla Firefox и Thunderbird (+ CD)	272 с.
Молочков В.П.	Nero 7 Premium: запись CD и DVD. Самоучитель (+CD)	336 с.
Молочков В.П.	Nero 8. Запись CD и DVD + Дистрибутив (на CD-ROM)	400 с.
Молочков В.П.	Photoshop CS3 для фотографов и дизайнеров (+ Видеокурс на DVD)	288 с.
Кузнецов М.А.	PHP 5	560 с.
Кирьянов Д.В.	Pinnacle Studio 9	320 с.
Дударева Н.Ю.	SolidWorks 2006 (+ CD)	336 с.
Леоненков А.В.	UML - 2 издание, доп.	432 с.
Гарнаев А.	VBA - 2 издание	560 с.
Карпов Р.Г.	Visual Basic .NET	592 с.
Шевякова Д.А.	Visual Basic 2005	576 с.
Ананьев А.	Visual Basic 6	624 с.
Омельченко Л.	Visual FoxPro 8	688 с.
Омельченко Л.	Visual FoxPro 9.0	608 с.
Гарнаев А.	Visual Studio .NET 2003	688 с.
Дунаев В.В.	Web-графика: нужные программы. Самоучитель	608 с.
Ощенко И.А.	Азбука программирования в 1С:Предприятие 7.7 + CD-ROM	528 с.
Соломенчук В.	Аппаратные средства персональных компьютеров	512 с.
Жуков А.	Ассемблер (+ прил. на дискете)	448 с.
Олифер К.В.	Без опасности в Windows XP	480 с.
Фрузоров С.Н.	Бесплатные разговоры через интернет (+CD)	464 с.
Герасевич В.	Блоги и RSS. Интернет-технологии нового поколения	256 с.
Березин С.В.	Ваш выход в Интернет. Секреты эффективной и безопасной работы	592 с.
Кузнецов И.	Видео на ПК (+ CD)	416 с.
Кирьянов Д.В.	Видеоанимация: After Effects, Premiere Pro, Flash. Самоучитель (+ Видеокурс на CD)	256 с.
Дунаев В.	Графика для Web	640 с.

Колисниченко Д.Н.	Движок для вашего сайта. CMS Joomla!, Slaed, PHP-Nuke(+CD-ROM)	368 с.
Молочков В.	Издательство на компьютере	736 с.
Колисниченко Д.Н.	Интернет: от "чайника" к пользователю.-2-е изд., перераб. и доп. (+CD)	528 с.
Герасевич В.	Компьютер для врача - 2 издание	512 с.
Сибрина Т.П.	Компьютер для офиса. Приемы грамотной и эффективной работы	528 с.
Комолова Н.	Компьютерная верстка и дизайн	512 с.
Дунаев В.В.	Основы Web-дизайна	512 с.
Медников В.	Основы компьютерной музыки	336 с.
Культин Н.Б.	Основы программирования в Delphi 2006 для Microsoft .NET Framework (+CD)	464 с.
Культин Н.Б.	Основы программирования в Delphi 2006 для Windows. Самоучитель (+ CD)	384 с.
Культин Н.Б.	Основы программирования в Delphi 2007. + DVD	480 с.
Культин Н.Б.	Основы программирования в Delphi 7 (+ CD)	608 с.
Культин Н.	Основы программирования в Delphi 8 для Microsoft .NET Framework (+ CD)	400 с.
Культин Н.Б.	Основы программирования в Delphi.-2-е изд., перераб. и доп. (+CD)	640 с.
Гаевский А.Ю.	Основы работы в Интернете	380 с.
Кетков Ю.	Практика программирования Visual Basic, C++ Builder, Delphi (+ прил. на дискете)	464 с.
Кетков Ю.	Практика программирования Бейсик, Си, Паскаль (+ прил. на дискете)	480 с.
Иванов Н.Н.	Программирование в Linux (+CD)	416 с.
Культин Н.Б.	Программирование в Turbo Pascal 7.0 и Delphi: 3-е изд., перераб. и доп.(+CD-ROM)	400 с.
Яковлев А.	Раскрутка и продвижение сайтов: основы, секреты, трюки	336 с.
Миловская О.С.	Самоучитель 3ds Max 2008. + Видеокурс (на DVD)	336 с.
Миловская О.С.	Самоучитель 3ds Max 2009	336 с.
Пекарев Л.Д.	Самоучитель 3ds Max 8	432 с.
Миловская О.С.	Самоучитель 3ds Max 9 (+ Видеокурс на CD)	256 с.
Бекаревич Ю.Б.	Самоучитель Access 2007 (+ Видеокурс на CD)	720 с.
Пономаренко С.И.	Самоучитель Adobe Acrobat 8. Формат PDF и печать (+ CD)	304 с.
Кирьянов Д.В.	Самоучитель Adobe After Effects CS3 + Видеокурс (на CD-ROM)	384 с.
Уллман Л.	Самоучитель Adobe AIR. Разработка приложений с помощью Ajax	560 с.

Осипова О.Г.	Самоучитель Adobe Dreamweaver CS3. + Видеокурс (на CD-ROM)	400 с.
Слепченко К.	Самоучитель Adobe Flash CS3 + Видеокурс (на CD)	544 с.
Федорова А.В.	Самоучитель Adobe Illustrator CS2	480 с.
Агапова И.В.	Самоучитель Adobe InDesign CS2 (+ CD)	304 с.
Агапова И.В.	Самоучитель Adobe InDesign CS3. + Видеокурс (на CD-ROM)	336 с.
Тучкевич Е.И.	Самоучитель Adobe Photoshop CS3 (+CD-ROM)	496 с.
Кирьянов Д.В.	Самоучитель Adobe Premier Pro 2.0 (+ Видеокурс на CD)	240 с.
Кирьянов Д.В.	Самоучитель Adobe Premiere Pro CS3 + видеокурс (на CD-ROM)	240 с.
Полещук Н.Н.	Самоучитель AutoCAD 2008	704 с.
Полещук Н.Н.	Самоучитель AutoCAD 2009. Двухмерное проектирование	544 с.
Полещук Н.Н.	Самоучитель AutoCAD 2009. Трехмерное проектирование	416 с.
Красноперов С.В.	Самоучитель Autodesk Inventor + Видеокурс (на CD)	576 с.
Трасковский А.В.	Самоучитель BIOS.-2е изд. перераб. и доп.	448 с.
Пахомов Б.И.	Самоучитель C/C++ и Borland C++Builder 2006	576 с.
Комолова Н.В.	Самоучитель CorelDRAW X4 (+CD-ROM)	656 с.
Хомоненко А.Д.	Самоучитель Delphi .NET	464 с.
Хомоненко А.Д.	Самоучитель Delphi. 2-е изд. + CD-ROM	576 с.
Долженков В.А.	Самоучитель Exel 2007. ил. + Видеокурс (на CD-ROM)	544 с.
Чебыкин Р.	Самоучитель HTML и CSS. Современные технологии.	624 с.
Хабибуллин И.	Самоучитель Java.-3-е изд., перераб. и доп. СПб.	768 с.
Рамел Д.	Самоучитель Joomla!: Пер. с англ.	448 с.
Колисниченко Д.Н.	Самоучитель Linux openSUSE 10.3 + Дистрибутив (на DVD)	416 с.
Колисниченко Д.Н.	Самоучитель Linux openSUSE 11 + Дистрибутив (на DVD)	432 с.
Колисниченко Д.Н.	Самоучитель Linux. 2-е изд.	448 с.
Жилинский А.А.	Самоучитель Microsoft SQL Server 2008	240 с.
Сидорина Т.Л.	Самоучитель Microsoft Visual Studio C++ и MFC (+CD)	848 с.
Омельченко Л.Н.	Самоучитель Microsoft Windows Vista (+ Видеокурс на CD)	624 с.
Кузнецов М.В.	Самоучитель MySQL 5 (+CD)	560 с

Кузнецов М.В.	Самоучитель PHP 5/6. -3-е изд., перераб. и доп.	672 с.
Кирьянов Д.В.	Самоучитель Pinnacle Studio Plus 11 (+ Видеокурс на CD-ROM)	336 с.
Яковлева Е.С.	Самоучитель Skype. Бесплатная связь через Интернет	304 с.
Крюков Д.А.	Самоучитель Slackware/MOPSLinux для пользователя (+ CD)	272 с.
Дударева Н.Ю.	Самоучитель SolidWorks 2008. + CD-ROM.	384 с.
Леоненков А.В.	Самоучитель UML 2	576 с.
Шевякова Д.А.	Самоучитель Visual Basic 2008 + Дистрибутив (на DVD)	592 с.
Рудикова Л.В.	Самоучитель Word 2007. + Видеокурс (на CD-ROM)	672 с.
Герасимов А.А.	Самоучитель КОМПАС-3D V9. Двумерное проектирование (+CD)	592 с.
Герасимов А.А.	Самоучитель КОМПАС-3D V9. Трехмерное проектирование. + демо-версия дистрибутив + CD-ROM	400 с.
Скрылина С.Н.	Самоучитель работы на Macintosh (+CD)	688 с.
Ревич Ю.В.	Самоучитель работы на ПК для всех (+ CD)	752 с.
Ревич Ю.В.	Самоучитель работы на ПК для всех.-2-е изд., перераб. и доп. (+CD)	512 с.
Пахомов Б.И.	Самоучитель C/C++ и C++ Builder 2007 + Дистрибутив (на DVD)	672 с.
Кенин А.М.	Самоучитель системного администратора	464 с.
Трасковский А.В.	Сбои и неполадки домашнего ПК: -2-е изд., перераб. и доп.	512 с.
Стахнов А.А.	Сеть для офиса и Linux-сервер своими руками	320 с.
Поляк-Брагинский А.В.	Сеть своими руками. 3-е изд.	640 с.
Немнюгин С.	Современный Фортран	496 с.
Боев В.Д.	Справочная правовая система КонсультантПлюс. Самоучитель (+ CD)	208 с.
Дунаев В.В.	Сценарии для Web-сайта. PHP и JavaScript	576 с.
Дунаев В.В.	Сценарии для Web-сайта: PHP и JavaScript. 2-е изд.	576 с.
Мур М.	Телекоммуникации. Руководствводство для начинающих	624 с.
Мотев А.А.	Уроки MySQL (+ CD)	208 с.
Ощенко И.А.	Учимся работать на компьютере. — 2-е изд. + Видеокурс (на CD-ROM)	464 с.
Ревич Ю.В.	Цифровая фотография на практике	368 с.

Культин Никита Борисович, кандидат технических наук, доцент Санкт-Петербургского государственного политехнического университета, где читает курс «Теория и технология программирования». Автор серии книг по программированию в Turbo Pascal, Delphi, C++ Builder и Visual Basic, которые вышли общим тиражом более 350 тыс. экземпляров.

Microsoft®

Visual C#

в задачах и примерах

Если вы хотите научиться программировать в Microsoft Visual C#, то эта книга для вас. В ней вы найдете хорошо документированные примеры программ – от простейших, демонстрирующих назначение базовых компонентов, до приложений работы с файлами, XML-документами, графикой и базами данных Microsoft Access и Microsoft SQL Server Compact Edition, раскрывающих тонкости программирования в Microsoft Visual C#. Приведены примеры использования технологии LINQ. Предлагаемые для самостоятельного решения задачи позволят закрепить полученные знания.

Компакт-диск содержит проекты, представленные в книге

БХВ-Петербург
190005, Санкт-Петербург,
Измайловский пр., 29
E-mail: mail@bhv.ru
Internet: www.bhv.ru
Тел.: (812) 251-42-44
Факс: (812) 320-01-79

ISBN 978-5-9775-0410-2